University of California

Los Angeles

Splitting Algorithms for Convex Optimization and Applications to Sparse Matrix Factorization

A thesis submitted in partial satisfaction

of the requirements for the degree

Master of Science in Electrical Engineering

by

Rong Rong

2013

Abstract of the Thesis

Splitting Algorithms for Convex Optimization and Applications to Sparse Matrix Factorization

by

Rong Rong

Master of Science in Electrical Engineering

University of California, Los Angeles, 2013

Professor Lieven Vandenberghe, Chair

Several important applications in machine learning, data mining, signal and image processing can be formulated as the problem of factoring a large data matrix as a product of sparse matrices. Sparse matrix factorization problems are usually solved via alternating convex optimization methods. These methods involve at each iteration a large convex optimization problem with non-differentiable cost and constraint functions, which is typically solved by block coordinate descent algorithm. In this thesis, we investigate first-order algorithms based on forward-backward splitting and Douglas-Rachford splitting algorithms, as an alternative to the block coordinate descent algorithm. We describe efficient methods to evaluate the proximal operators and resolvents needed in the splitting algorithms. We discuss in detail two applications: Structured Sparse Principal Component Analysis and Sparse Dictionary Learning. For these two applications, we compare the splitting algorithms and block coordinate descent on synthetic data and benchmark data sets. Experimental results show that several of the splitting methods, in particular Tseng's modified forward-backward method and the Chambolle-Pock splitting method, are often faster and more accurate than the block coordinate descent algorithm.

The thesis of Rong Rong is approved.

Lara Dolecek

Vwani Roychowdhury

Lieven Vandenberghe, Committee Chair

University of California, Los Angeles

2013

Table of Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Sparse Matrix Factorization

Matrix factorization problems are widely encountered in several fields, including signal processing [CDS98], statistical and machine learning [Tib96, XLG03], pattern classification [LHZ01], and image processing [EA06, BBL07]. They can be seen as extensions of the singular value decomposition (SVD) with additional properties imposed on the matrix factors. *Sparse* matrix factorization problems arise in sparse coding in signal processing [LBR07], and sparse principal component analysis (PCA) [ZHT06]. In other applications such as Structured Sparse PCA [JOB10] or hierarchical dictionary learning [JMO10], certain sparsity structures or preferred sparsity patterns are imposed as well. There are many other types of modified frameworks of matrix factorizations, such as penalized matrix decomposition (PMD) [WTH09], minimum cardinality relaxations and approximations [BMP08], and non-negative matrix factorization (NMF) [DLJ06]. Similar to sparse matrix factorization, these matrix factorization problems are extremely complex and non-convex. Algorithms for these matrix factorization problems do not give guarantees of convergence to global optimal solutions.

Many of these applications can be formulated as approximating a matrix by a product of two matrices with certain sparsity patterns. More precisely, given a matrix $X \in \mathbb{R}^{n\times p}$, the goal is to find an approximation $\hat{X} \approx X$ of the form

$\hat{X} = UV^T, U \in \mathbb{R}^{n\times r}, V \in \mathbb{R}^{p\times r}$, such that the matrix factors U and/or V are sparse. The number of columns r for both matrices depends on the specific applications and conditions on the sparsity pattern.

To formulate the sparse matrix factorization as a minimization problem, we approximate the matrix X by $\hat{X} = UV^T$, such that the square Frobenius norm of $X - \hat{X}$ is minimized with additional convex penalties or constraints on U and V that promote sparsity. More precisely, consider the problem

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X - UV^T\|_F^2 + \lambda g(V) \\ \text{subject to} \quad & h(U) \leq 1. \end{aligned} \tag{1.1}$$

The parameter $\lambda > 0$ controls the level of regularization: Choosing $\lambda = 0$ solves the rank r matrix factorization exactly while choosing larger λ introduces more regularization. The choice of the regularization functions $g(V)$ and $h(U)$ depends on the applications and the desired type of sparsity patterns.

Generally it is hard to directly impose sparsity constraints. It requires knowledge of the exact sparsity pattern. In general the formulation is non-convex and scales poorly with problem size. Instead, applying additional convex penalties that indirectly promote sparsity is much more efficient in terms of problem formulation and scalability. For example, the typical lasso type penalty [Tib96] applying as the matrix l_1 norm

$$g(V) = \|V\|_1 = \sum_i \sum_j |V_{i,j}|$$

promotes element-wise sparsity in the factorization matrix V. Witten et al. [WTH09] proposed that adding an l_1 penalty on the columns of V in least squares problem gives a sparse solution for the PCA problem. Furthermore, Jenatton et al. [JOB10] proposed that adding suitable l_1-l_2 penalties on the columns of V introduces structured sparsity in the principal components according to the predefined structure.

Mairal et al. [MBP09] also proposed that adding suitable penalties on the rows of V gives a factorization of X with a sparse decomposition matrix factor, which can be used to solve sparse dictionary coding problems.

In this thesis, we focus on column-wise sparsity or group sparsity using l_1, l_1-l_2 or l_1-l_∞ norms as regularization penalty. They are all targeted at promoting sparsity pattern with certain structures. Namely, consider a more specific formulation from (Eq. 1.1):

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_k g(V^k) \\ &\text{subject to} \quad h(U^k) \le 1, \quad \forall k, \end{aligned} \tag{1.2}$$

where V^k and U^k are the k-th columns of matrix V and U, respectively. The convex penalty or constraint functions

$$g(V^k) = \sum_{\alpha \in A} \|V_\alpha^k\|_p, \qquad h(U^k) = \sum_{\beta \in B} \|U_\beta^k\|_q, \tag{1.3}$$

with $p, q \in \{2, \infty\}$, are considered as group l_1 norms. A and B are groups of index sets based on prior knowledge of sparsity structure of matrix U and V, such that $\alpha \in A$ and $\beta \in B$ are index set whose elements belong to the same sparsity group.

In the next section, we discuss algorithms developed for problem (Eq. 1.1).

1.2 Previous Approaches and Algorithms

The optimization problem (Eq. 1.1) is non-convex and hence extremely hard to solve in general. However, there are some important properties of this optimization problem.

The most important one is bi-convexity. The objective function itself in (Eq. 1.1) is not convex jointly in U and V. However, for either U or V alone, it is a regularized quadratic minimization problem if the other variable is fixed: For a fixed V, the objective is just a quadratic function of U with a domain constraint,

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|X - UV^T\|_F^2 \\ &\text{subject to} \quad h(U^k) \leq 1, \quad \forall k. \end{aligned} \tag{1.4}$$

As long as the domain defined by $h(U^k) \leq 1$ is convex, this objective is convex in U for a fixed V. Similarly, for a fixed U, the objective is a penalized quadratic function,

$$\text{minimize} \quad \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_k g(V^k). \tag{1.5}$$

It is also convex in V for a fixed U if the regularization function $g(V^k)$ is convex.

The other important property of (Eq. 1.1) or (Eq. 1.2) is non-differentiability: In many applications, the constraints and/or penalty functions (Eq. 1.3) are non-differentiable. In sparse matrix factorization, it is hard to use gradient type descent methods since the gradient is not defined on the entire domain. Fortunately, there are many ways to handle non-differentiability in various different algorithms for both of the convex optimization problem in (Eq. 1.4) and (Eq. 1.5).

Since this optimization problem is convex in U for a fixed V and vice versa, it is simple to consider one variable at a time. Therefore, an algorithm called alternating descent is a promising way to achieve local optimality. This alternating descent algorithm works for various matrix factorization problems [Eld07, JOB10]. For sparse matrix factorization problem in (Eq. 1.2), this algorithm can be formulated as:

$$\begin{aligned} U^+ &= \underset{U:h(U^k)\leq 1}{\arg\min} \left\{ \frac{1}{2}\|X - UV^T\|_F^2 \right\}, \\ V^+ &= \underset{V}{\arg\min} \left\{ \frac{1}{2}\|X - U^+V^T\|_F^2 + \lambda \sum_k g(V^k) \right\}. \end{aligned} \tag{1.6}$$

The alternating descent algorithm just solves the two separate minimization problems in an alternating manner. One can use different approaches for each of the two sub-problems in (Eq. 1.6) to handle the non-differentiability of the constraint and penalty term. In fact, various different approaches have been proposed to solve these two convex sub-problems.

1.2.1 Block Coordinate Descent

In many applications, the variables are in some way separable depending on the regularization penalty and constraint. For example, column-wise sparsity penalties (Eq. 1.3) are independent between columns. Jenatton et al. [JOB10] proposed a block coordinate descent (BCD) method for Structured Sparse Principal Component Analysis. Similarly Jenatton et al. [JMO10] also proposed a block coordinate ascent method, which is BCD applied to the dual problem, for sparse hierarchical dictionary learning. Block coordinate descent algorithm updates variables in each block in a cyclic manner, by solving a smaller problem of the desired variables while fixing the other.

BCD explicitly exploits the separable structure of the penalty and constraint in the matrix factorization problem. Many of the coordinates or variables are formulated in groups which are independent across different groups. Each inner iteration of BCD deals with one group of variables at a time and consider the other variables constant in order to reduce complexity. This makes BCD a fast algorithm and easy to implement. Since BCD updates one group of variables at a time, it can handle more complex, non-convex penalties as long as the problem for each inner iteration remains easy and fast to solve. Namely, Jenatton et al. [JOB10] proposed a method to handle non-convex l_p norm where $p < 1$ for (Eq. 1.3) in the convex sub-problems for alternating descent.

On the outer iteration, there are some heuristic strategies to arrange the order in which the block variables are being updated. Yet there is no general convergence analysis. There is no guaranteed bounds on convergence rate because BCD highly depends on application, input data, and the strategy of ordering. For some applications, these strategies are limited. Moreover, BCD does not provide certificates of optimality since it minimizes a subset of variables at a time. In fact, there are no guarantee of local optimality for each sub-problem in (Eq. 1.6). Even if it converges to local optimum, there is no way to examine or verify such optimality. Many applications of BCD simply use a fixed number of iterations without stopping condition [JOB10].

1.2.2 Proximal Point Methods

Both sub-problems in (Eq. 1.6) are least squares problems with a regularization penalty or constraint. It is hard to minimize the objective function due to the non-differentiable regularization terms. The proximal point algorithm (PPA) [G91] is usually considered suitable for these type of problems. Starting from an initial point, PPA solves a convex optimization problem obtained by adding to the objective a quadratic penalty of the distance to the point from previous iteration. It iteratively solves them until the point is close enough to the optimum of the original problem. Some famous algorithms are special cases of PPA, such as augmented Lagrangian methods and proximal methods of the multipliers.

Each iteration of PPA updates all variables at the same time. No update sequence or ordering of variables is necessary for PPA. In addition, PPA only requires one simple update for every iteration, which makes it perfectly simple. In addition, PPA guarantees rate of convergence for convex objective and provides an easy way to verify optimality if certain conditions are satisfied. In the alternating descent algorithm (Eq. 1.6), it is important to know the error from the exact global optimum when solving each of the two convex sub-problems.

On the other hand, PPA solves at each iteration a convex optimization problem obtained by adding a quadratic penalty. Normally this quadratic penalty is added to handle non-differentiable regularizations. It does not reduce complexity of the objective. In fact, most of the problems in real applications do not have easy or closed form solutions. Therefore, this convex optimization problem for each iteration of PPA might be very expensive and sometimes impossible to solve.

1.3 Splitting Algorithms

The splitting algorithms are inspired by the advantages and difficulties of PPA. It splits the original objective in to two parts and solves two convex optimization problems obtained by adding quadratic penalties similar to PPA, which are much easier to solve in many cases [CP10]. Namely, if the original objective can be separated into two simple functions, as in

$$\text{minimize} \quad f(x) + g(x),$$

then the splitting algorithms provide such an iterative update method that deals with the two functions $f(x)$ and $g(x)$ separately. As long as the two separate functions remain simple, these splitting algorithms do not require additional properties such as block separability. For sparse matrix factorization problem (Eq. 1.1), the splitting algorithms only require the penalty and constraint functions to be simple. However, BCD could only deal with problems like (Eq. 1.2) since it requires the column-wise separability.

The splitting algorithms make it possible to deal with complex coupled objective functions that PPA cannot handle in a reasonable complexity. However, they require to solve two simple problems in each iteration. These two problems have to be very cheap to evaluate and usually one or both of them must have closed form solutions. In the sparse matrix factorization problem, the Frobenius norm square term in (Eq. 1.1) is a simple, strongly convex function. Splitting algorithms are excellent choices if the penalty and constraint are simple. As an example, the column-wise sparsity penalties in (Eq. 1.3) certainly fit in this category.

1.4 Outline of Thesis

The remainder of this thesis is organized as follows: Convex optimization background is introduced in Chapter 2, where we briefly discuss resolvent and monotone operator followed by splitting algorithms. In Chapter 3 and Chapter 4, we will discuss two specific applications of sparse matrix factorization: Structured Sparse Principal Component Analysis [JOB10] and Sparse Hierarchical Dictionary Learning [JMO10]. At last, we discuss possible further applications of sparse matrix factorization as well as conclude our contributions in Chapter 5.

CHAPTER 2

Proximal Operators and Splitting Algorithms

In this chapter, we introduce convex optimization background related to the alternating descent algorithm in (Eq. 1.6) for sparse matrix factorization. First we discuss the regularization penalties and constraints (Eq. 1.3) and their properties for convex optimization algorithms. Then we introduce the optimality conditions and various splitting algorithms for a general convex optimization problem which includes both convex optimization sub-problems that arise in the alternating descent algorithm.

2.1 Sparsity Promoting Penalty

The sparsity promoting penalty term in problem (Eq. 1.5), one of the convex sub-problems in (Eq. 1.6), is essential in sparse matrix factorization. It is well known that the l_1 norm penalty or constraint promotes sparsity. Applying the l_1 regularization to least squares problem leads to the well-known lasso formulation [Tib96]. Correspondingly, applying the l_1-l_2 norm penalty described in (Eq. 1.3) leads to group or overlapping group lasso formulation [FHT10]. The l_1-l_2 norm promotes sparsity over all variables in the same group. Consider the l_1-l_2 norm,

$$g(x) = \sum_{\alpha \in A} \|x_\alpha\|_p. \tag{2.1}$$

It is defined by the index sets group A. One simple example is the l_1 norm in $\mathbb{R}^n$. Each element forms its own group: $A = \{\{1\}, \{2\}, \ldots, \{n\}\}$. This is an

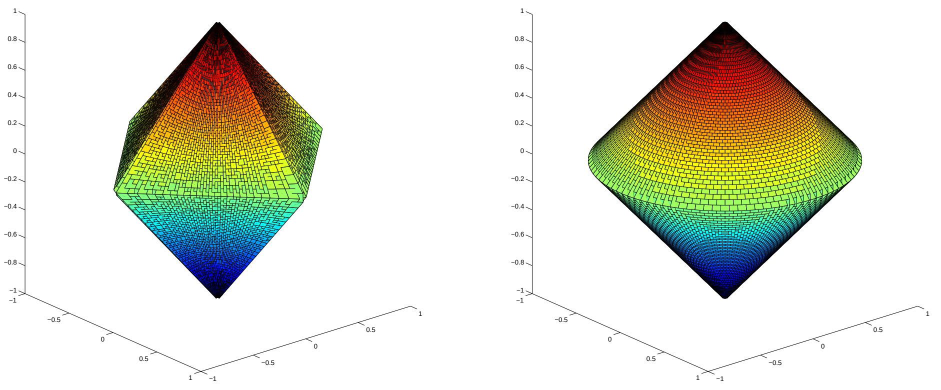


Figure 2.1: *The l_1 norm ball (left) and the group l_1-l_2 norm ball (right) in 3 dimensional space.*

element-wise sparse penalty, and it prefers the extreme points of the l_1 norm ball. See its level set in 3 dimensional space in Figure 2.1. Another example is the group l_1-l_2 norm with $A = \{\{1\}, \{2,3\}\}$. The level set $\{x|g(x) = 1\}$ is shown in Figure 2.1 for

$$\begin{aligned} g(x) &= \sum_{\alpha \in A} \|x_\alpha\|_2 \\ &= \|x_{\{1\}}\|_2 + \|x_{\{2,3\}}\|_2 \\ &= |x_1| + \sqrt{x_2^2 + x_3^2} \\ &= 1. \end{aligned}$$

It prefers the boundary of the disk where $x_1 = 0$ and the two conic extreme points where $x_2 = x_3 = 0$. In other words, either the first or the second group of variables are set to be sparse.

In general, one variable can appear in different groups. For example, consider $A = \{\{1\}, \{1,2\}, \dots, \{1,2,\dots,n\}\}$. Figure 2.2 shows the level set $\{x|g(x) = 1\}$ for this overlapping group l_1-l_2 norm in 3 dimensional space.

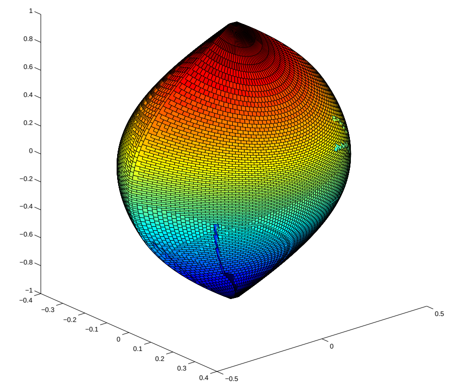


Figure 2.2: *The overlapping group l_1-l_2 norm ball in 3 dimensional space.*

Additionally, some linear scaling or weight functions can be imposed on each group. This leads to penalty functions of the form

$$g(x) = \sum_{\alpha \in A} \|D_\alpha x_\alpha\|_p, \tag{2.2}$$

where D_α is a square diagonal weight matrix. There are many different ways to determine weights for a given index set α. A straightforward example is the group cardinality weight matrix

$$D_\alpha = |\alpha| I.$$

This kind of weight matrix is used to give larger groups more weight.

Based on the prior knowledge of the problem, the desired sparsity pattern, and other rules from the applications, the index set and weight matrix can be very different so that this function (Eq. 2.2) is suitable to be used as penalty or constraint for various different applications. However, the formulation of this function remains simple regardless of how the group of index sets is formed.

2.2 Optimality Conditions

Recall the two convex sub-problems (Eq. 1.4) and (Eq. 1.5) for alternating descent algorithm (Eq. 1.6),

$$\begin{aligned}
&\underset{U}{\text{minimize}} \quad \frac{1}{2}\|X - UV^T\|_F^2 \qquad \text{s.t. } h(U^k) \leq 1, \quad \forall k. \\
&\underset{V}{\text{minimize}} \quad \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_k g(V^k).
\end{aligned}$$

Consider the convex constraints $h(U^k) \leq 1$ as indicator functions, which takes 0 as function value when it is feasible and $+\infty$ otherwise. Then both of the sub-problems are obtained by adding a regularization to the least squares problem. Namely, consider a general optimization problem

$$\text{minimize} \quad f(x) + g(Cx). \tag{2.3}$$

Function f and g are closed, convex with non-empty domains. The two convex sub-problems are just special cases of (Eq. 2.3) with f as a quadratic function and g as a regularization function.

2.2.1 Primal Optimality

Consider the problem (Eq. 2.3). The optimality condition for this problem is

$$0 \in \partial f(x) + C^T \partial g(Cx). \tag{2.4}$$

It is called a monotone inclusion problem. In (Eq. 2.4), ∂f stands for the subdifferential of function f, such that vector $y \in \partial f(x)$ if and only if

$$f(x') \geq f(x) + y^T(x' - x), \quad \forall x' \in \mathbf{dom} f. \tag{2.5}$$

Generally $\partial f(x)$ is not unique for non-smooth functions. When f is differentiable at x, it is equivalent to a singleton with its value equals to the gradient of f at x:

$\partial f(x) = \{\nabla f(x)\}$. If f is smooth over the entire domain, the optimality condition (Eq. 2.4) simply means the gradient of the objective function is zero. When both f and g are convex, a solution x for this problem is optimal if and only if it satisfies the condition (Eq. 2.4) [BV04].

2.2.2 Dual Optimality

Consider the convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(x) + g(y) \\ \text{subject to} & y = Cx. \end{array} \tag{2.6}$$

It is equivalent to the problem (Eq. 2.3). The Lagrangian of (Eq. 2.6) is

$$L(x, y, z) = f(x) + g(y) + z^T(Cx - y). \tag{2.7}$$

The Lagrange dual function [BV04] is obtained by minimizing the Lagrangian (Eq. 2.7) over the primal variables x and y:

$$\begin{aligned} \phi(z) &= \inf_{x,y} L(x, y, z) \\ &= \inf_{x} \left\{ f(x) + (C^T z)^T x \right\} + \inf_{y} \left\{ g(y) - z^T y \right\} \\ &= -f^*(-C^T z) - g^*(z). \end{aligned} \tag{2.8}$$

In (Eq. 2.8), the function f^* is defined as

$$f^*(x) = \sup_{u \in \mathbf{dom} f} \left\{ x^T u - f(u) \right\}. \tag{2.9}$$

It is called the Fenchel conjugate function [Fen49] of f. The Fenchel conjugate function is always closed and convex even though the original function is not convex. The Lagrangian dual of the problem (Eq. 2.6) is to maximize the Lagrange dual function $\phi(z)$:

$$\text{maximize} \quad -f^*(-C^T z) - g^*(z). \tag{2.10}$$

Its optimality condition is

$$0 \in -C\partial f^*(-C^T z) + \partial g^*(z). \tag{2.11}$$

The primal dual pair of this problem is similar with each other both in formulations (Eq. 2.3, 2.10) and in optimality conditions (Eq. 2.4, 2.11). One can easily transfer analysis and algorithms for the primal problem to the dual problem.

2.2.3 Primal-Dual Optimality

The Karush-Kuhn-Tucker (KKT) conditions [BV04] of the problem (Eq. 2.6) are

$$\begin{aligned} 0 &\in \partial_x L(x^*, y^*, z^*) = \partial f(x^*) + C^T z^* && (2.12) \\ 0 &\in \partial_y L(x^*, y^*, z^*) = \partial g(y^*) - z^* && (2.13) \\ 0 &= y^* - Cx^*. && (2.14) \end{aligned}$$

(Eq. 2.12) and (Eq. 2.13) are the optimality condition of the Lagrangian (Eq. 2.7) over the primal variables. (Eq. 2.14) guarantees that the constraint is satisfied at the optimum since a primal-dual pair (x^*, y^*, z^*) is optimal if and only if it is primal and dual feasible, the primal variables (x^*, y^*) are the minimizer of the Lagrangian, and the complementary slackness is satisfied. The KKT conditions can also be written in a single monotone inclusion problem:

$$0 \in \begin{bmatrix} 0 & 0 & C^T \\ 0 & 0 & -I \\ -C & I & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g(y) \\ 0 \end{bmatrix}. \tag{2.15}$$

This gives an optimality condition for the primal and dual problem together. Furthermore, applying a property of the Fenchel conjugate pair of convex functions [Van12]:

$$u \in \partial f(v) \iff v \in \partial f^*(u), \tag{2.16}$$

the above monotone inclusion problem can be further simplified as

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}. \tag{2.17}$$

The primal-dual optimality condition gives another way to look at the optimality of problem (Eq. 2.3). The convex optimization algorithms that work with the primal-dual optimality condition usually update primal and dual variables at the same time. The Lagrangian dual function (Eq. 2.8) value for any dual feasible variable always gives a lower bound on the optimal value of the primal [BV04]. It provides a stopping criterion almost without extra cost.

2.3 Proximal Operators and Resolvent

For a general problem formulated as (Eq. 2.3), a variable is optimal if and only if the optimality conditions are satisfied. Therefore, finding a solution that satisfies either one of the optimality conditions gives the solution to the original problem. However, instead of solving an equation, one needs to find one solution that satisfies a monotone inclusion problem because the objective function might not be smooth. The definition of proximal operator and resolvent are important to address the non-smoothness before finding the solution to any monotone inclusion problem.

2.3.1 Monotone Operator and Resolvent

An operator F is defined as a set function that maps a vector to a set of vectors of the same dimension, or

$$x \in \mathbb{R}^n \quad \to \quad F(x) \subset \mathbb{R}^n.$$

The graph of an operator F is defined as

$$\mathbf{gr}(F) = \{(x, y) \in \mathbb{R}^n \times \mathbb{R}^n | y \in F(x)\}.$$

For example, a subdifferential ∂f is an operator that maps a point within the domain of f to any vector that satisfies (Eq. 2.5). The graph of the subdifferential ∂f is any two vectors (x, y) such that y is an element in the subdifferential ∂f evaluated at point $x \in \mathbf{dom} f$. An operator F is monotone if

$$(y - \hat{y})^T (x - \hat{x}) \geq 0, \quad \forall x, \hat{x} \in \mathbf{dom} F, y \in F(x), \hat{y} \in F(\hat{x}).$$

An operator F is maximal monotone if the graph of F is not properly contained in the graph of another monotone operator [Van12]. Monotone inclusion problem is to find the solution to

$$0 \in F(x). \tag{2.18}$$

The maximal monotonicity of F implies that the solution to the monotone inclusion problem is a closed convex set. The type of monotone operators discussed in this thesis are subdifferentials and skew-symmetric linear operators. The subdifferential of a proper closed convex function $F(x) = \partial f(x)$ is maximal monotone. A linear operator $F(x) = Mx$ is a single-valued maximal monotone operator if $M + M^T \succeq 0$ [Van12]. Obviously a skew-symmetric linear operator is maximal monotone because $M + M^T = 0$ for a skew-symmetric matrix M.

The resolvent of an operator F is another operator. For any $t > 0$, the resolvent of F is defined as

$$x \in \mathbb{R}^n \quad \to \quad (I + tF)^{-1}(x) \subset \mathbb{R}^n$$

Note that the inverse sign means the inverse of the operator,

$$y \in F^{-1}(x) \quad \Longleftrightarrow \quad x \in F(y).$$

Its definition is closely related to the monotone inclusion problem of operator F:

$$x \in (I + tF)^{-1}(\hat{x}) \quad \Longleftrightarrow \quad \frac{\hat{x} - x}{t} \in F(x).$$

If F is a maximal monotone operator, the resolvent is a single-valued operator and is defined for all x [Van12]. This indicates that even if an operator is not singleton, its resolvent is still a singleton and defined on the entire space by maximal monotonicity. This is a very important property as it enables all proximal-based methods.

As a special case, the resolvent of a skew-symmetric linear operator $F(x) = Mx$ is

$$x = (I + tM)^{-1}(\hat{x}),$$

in which the inverse of the operator is just the matrix inversion becase matrix M is skew-symmetric and $M + M^T = 0$ by definition. In practice, matrix M is usually large but structured such that there exist many efficient ways to calculate the solution.

As discussed before, we are interested in the monotone inclusion problem involving operators such as subdifferential and skew-symmetric linear operators. In the rest of this section, we work out the detail of the resolvent for subdifferential.

2.3.2 Proximal Operator

Consider the subdifferential operator of a closed convex function $\partial f(x)$. Its resolvent is equivalent to a strongly convex minimization problem:

$$\begin{aligned} x \in (I + t\partial f)^{-1}(\hat{x}) \quad &\Longleftrightarrow \quad \hat{x} \in x + t\partial f(x) \\ &\Longleftrightarrow \quad 0 \in \partial f(x) + \frac{1}{t}(x - \hat{x}) \\ &\Longleftrightarrow \quad x = \arg\min_x \left(f(x) + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right). \end{aligned} \tag{2.19}$$

This defines the proximal operator of function f. For any $t > 0$,

$$\operatorname{prox}_{tf}(\hat{x}) = \arg\min_x \left(f(x) + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right). \tag{2.20}$$

Since the proximal operator is the resolvent of subdifferential from (Eq. 2.19), the proximal operator is defined and unique for every $\hat{x}$. This property also follows from (Eq. 2.20), that the minimization problem is strongly convex hence the optimum is unique and attained in $\mathbf{dom} f$. The optimality condition of the minimization problem (Eq. 2.20) gives an element in the subdifferential $\partial f(x)$,

$$x = \operatorname{prox}_{tf}(\hat{x}) \quad \Longleftrightarrow \quad \frac{1}{t}(\hat{x} - x) \in \partial f(x).$$

It exists and is unique for all $\hat{x}$.

Interestingly, the proximal operator of the conjugate function f^* can be easily calculated from the proximal operator of f. Consider the Fenchel conjugate pair f and f^*, the Moreau decomposition [Van12] states that

$$x = \operatorname{prox}_f(x) + \operatorname{prox}_{f^*}(x), \quad \forall x,$$

which follows from the property of Fenchel conjugate in (Eq. 2.16),

$$\begin{aligned} u = \operatorname{prox}_f(x) \quad &\Longleftrightarrow \quad x - u \in \partial f(u) \\ &\Longleftrightarrow \quad u \in \partial f^*(x - u) \\ &\Longleftrightarrow \quad x - u = \operatorname{prox}_{f^*}(x). \end{aligned}$$

In practice, we are more interested in a more general extension of the Moreau decomposition:

$$x = \mathrm{prox}_{th*}(x) + t\mathrm{prox}_{h/t}(x/t).$$

This property is essential for proximal evaluation of conjugate function.

2.3.3 Augmented Lagrangian

Consider the subdifferential of the conjugate function $f^*(-C^T z)$ from (Eq. 2.8). Its resolvent is

$$\begin{aligned} z \in (I + tA)^{-1}(\hat{z}) \quad &\iff \quad 0 \in -C\partial f^*(-C^T z) + \frac{1}{t}(z - \hat{z}) \\ &\iff \quad z = \arg\min_z \left(f^*(-C^T z) + \frac{1}{2t}\|z - \hat{z}\|_2^2 \right). \end{aligned}$$

This monotone inclusion problem is very similar to the proximal operator (Eq. 2.19). It is also strongly convex and has a unique, attained optimum. The only difference is that the minimization problem involves a linear transformation. By introducing a new variable x and applying (Eq. 2.16) such that

$$x \in \partial f^*(-C^T z) \quad \iff \quad -C^T z \in \partial f(x),$$

the monotone inclusion problem can be written as

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ 0 \end{bmatrix} + \frac{1}{t} \begin{bmatrix} 0 \\ z - \hat{z} \end{bmatrix}. \tag{2.21}$$

This is an optimality condition to another minimization problem:

$$x = \arg\min_x \left(f(x) + \frac{t}{2}\|Cx - \hat{z}/t\|_2^2 \right) \tag{2.22}$$

$$z = \hat{z} + tCx. \tag{2.23}$$

One can find the resolvent via solving (Eq. 2.22) then simply applying the result to (Eq. 2.23).

2.4 Proximal Point Algorithm

Consider the optimality conditions (Eq. 2.4, 2.11, 2.15, 2.17). Each of them is a monotone inclusion problem for some maximal monotone operator F as in (Eq. 2.18). Sometimes this monotone inclusion problem is hard to solve directly. Instead, if F is maximal monotone, the optimal x^* of (Eq. 2.18) is a fixed point of the resolvent of F,

$$0 \in F(x^*) \quad \Longleftrightarrow \quad x^* = (I + tF)^{-1}(x^*) \tag{2.24}$$

for some $t > 0$. Finding the solution to (Eq. 2.18) is equivalent to finding a fixed point for the corresponding resolvent (Eq. 2.24). Starting from any feasible point x, the fixed point iteration of the resolvent is defined as

$$x^+ = (I + tF)^{-1}(x). \tag{2.25}$$

It is guaranteed to converge to the fixed point if F is monotone [Van12] with any positive $t > 0$. Generally t affects the rate of convergence. If we can find an algorithm that solves the resolvent efficiently, then we have an iterative algorithm to the fixed point problem with a trivial update (Eq. 2.25). This is known as the proximal point algorithm (PPA) [G91].

The crucial step in PPA is to find an efficient way to evaluate the resolvent. Consider the problem (Eq. 2.3). By choosing one of the optimality conditions discussed in Section 2.2, each iteration of PPA needs to solve a minimization problem to evaluate the resolvent. For example, consider the dual optimality condition (Eq. 2.11), the operator F_1 is

$$F_1(z) = -C\partial f^*(-C^T z) + \partial g^*(z).$$

The fixed point iteration of PPA derived from (Eq. 2.25) is

$$\frac{z - z^+}{t} \in C\partial f^*(-C^T z^+) + \partial g^*(z^+).$$

It is equivalent to solving the minimization problem

$$\begin{aligned}(x^+, y^+) &= \arg\min_{x,y} \left\{ f(x) + g(y) + \hat{z}^T(Cx - y) + \frac{t}{2}\|Cx - y\|_2^2 \right\} \\ z^+ &= \hat{z} + t(Cx^+ - y^+). \end{aligned} \tag{2.26}$$

The objective is obtained by adding a strongly convex term $\frac{t}{2}\|Cx - y\|_2^2$ to the Lagrangian of the equivalent problem (Eq. 2.6). This is the well known augmented Lagrangian method [Ber82].

Another example, consider the primal-dual optimality condition (Eq. 2.17). The operator F_2 is

$$F_2(x, z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}.$$

The fixed point iteration of PPA derived from (Eq. 2.25) is

$$\frac{1}{t}\begin{bmatrix} x - \hat{x} \\ z - \hat{z} \end{bmatrix} \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}.$$

It is equivalent to solving the minimization problem

$$\begin{aligned}(x^+, y^+) &= \arg\min_{x,y} \left\{ f(x) + g(y) + \hat{z}^T(Cx - y) + \frac{t}{2}\|Cx - y\|_2^2 + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right\} \\ z^+ &= \hat{z} + t(Cx^+ - y^+). \end{aligned} \tag{2.27}$$

It is similar to augmented Lagrangian problem but with an additional augmented term $\frac{1}{2t}\|x - \hat{x}\|_2^2$. This is known as the proximal method of multipliers [Van12].

In general, PPA solves a minimization problem at each iteration to update the fixed point variable. The complexity of the algorithm depends on the cost of solving this minimization problem.

2.5 Splitting Algorithms

PPA is definitely an easy algorithm to implement as it only involves a simple iterative update. However the underlying minimization problem for each iteration might be very difficult to solve. Take problem (Eq. 2.6) as an example. It is difficult to solve problems in (Eq. 2.26) or (Eq. 2.27), even with simple f and g. Therefore, many different techniques are introduced to split the optimality condition as a sum of two operators, so that the resolvent of each operator is much easier to compute. Consider any optimality condition in the format of

$$0 \in A(x) + B(x). \tag{2.28}$$

The splitting algorithms only require the resolvents of $A(x)$ and $B(x)$ but not the resolvent of the sum.

2.5.1 Douglas-Rachford Splitting Algorithm

The Douglas-Rachford splitting algorithm [EB92] is a very simple algorithm that dates back to the 1950s. Consider the monotone inclusion problem (Eq. 2.28). Starting from any feasible point z, the Douglas-Rachford splitting algorithm updates variables as following:

$$\begin{aligned} x^+ &= (I + tB)^{-1}(z) \\ y^+ &= (I + tA)^{-1}(2x^+ - z) \\ z^+ &= z + y^+ - x^+, \end{aligned}$$

with any $t > 0$. The Douglas-Rachford splitting algorithm requires the resolvent of A and B, but not $A + B$. Depending on the complexity to evaluate these two resolvents, it can be considerably faster than PPA. In general, one can choose any optimality conditions as the target monotone inclusion problem and this monotone inclusion problem can be split into a sum of any two operators. Therefore, there are many well-known implementations of the Douglas-Rachford splitting algorithm.

Consider the format of (Eq. 2.28) applied to the dual optimality condition (Eq. 2.11):

$$0 \in -C\partial f^*(-C^T z) + \partial g^*(z),$$

with

$$A(z) = -C\partial f^*(-C^T z), \qquad B(z) = \partial g^*(z).$$

The Douglas-Rachford Splitting algorithm can be represented as

$$\begin{aligned}
v^+ &= (I + tA)^{-1}(z - w) \\
&= \arg\min_v \left\{ f^*(-C^T v) + \frac{1}{2t}\|v - (z - w)\|_2^2 \right\} \\
z^+ &= (I + tB)^{-1}(v^+ + w) \\
&= \operatorname{prox}_{tg^*}(v^+ + w) \\
&= \arg\min_z \left\{ g^*(z) + \frac{1}{2t}\|z - (v^+ + w)\|_2^2 \right\} \\
w^+ &= w + v^+ - z^+.
\end{aligned}$$

This can be shown to be equivalent to the alternating direction method of multipliers (ADMM) [BPC11].

Consider the same format (Eq. 2.28) applied the primal-dual optimality condition (Eq. 2.17):

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix},$$

with

$$A(x, z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}, \qquad B(x, z) = \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}.$$

Instead of directly solving the linear equations, the first resolvent $(x, z) = (I + tA)^{-1}(\hat{x}, \hat{z})$ is equivalent to solving

$$
\begin{aligned}
x &= \arg\min_x \left(\frac{t}{2}\|Cx - \frac{\hat{z}}{t}\|_2^2 + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right) \\
z &= \hat{z} + tCx.
\end{aligned}
$$

It has a closed form solution as the minimization problem is just a least squares problem. The second resolvent $(x, z) = (I + tB)^{-1}(\hat{x}, \hat{z})$ is equivalent to solving

$$
\begin{aligned}
(x, y) &= \arg\min_{x,y} \left(f(x) + g(y) + \frac{t}{2}\|\frac{\hat{z}}{t} - y\|_2^2 + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right) \\
&= \arg\min_x \left(f(x) + \frac{1}{2t}\|x - \hat{x}\|_2^2 \right) + \arg\min_y \left(g(y) + \frac{t}{2}\|\frac{\hat{z}}{t} - y\|_2^2 \right) \\
&= \big(\mathrm{prox}_{tf}(\hat{x}), \mathrm{prox}_{g/t}(\hat{z}/t)\big) \\
z &= \hat{z} - ty \\
&= \hat{z} - t\mathrm{prox}_{g/t}(\hat{z}/t) \\
&= \mathrm{prox}_{tg^*}(\hat{z}).
\end{aligned}
$$

It evaluates the two proximal operators for functions f and g^* separately.

The step size in these Douglas-Rachford splitting methods is an arbitrary positive number. However, this step size is usually limited by numerical instability in practice.

2.5.2 Forward-Backward Algorithm

Consider the monotone inclusion problem (Eq. 2.28). The forward-backward algorithm has a very simple iterative update:

$$x^+ = (I + tB)^{-1}(x - tA(x)),$$

with a $t > 0$ that satisfies some conditions of convergence. It requires only the resolvent of B, not the resolvent of $A+B$ or A. As a trade-off, there is a limitation on the step size t. The efficiency of the forward-backward algorithm depends on the complexity to evaluate the resolvent of B and the limitation on the step size t introduced by the forward evaluation $x - tA(x)$. The convergence theory requires co-coercivity of A [Van12]. A is co-coercive if

$$(A(u) - A(v))^T(u - v) \geq \frac{1}{L}\|A(u) - A(v)\|_2^2, \quad \forall u, v \in \mathbf{dom} A$$

for some $L > 0$. The step size is limited by $t \in (0, 2/L)$. This is a rather strong condition. For example, a skew-symmetric linear operator is not co-coercive because the left hand side of this inequality is always zero. Instead, Tseng [Tse00] proposed another modified forward-backward method,

$$\begin{aligned} y &= (I + tB)^{-1}(x - tA(x)) \\ x^+ &= y - t(A(y) - A(x)), \end{aligned}$$

which requires only Lipschitz continuity of A. A is Lipschitz continuous if

$$\|A(u) - A(v)\|_2 \leq L\|u - v\|_2, \quad \forall u, v \in \mathbf{dom} A.$$

In fact, co-coercivity implies Lipschitz continuity, which follows directly from Cauchy-Schwarz inequality.

There are several typical examples for the forward-backward algorithm. Consider the primal optimality condition in (Eq. 2.4) for differentiable g as the monotone inclusion problem in the format of (Eq. 2.28)

$$0 \in \partial f(x) + C^T \nabla g(Cx),$$

with,

$$A(x) = C^T \nabla g(Cx), \qquad B(x) = \partial f(x).$$

The forward-backward splitting algorithm is just proximal gradient method

$$x^+ = \mathrm{prox}_{tf}(x - tC^T \nabla g(Cx)).$$

The convergence rate for the proximal gradient method is generally considered slower than normal gradient method. However, there are many different accelerated proximal gradient methods like FISTA [BT09] and Nesterov methods [Nes07] that could improve convergence rate for proximal gradient methods.

As discussed above, the forward-backward algorithm requires co-coercivity for the forward operator, a stronger condition than Lipschitz continuity. If the forward operator is only Lipschitz continuous, the formulation proposed by Tseng [Tse00] can be used. Consider the primal-dual optimality condition (Eq. 2.17)

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix},$$

with

$$A(x,z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}, \qquad B(x,z) = \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}.$$

The first operator is a skew-symmetric linear operator, which does not satisfy the co-coercivity condition. However, it is Lipschitz continuous with $L = \|C\|_2$ which enables the modified forward-backward splitting method

$$\begin{aligned} u &= \operatorname{prox}_{tf}(x - tC^T z) \\ v &= \operatorname{prox}_{tg^*}(z + tCx) \\ x^+ &= u - tC^T(v - z) \\ z^+ &= v + tC(u - x). \end{aligned}$$

These forward-backward methods require only one of the resolvents if the other operator is co-coercive or Lipschitz continuous. In terms of complexity, the forward-backward methods are more efficient in each iteration since only one resolvent evaluation is necessary. The trade-off is the limitation on the step size.

2.5.3 Semi-Implicit Splitting Algorithm

Semi-implicit splitting methods generally apply to the primal-dual optimality condition. For example, consider the monotone inclusion problem (Eq. 2.28) for optimality condition (Eq. 2.17),

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix},$$

with

$$A(x,z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z \end{bmatrix}, \qquad B(x,z) = \begin{bmatrix} \partial f(x) \\ \partial g^*(z) \end{bmatrix}.$$

In the proximal point algorithm, (x^+, z^+) is the solution of the monotone inclusion problem

$$\frac{1}{t}(x - x^+, z - z^+) \in A(x^+, z^+) + B(x^+, z^+).$$

For both of the operator A and B, evaluations are based on the resulting point (x^+, z^+), which in this case are called backward evaluations. Typically backward evaluation requires the solution of a minimization problem. In the forward-backward algorithm, (x^+, z^+) is the solution of the monotone inclusion problem

$$\frac{1}{t}(x - x^+, z - z^+) \in A(x, z) + B(x^+, z^+).$$

Here operator A is evaluated at current point (x, z), which is called a forward evaluation while B is evaluated at the resulting point (x^+, z^+), which is called a backward evaluation.

In a so-called semi-implicit algorithm, the primal and dual variable are treated separately: For one operator, it evaluates the primal variable as a forward step and the dual variable as a backward step or vice versa. For example, the monotone inclusion problem

$$\frac{1}{t}(x - x^+, z - z^+) \in A(x, z^+) + B(x^+, z^+)$$

corresponds to a semi-implicit algorithm. If we simplify this monotone inclusion problem:

$$\frac{1}{t}\begin{bmatrix} x - x^+ \\ z - z^+ \end{bmatrix} \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} x \\ z^+ \end{bmatrix} + \begin{bmatrix} \partial f(x^+) \\ \partial g^*(z^+) \end{bmatrix},$$

then this semi-implicit algorithm involves two resolvent evaluations:

$$\begin{aligned} z^+ &= \operatorname{prox}_{tg^*}(z + tCx) \\ x^+ &= \operatorname{prox}_{tf}(x - tC^T z^+). \end{aligned}$$

It consists of two very simple update steps that only require the proximal operators of f and g. This is known as Arrow-Hurwicz algorithm, which dates back to the 1950s. There are many variations of Arrow-Hurwicz algorithm. Chambolle and Pock [CP11] proposed a modified algorithm for the primal-dual optimality condition (Eq. 2.17),

$$\begin{aligned} z^+ &= \operatorname{prox}_{tg^*}(z + tC\bar{x}) \\ x^+ &= \operatorname{prox}_{tf}(x - tC^T z^+) \\ \bar{x} &= x^+ + \theta(x^+ - x), \quad \theta \in [0,1]. \end{aligned}$$

It is a modified Arrow-Hurwicz method by adding a predictor $\bar{x}$. As a special case, when $\theta = 0$ it is exactly the same as Arrow-Hurwicz algorithm. The parameter θ controls the level of inertia corresponding to the current update direction $x^+ - x$. The step size is limited by the first operator.

Chen and Teboulle [CT94] proposed another algorithm based on the primal-dual optimality condition in (Eq. 2.15),

$$0 \in \begin{bmatrix} 0 & 0 & C^T \\ 0 & 0 & -I \\ -C & I & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} \partial f(x) \\ \partial g(y) \\ 0 \end{bmatrix}.$$

with the trivial splitting. The update steps for this semi-implicit algorithm are

$$\begin{aligned}
\bar{z} &= z + t(Cx - y) \\
x^+ &= \mathrm{prox}_{tf}(x - tC^T\bar{z}) \\
y^+ &= \mathrm{prox}_{tg}(y + t\bar{z}) \\
z^+ &= z + t(Cx^+ - y^+).
\end{aligned}$$

It is a modified Arrow-Hurwicz algorithm with z^+ substituted by $\bar{z}$ as a predictor. Similar to the algorithm Chambolle and Pock proposed, this heuristic is used for better performance in convergence and stability. The step size is limited by the first operator as well.

These semi-implicit methods are adaptable for many different splittings of the operator for the primal-dual optimality condition. They have limitations on the step size similar to the forward-backward methods in general. But they are simpler to calculate compared to the Douglas-Rachford splitting methods.

We have proposed three general forms of splitting algorithms for a general convex optimization problem (Eq. 2.3). In Chapter 3 and Chapter 4, we further discuss the formulations, performance and other properties of the specific splitting methods in practical applications.

CHAPTER 3

Structured Sparse Principal Component Analysis

3.1 Problem Description

Principal Component Analysis (PCA) is a very useful tool in unsupervised dimension reduction, lossy data compression, feature extraction, and data visualization [Jol02]. There are two commonly used definitions of PCA [Bis06].

- The orthogonal projection of the data onto a lower dimensional linear subspace so that the variance of the projected data is maximized.

  Consider a data set $\{x_n\}$, where $n = 1, 2, \ldots, N$. The variance of the data set projected on v is

  $$\frac{1}{N}\sum_{n=1}^{N}\left(v^T x_n - v^T \bar{x}\right)^2 = v^T S v,$$

  where S is the data covariance matrix:

  $$S = \frac{1}{N}\sum_{n=1}^{N}(x_n - \bar{x})(x_n - \bar{x})^T.$$

  The goal is to find the unit vector v, $\|v\|_2 = 1$, such that the projection variance $v^T S v$ is maximized. This vector v is call the *principal component*. One can prove that v is the eigenvector of S corresponding to the largest eigenvalue. PCA tries to find a set of these orthogonal projection vectors v_i such that the overall variance of projection is maximized [Bis06, Eld07].

- The orthogonal projection of the data onto a lower dimensional linear subspace so that the average projection lost, or the total distance between the data points and their projections is minimized.

  Consider a set of complete D-dimensional basis vectors $v_i \in \mathbb{R}^D$, where $i = 1, 2, \ldots, D$. The goal is to approximate the data set $\{x_n\}$ using $M < D$ number of basis vectors. This can be considered as a projection to a lower-dimension subspace:

  $$\tilde{x}_n = \sum_{i=1}^{M} u_{ni} v_i + \sum_{i=M+1}^{D} b_i v_i,$$

  such that the total projection error between x_n and $\tilde{x}_n$ is minimized. The coefficient u_{ni} is called the *loading vector* of the principal component v_i for $\tilde{x}_n$. The last term $\sum_{i=M+1}^{D} b_i v_i$ is the center of the projection. It is defined such that $b_i = \bar{x}^T v_i$. The precise definition of total projection error is

  $$\begin{aligned} J &= \frac{1}{N} \sum_{n=1}^{N} \|x_n - \tilde{x}_n\|^2 \\ &= \frac{1}{N} \sum_{n=1}^{N} \| \sum_{i=M+1}^{D} \left((x_n - \bar{x})^T v_i \right) v_i \|^2 \\ &= \frac{1}{N} \sum_{n=1}^{N} \sum_{i=M+1}^{D} \left(x_n^T v_i - \bar{x}^T v_i \right)^2 \\ &= \sum_{i=M+1}^{D} v_i^T S v_i. \end{aligned}$$

  One can prove from the last equation that the M optimal projection vectors v_i are the M eigenvectors of S corresponding to the M largest eigenvalues because the projection error is minimized when choosing the $D - M$ eigenvectors of S corresponding to the $D - M$ smallest eigenvalues.

PCA can be computed via a singular value decomposition (SVD) of the data matrix. It is also related to matrix factorization problem if the principal components and corresponding loading vectors are formed in two matrices. Namely, if we assume the center of the projection is at the origin, then the projection error function is

$$\begin{aligned} J &= \frac{1}{N}\sum_{n=1}^{N}\|x_n - \tilde{x}_n\|^2 \\ &= \frac{1}{N}\|X - \tilde{X}\|_F^2 \\ &= \frac{1}{N}\|X - VU^T\|_F^2, \end{aligned} \tag{3.1}$$

where $V = (v_1, v_2, \ldots, v_M)$ and $U = (u_1^T, u_2^T, \ldots, u_n^T)^T$ and $X = (x_1, x_2, \ldots, x_n)$. Therefore, PCA can be formulated as a matrix factorization problem where the solution is related to SVD of the data matrix. However, the regular PCA does not consider any sparsity pattern in the principal components or the loading vectors even if prior knowledge of sparsity is known in the applications.

In many applications, the principal components obtained from PCA are not sparse especially when the original data set is corrupted by heavy noise. There are many alternatives that aim at promoting sparsity in principal components while achieving a dimension reduction similar to PCA. These alternative methods are known as Sparse Principal Component Analysis (SPCA). Zou et al. [ZHT06] proposed a regression-type approach to SPCA. They established a connection between PCA and l_2 regression, and extended this connection to SPCA using lasso and elastic net. d'Aspremont et al. [dBG08] proposed a direct formulation of SPCA by minimizing the approximation error of the data covariance matrix with additional cardinality penalty. It can be formulated as a semi-definite programming (SDP) problem using convex relaxation. Lee et al. [LBR07] proposed a sparse coding method based on non-negative matrix factorization (NMF) technique. The sparse coding technique promotes sparsity in the principal components as well as minimizes the projection cost similar to other SPCA formulations. However, the sparse coding problem does not emphasize the orthogonality between principal components.

Although SPCA successfully promotes sparsity in principal components while still managing to achieve dimension reduction, many applications require certain structures or patterns of sparsity to be imposed based on some prior knowledge of the application. Most of these applications emphasize sparsity structures more than orthogonality between principal components. Jenatton et al. [JOB10] proposed a structured sparse principal component analysis (SSPCA) application based on the sparse coding formulation. This is one of the applications that can be handled efficiently by formulating it as a sparse matrix factorization problem. Namely, given a data matrix $X \in \mathbb{R}^{n \times p}$. The goal is to find a structured sparse principal component matrix V and a corresponding loading matrix U, such that the reconstruction error of X by the representation $\hat{X} = UV^T$ is minimized.

3.2 Matrix Factorization Formulation

In this thesis, we represent the SSPCA problem based on the formulation from Jenatton et al. [JOB10]. It can also be interpreted as a sparse coding formulation from Lee et al. [LBR07]. The problem is

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_{k=1}^{r} g(V^k) \\ &\text{subject to} \quad h(U^k) \le 1, \quad k = 1, 2, \ldots, r. \end{aligned} \tag{3.2}$$

The n rows of the data matrix $X \in \mathbb{R}^{n \times p}$ correspond to n observations in $\mathbb{R}^p$. The n rows of the decomposition matrix $U \in \mathbb{R}^{n \times r}$ correspond to n different loading vectors in $\mathbb{R}^r$. The r columns of the principal component matrix $V \in \mathbb{R}^{p \times r}$ correspond to r principal components in $\mathbb{R}^p$. This is identical to the problem defined in (Eq. 1.2). It also relates to the minimal projection loss definition of PCA discussed in (Eq. 3.1).

The structured sparsity inducing norm $g(V^k)$ is a group sparsity norm defined as (Eq. 2.2). It is defined by the groups of index sets and the group weight matrix. The index set group is defined based on the prior knowledge of the structured sparsity pattern. It is similar to the column-wise sparsity norm in (Eq. 1.3). In SSPCA, this structured sparsity inducing norm proposed by Jenatton et al. [JOB10] is known to exploit structured sparsity. However, determining the best overlapping sparsity group and weight matrix for each group still remains an open problem. We will not discuss this topic in this thesis.

Additionally, since the penalty function can be close to zero if we do not constrain the magnitude of U, a regularization constraint on the decomposition matrix U is introduced. Typically, the l_2 norm constraints $h(U^k) = \|U^k\|_2 \leq 1$ for the loading vectors are considered suitable for SSPCA application.

Consider the typical SSPCA formulation,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_{k=1}^{r} g(V^k) \\ \text{subject to} \quad & \|U^k\|_2^2 \leq 1, \quad k = 1, 2, \ldots, r. \end{aligned} \tag{3.3}$$

It can be formulated as two convex sub-problems through the alternating descent algorithm in (Eq. 1.6): The convex minimization problem over the weight decomposition matrix U with a fixed principal component matrix V,

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X - UV^T\|_F^2 \\ \text{subject to} \quad & \|U^k\|_2^2 \leq 1, \quad k = 1, 2, \ldots, r, \end{aligned} \tag{3.4}$$

and the convex minimization problem over the principal component matrix V with a fixed weight decomposition matrix U,

$$\text{minimize} \quad \frac{1}{2}\|X - UV^T\|_F^2 + \lambda \sum_{k=1}^{r} g(V^k). \tag{3.5}$$

These two problems both fit in the same format of the generalized problem (Eq. 2.3) discussed in Chapter 2. We will discuss these two problems separately due to the difference in problem characteristics and properties.

3.3 Minimization over Weight Decomposition Matrix

Consider the convex minimization problem over weight decomposition matrix U with a fixed principal component matrix V in (Eq. 3.4). The l_2 norm constraint can be represented as an indicator function of the unit l_2 norm ball. The problem (Eq. 3.4) can be written as

$$\text{minimize} \quad \tilde{f}(U) + \tilde{h}(U),$$

with the two parts of the objective function as:

$$\begin{aligned} \tilde{f}(U) &= \frac{1}{2}\|X - UV^T\|_F^2 && (3.6) \\ \tilde{h}(U) &= \sum_{k=1}^{r} I_{\|.\|_2 \leq 1}(U^k). && (3.7) \end{aligned}$$

The function $\tilde{h}$ is the sum of the l_2 norm ball indicator functions applied to all columns of matrix U. It is in the same format of problem (Eq. 2.3) with the linear mapping as identity matrix: $C = I$.

3.3.1 Proximal Operators

In order to use splitting algorithms discussed in Chapter 2 to solve this problem, the following proximal evaluations are required in general:

The first resolvent is for the subdifferential $\partial \tilde{f}(U)$. It is the proximal operator for the quadratic loss function in (Eq. 3.6):

$$\begin{aligned} & U = \arg\min_{U} \left\{ \frac{1}{2}\|X - UV^T\|_F^2 + \frac{1}{2t}\|U - \hat{U}\|_2^2 \right\} \\ \iff & 0 = (UV^T - X)V + \frac{1}{t}(U - \hat{U}) \\ \iff & U = (XV + \frac{1}{t}\hat{U})(V^TV + \frac{1}{t}I)^{-1}. \end{aligned}$$

The second resolvent is the proximal operator for the indicator function of a l_2 norm ball (Eq. 3.7). It is a projection onto the l_2 norm ball:

$$\begin{aligned} U^k &= \arg\min_{U:\|U^k\|_2 \le 1} \left\{ \frac{1}{2t}\|U^k - \hat{U}^k\|_2^2 \right\} \\ &= P_{\|.\|_2 \le 1}(\hat{U}^k) \\ &= \begin{cases} \dfrac{\hat{U}^k}{\|\hat{U}^k\|_2}, & \|\hat{U}^k\|_2 > 1 \\ \hat{U}^k, & \text{otherwise.} \end{cases} \end{aligned}$$

These two resolvents are easy to compute with explicit solutions. In the splitting methods, at least one of the two proximal operator is used each iteration.

3.3.2 Methods

The problem formulation does not involve any linear mapping of U. Additionally, the subdifferential for the quadratic loss function is a singleton. Therefore, it is generally considered appropriate to apply primal (Eq. 2.4) or dual (Eq. 2.11) optimality condition to the problem.

Consider the primal optimality condition as an example. This problem can be solved efficiently using the projected gradient method, a special case of proximal gradient method. The projected gradient method is a forward-backward method with the forward operator as a gradient update and the backward operator as a projection on a convex set. The projected gradient method for problem (Eq. 3.4) has a simple update:

$$\begin{aligned} U^+ &= \operatorname{prox}_{t\tilde{h}}(U - t\nabla\tilde{f}(U)) \\ &= P_{\|.\|_2\leq 1}(U - t(UV^T - X)V). \end{aligned}$$

Since the forward operator is just gradient mapping, there are many accelerated methods suitable in this problem with better convergence rate. Consider using FISTA [BT09] as an example. The accelerated proximal gradient method is

$$\begin{aligned} \tilde{U} &= U_{(k-1)} + \frac{k-2}{k+1}(U_{(k-1)} - U_{(k-2)}) \\ U_{(k)} &= P_{\|.\|_2\leq 1}(\tilde{U} - t(\tilde{U}V^T - X)V). \end{aligned}$$

The gradient or forward step is evaluated at a point along the previous update direction. Intuitively, it uses the previous update information as a corrector for the current update. Generally the accelerated gradient methods are considered much faster than simple projected gradient method.

3.4 Minimization Over Principal Component Matrix

Consider the convex minimization problem over the principal component matrix V with a fixed weight decomposition matrix U in (Eq. 3.5). The typical overlapping group sparsity penalty function is in the format of (Eq. 1.3),

$$g(V^k) = \sum_{\alpha\in A} \|V^k_\alpha\|_2.$$

Since the group sparsity penalty has coupled variables within each column of V, it is hard to compute the proximal operator for it. Instead, the overlapping group sparsity constraint over g can be formulated into a non-overlapping group sparsity constraint applied to some linear transformation of V. Namely, consider writing each index set explicitly:

$$\begin{aligned} g(V^k) &= \sum_{\alpha \in A} \|V^k_\alpha\|_2 \\ &= \sum_{i=1}^{|A|} \|V^k_{\alpha_i}\|_2. \end{aligned}$$

The variable $V^k_{\alpha_i}$ is the vector formulated by elements in the i-th index set. Denote vector $x_{(i)} = V^k_{\alpha_i}$ and $x = (x^T_{(1)}, x^T_{(2)}, \ldots, x^T_{(|A|)})^T$. Then the overlapping group l_1-l_2 norm (Eq. 1.3) is equivalent to

$$\tilde{g}(x) = \sum_{i=1}^{|A|} \|x_{(i)}\|_2. \tag{3.8}$$

This is a non-overlapping group norm with respect to its own variable x. Interestingly, notice that vector x is a linear transform of V^k:

$$x = CV^k = \begin{bmatrix} V^k_{\alpha_1} \\ V^k_{\alpha_2} \\ \vdots \\ V^k_{\alpha_{|A|}} \end{bmatrix}, \quad \forall \alpha_i \in A. \tag{3.9}$$

This linear mapping is extremely simple and therefore, the matrix C is expected to be extremely sparse.

Let's use an example to explain the linear transformation. Consider a vector $v = (v_1, v_2, v_3, v_4) \in \mathbb{R}^4$ and the overlapping group

$$A = \{\{1\}, \{1, 2\}, \{1, 2, 3\}, \{1, 2, 3, 4\}\}.$$

We use the linear transformation (Eq. 3.9):

$$x = Cv = \begin{bmatrix} (v_1) \\ (v_1, v_2)^T \\ (v_1, v_2, v_3)^T \\ (v_1, v_2, v_3, v_4)^T \end{bmatrix} \in \mathbb{R}^{10}.$$

Then the overlapping group norm can be represented as:

$$\begin{aligned} g(v) &= \sum_{\alpha \in A} \|V_\alpha\|_2 \\ &= |v_1| + \|(v_1, v_2)\|_2 + \|(v_1, v_2, v_3)\|_2 + \|(v_1, v_2, v_3, v_4)\|_2 \\ &= \|x_{(1)}\|_2 + \|x_{(2)}\|_2 + \|x_{(3)}\|_2 + \|x_{(4)}\|_2 \\ &= \tilde{g}(x) \\ &= \tilde{g}(Cv). \end{aligned}$$

For a more complicated group sparsity norm with a weight matrix such as (Eq. 2.2), we can use the same non-overlapping function (Eq. 3.8) with a variation of the linear mapping (Eq. 3.9):

$$x = CV^k = \begin{bmatrix} D_{\alpha_1} V^k_{\alpha_1} \\ D_{\alpha_2} V^k_{\alpha_2} \\ \vdots \\ D_{\alpha_{|A|}} V^k_{\alpha_{|A|}} \end{bmatrix}, \quad \forall \alpha_i \in A, \tag{3.10}$$

such that the modified mapping matrix C includes the weight matrix for each group. This modified matrix C in (Eq. 3.10) has the same sparsity pattern as the matrix C in (Eq. 3.9).

With the above linear transformation, the problem (Eq. 3.5) can be reformulated as:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X^T - VU^T\|_F^2 + \lambda \sum_{k=1}^{r} \sum_{i=1}^{|A|} \|W^k_{(i)}\|_2 \\ \text{subject to} \quad & W = CV, \end{aligned} \tag{3.11}$$

with the matrix C defined above. Notice that the Frobenius norm square does not change if the matrix is transposed. Problem (Eq. 3.11) has the same format as problem (Eq. 2.3). It can be written as

$$\begin{aligned} \text{minimize} \quad & \tilde{f}(V) + \tilde{g}(W) \\ \text{subject to} \quad & W = CV, \end{aligned}$$

with the two parts of the objective in (Eq. 3.11) as:

$$\tilde{f}(V) = \frac{1}{2}\|X^T - VU^T\|_F^2 \tag{3.12}$$

$$\tilde{g}(W) = \lambda \sum_{k=1}^{r} \sum_{i=1}^{|A|} \|W_{(i)}^k\|_2. \tag{3.13}$$

This is the exact same formulation in (Eq. 2.6) with a linear transformation matrix C defined in (Eq. 3.9) or (Eq. 3.10).

3.4.1 Linear Mapping

The linear mapping in problem (Eq. 3.11) is much more complicated than in problem (Eq. 3.4). There are some interesting properties from the definition in (Eq. 3.9, 3.10).

Consider the linear mapping for a non-weighted matrix C in (Eq. 3.9). The elements in C are either 1 or 0. Each row of C is a unit vector e_i corresponding to the particular element in vector V^k. This comes directly from the property of C, an element-wise mapping from an original vector V^k to the new vector W^k. Additionally, C^TC is a diagonal matrix. All columns of C are orthogonal to each other. The diagonal element $(C^TC)_{ii}$ equals to the total number of appearances of element V_i^k in all index sets.

For a weighted matrix C in (Eq. 3.10), each row of C is the unit vector e_i scaled by the corresponding diagonal element in the weight matrix. The sparsity pattern of C does not change by these weight matrices. C^TC is still a diagonal matrix. The diagonal elements now equal to the l_2 norm square for each column of C. In other words, it is the square sum of all group weights for every element V_i^k.

3.4.2 Proximal Operators

In order to use splitting algorithms to solve this problem, the following proximal evaluations are required:

The first resolvent is for the subdifferential $\partial\tilde{f}$. It is the proximal operator for the quadratic loss function (Eq. 3.12):

$$\begin{aligned}
& V = \operatorname{prox}_{t\tilde{f}}(\hat{V}) \\
\iff\quad & V = \arg\min_{V} \left\{ \frac{1}{2}\|X^T - VU^T\|_F^2 + \frac{1}{2t}\|V - \hat{V}\|_F^2 \right\} \\
\iff\quad & 0 = (VU^T - X^T)U + \frac{1}{t}(V - \hat{V}) \\
\iff\quad & V = (X^T U + \frac{1}{t}\hat{V})(U^T U + \frac{1}{t}I)^{-1}.
\end{aligned}$$

The second resolvent is the proximal operator for the non-overlapping l_1-l_2 penalty function (Eq. 3.13). Since the non-overlapping l_1-l_2 norm is separable, the proximal operator is also separable:

$$\begin{aligned}
& W = \operatorname{prox}_{t\tilde{g}}(\hat{W}) \\
\iff\quad & W = \arg\min_{W} \left\{ \lambda \sum_{k=1}^{r} \sum_{i=1}^{|A|} \|W_{(i)}^k\|_2 + \frac{1}{2t}\|W - \hat{W}\|_F^2 \right\} \\
\iff\quad & W^k = \arg\min_{W^k} \left\{ \lambda \sum_{i=1}^{|A|} \|W_{(i)}^k\|_2 + \frac{1}{2t}\|W^k - \hat{W}^k\|_2^2 \right\} \\
\iff\quad & W_{(i)}^k = \arg\min_{W_{(i)}^k} \left\{ \lambda\|W_{(i)}^k\|_2 + \frac{1}{2t}\|W_{(i)}^k - \hat{W}_{(i)}^k\|_2^2 \right\} \\
\iff\quad & W_{(i)}^k = \arg\min_{W_{(i)}^k} \left\{ \|W_{(i)}^k\|_2 + \frac{1}{2\lambda t}\|W_{(i)}^k - \hat{W}_{\tilde{\alpha}_i}^k\|_2^2 \right\} \\
\iff\quad & W_{(i)}^k = \operatorname{prox}_{\lambda t\|.\|_2}(\hat{W}_{(i)}^k).
\end{aligned}$$

The notation $\operatorname{prox}_{\lambda t\|.\|_2}$ is the proximal operator of the l_2 norm function scaled by

λt. It has a closed form solution:

$$\operatorname{prox}_{\lambda t\|.\|_2}(\hat{x}) = \begin{cases} \left(1 - \dfrac{\lambda t}{\|\hat{x}\|_2}\right)\hat{x} & \|\hat{x}\|_2 \geq \lambda t \\ 0 & \text{otherwise.} \end{cases}$$

This is a soft-thresholding function for a scaled l_2 norm. It is zero inside the norm ball and scale linearly with respect to the distance from the boundary of the norm ball.

The problem (Eq. 3.11) is considered much more complicated than (Eq. 3.4). Generally speaking there is no way to judge which splitting method is the best unless with actual experiment results. For all these splitting methods, we denote V and W as the primal variable, Z as the dual multiplier for the equality constraint $W = CV$. We discuss these methods in three categories based on the optimality conditions.

3.4.3 Primal Methods

Problem (Eq. 3.11) is equivalent to the reformulation:

$$\begin{array}{ll} \text{minimize} & \tilde{f}(V) + \tilde{g}(W) \\ \text{subject to} & \begin{bmatrix} -I & C \end{bmatrix} \begin{bmatrix} W \\ V \end{bmatrix} = 0, \end{array} \qquad (3.14)$$

with the $\tilde{f}$ and $\tilde{g}$ defined in (Eq. 3.12, 3.13). It is the similar definition in (Eq. 2.6) except this formulation explicitly shows that the constraint is a subspace with respect to the joint variable (V, W).

Douglas-Rachford Splitting

Spingarn's method [LS86] is Douglas-Rachford splitting method apply to a specific type of problem like (Eq. 3.14). Consider

$$\begin{aligned} \text{minimize} \quad & f(x) \\ \text{subject to} \quad & x \in S, \end{aligned}$$

where S is a subspace. The Douglas-Rachford splitting algorithm for this type of problem is

$$x^+ = \operatorname{prox}_{tf}(z) \tag{3.15}$$

$$y^+ = P_S(2x^+ - z) \tag{3.16}$$

$$z^+ = z + y^+ - x^+, \tag{3.17}$$

with the first resolvent as the proximal operator over f and the second resolvent as a projection onto the subspace S. Apparently the modified primal problem (Eq. 3.14) falls in this category because the linear constraint is a subspace of the combined variable V and W.

In the Spingarn's method formulation, the first resolvent requires the proximal operators for both $\tilde{f}$ and $\tilde{g}$ because the combined variable V and W are separable without the equality constraint in (Eq. 3.11). The second resolvent requires a projection onto the subspace formulate by the linear constraint $W = CV$. The projection can be represented as the minimum distance point on the subspace from the point of evaluation:

$$P_S(\hat{W}, \hat{V}) = \underset{(W,V)\in S}{\arg\min} \left\{ \|V - \hat{V}\|_F^2 + \|W - \hat{W}\|_F^2 \right\},$$

where the subspace is defined as

$$S = \left\{ \begin{bmatrix} W \\ V \end{bmatrix} \middle| \begin{bmatrix} -I & C \end{bmatrix} \begin{bmatrix} W \\ V \end{bmatrix} = 0 \right\}.$$

This projection has a closed form solution:

$$\begin{aligned}\begin{bmatrix} W^* \\ V^* \end{bmatrix} &= \left\{ I - \begin{bmatrix} I \\ -C^T \end{bmatrix} \left(I + CC^T\right)^{-1} \begin{bmatrix} I & -C \end{bmatrix} \right\} \begin{bmatrix} \hat{W} \\ \hat{V} \end{bmatrix} \\ &= \left\{ I - \begin{bmatrix} I \\ -C^T \end{bmatrix} \left[I - C \left(I + C^T C\right)^{-1} C^T \right] \begin{bmatrix} I & -C \end{bmatrix} \right\} \begin{bmatrix} \hat{W} \\ \hat{V} \end{bmatrix}. \end{aligned}$$

The second formulation is much more efficient because $C^T C$ is diagonal. This is an advantage of using Spingarn's method. However, matrix C is usually extremely overdetermined, especially in SSPCA application. Therefore, W is much larger than V. It is generally considered not efficient to project on the subspace S of the joint variable (W, V).

3.4.4 Primal-Dual Methods

Consider the primal-dual optimality condition in (Eq. 2.17),

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} V \\ Z \end{bmatrix} + \begin{bmatrix} \partial \tilde{f}(V) \\ \partial \tilde{g}^*(Z) \end{bmatrix}.$$

We consider the obvious splitting of the monotone inclusion as $0 \in A(V, Z) + B(V, Z)$ with

$$A(V, Z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} V \\ Z \end{bmatrix}, \qquad B(V, Z) = \begin{bmatrix} \partial \tilde{f}(V) \\ \partial \tilde{g}^*(Z) \end{bmatrix}.$$

Douglas-Rachford Splitting

Douglas-Rachford splitting applied to the primal-dual optimality condition with the obvious splitting above gives a simple method to solve this problem. The first resolvent for operator $A(V, Z)$ is equivalent to the following minimization problem:

$$V = \arg\min_V \left(\frac{t}{2}\|CV - \frac{\hat{Z}}{t}\|_F^2 + \frac{1}{2t}\|V - \hat{V}\|_F^2 \right),$$

which has a closed form solution:

$$\begin{aligned} V^* &= (t^2 C^T C + I)^{-1}(-tC^T\hat{Z} + \hat{V}) \\ Z^* &= \hat{Z} + tC\hat{V}. \end{aligned}$$

The second resolvent for operator $B(V, Z)$ is separable between $\tilde{f}$ and $\tilde{g}^*$. The resolvents are just proximal operators applied to the subdifferential of the two functions. Moreover, the proximal operator for the conjugate $\tilde{g}^*$ is the projection onto the scaled l_2 norm ball because the original function is the scaled l_2 norm. The Douglas-Rachford splitting algorithm:

$$\begin{aligned} \tilde{V} &= (t^2 C^T C + I)^{-1}(-tC^T\hat{Z} + \hat{V}), \\ \tilde{Z} &= \hat{Z} + tC\tilde{V} && (3.18) \\ \bar{V} &= \text{prox}_{t\tilde{f}}(2\tilde{V} - \hat{V}), \\ \bar{Z} &= \text{prox}_{t\tilde{g}^*}(2\tilde{Z} - \hat{Z}) && (3.19) \\ \hat{V}^+ &= \hat{V} + \bar{V} - \tilde{V}, \\ \hat{Z}^+ &= \hat{Z} + \bar{Z} - \tilde{Z}. && (3.20) \end{aligned}$$

Comparing to the Spingarn's method, the extra resolvent only requires to solve a system with the size the same as the matrix $C^T C$, which is relatively small for an under-determined matrix C. Theoretically it should be more efficient than the Spingarn's method.

Forward-Backward Splitting

The the first operator $A(V, Z)$ of the primal-dual optimality condition is an skew-symmetric linear operator which is not co-coercive. Therefore, the modified forward-backward method proposed by Tseng is suitable. It uses the linear operator $A(V,Z)$ as the forward operator and the combined subdifferential $B(V,Z)$ as the backward operator. The forward-backward method is

$$\begin{aligned} \tilde{V} &= \operatorname{prox}_{t\tilde{f}}(V - tC^T Z) && (3.21) \\ \tilde{Z} &= \operatorname{prox}_{t\tilde{g}^*}(Z + tCV) && (3.22) \\ V^+ &= \tilde{Z} - tC^T(\tilde{Z} - Z) && (3.23) \\ Z^+ &= \tilde{V} + tC(\tilde{V} - V). && (3.24) \end{aligned}$$

The skew-symmetric linear operator is Lipschitz continuous with $L = \|C\|_2$. Therefore, Tseng's modified forward-backward method can be applied with a limited step size $t \leq 1/\|C\|_2$.

Semi-Implicit Splitting Methods

Consider the primal-dual optimality condition in (Eq. 2.17). It is suitable for semi-implicit algorithms. The modified Arrow-Hurwicz method proposed by Chambolle and Pock [CP11] is

$$\begin{aligned} Z^+ &= \operatorname{prox}_{t\tilde{g}^*}(Z + tC\bar{V}) && (3.25) \\ V^+ &= \operatorname{prox}_{t\tilde{f}}(V - tC^T Z^+) && (3.26) \\ \bar{V} &= V^+ + \theta(V^+ - V). && (3.27) \end{aligned}$$

Its step size is limited by $t < 1/\|C\|_2$ when using $\theta = 1$. Another semi-implicit splitting method is based on the primal-dual optimality condition in (Eq. 2.15) with the obvious splitting. It corresponds to the semi-implicit method proposed by Chen and Teboulle [CT94]:

$$\begin{aligned}
\bar{Z} &= Z + t(CV - W) && (3.28)\\
V^+ &= \text{prox}_{t\tilde{f}}(V - tC^T\bar{Z}) && (3.29)\\
W^+ &= \text{prox}_{t\tilde{g}}(W + t\bar{Z}) && (3.30)\\
Z^+ &= Z + t(CV^+ - W^+). && (3.31)
\end{aligned}$$

The step size limitation from Chen and Teboulle is $t < \frac{1}{2}\min\{1, 1/\|C\|_2\}$.

Similar to Tseng's forward-backward method, these semi-implicit splitting methods only require the evaluations of the two proximal operators $\tilde{f}$ and $\tilde{g}$. They are more efficient than Douglas-Rachford splitting methods per iteration. However, its step size t is limited by the Lipschitz continuity parameter. Usually this Lipschitz constant for the linear operator $L = \|C\|_2$ depends highly on the type of sparsity structure required by the application. Fortunately, it is sufficiently small in SSPCA application.

3.4.5 Dual Methods

Consider the dual optimality condition in (Eq. 2.11),

$$0 \in -C\partial\tilde{f}^*(-C^TZ) + \partial\tilde{g}^*(Z).$$

We consider the trivial splitting of the monotone inclusion as $0 \in A(Z) + B(Z)$ with

$$A(Z) = -C\partial\tilde{f}^*(-C^TZ), \qquad B(Z) = \partial\tilde{g}^*(Z).$$

Douglas-Rachford Splitting

The Douglas-Rachford Splitting applying to the dual optimality condition is the ADMM method:

$$\begin{aligned}
P^+ &= \arg\min_P \left\{ \tilde{f}^*(-C^TP) + \frac{1}{2t}\|P-(Q-Z)\|_F^2 \right\} \\
Q^+ &= \arg\min_Q \left\{ \tilde{g}^*(Q) + \frac{1}{2t}\|Q-(P^++Z)\|_F^2 \right\} \\
&= \operatorname{prox}_{t\tilde{g}^*}(P^++Z) \\
Z^+ &= Z+Q^+-P^+.
\end{aligned}$$

P and Q are dual variables. Alternatively, it can also be written in primal formulation by manipulating the relationship between primal an dual variables using the property of Fenchel's conjugate (Eq. 2.16). Its primal formulation is

$$\begin{aligned}
V^+ &= \arg\min_V \left\{ \tilde{f}(V) + \frac{t}{2}\|CV-W+Z/t\|_F^2 \right\} && (3.32) \\
W^+ &= \arg\min_W \left\{ \tilde{g}(W) + \frac{t}{2}\|CV^+-W+Z/t\|_F^2 \right\} \\
&= \operatorname{prox}_{\tilde{g}/t}(CV^++Z/t) && (3.33) \\
Z^+ &= Z+t(CV^+-W^+). && (3.34)
\end{aligned}$$

The first resolvent is an augmented Lagrangian. It requires the solution for

$$V(U^TU)+(tC^TC)V = X^TU+tC^T(W-Z/t), \quad (3.35)$$

which is a rectangular Lyapunov equation. It is also a special case of the general Sylvester matrix equation [GLA92]. Fortunately, matrix C^TC is diagonal and (Eq. 3.35) can be solved using singular value decomposition.

Denote $M = C^TC$, $N = [X^TU + tC^T(W-Z/t)]$. Since M is diagonal by construction, the equation (Eq. 3.35) is equivalent to

$$V^{(k)}(U^TU+tM_{k,k}I) = N^{(k)}, \quad \forall k,$$

where $V^{(k)}$ and $N^{(k)}$ are the k-th rows of matrix V and N, respectively. Let $U = P\Sigma Q^T$ be the singular value decomposition (SVD) of U. Because matrix Q is an orthogonal matrix, the equation (Eq. 3.35) can be further simplified as

$$\begin{aligned} N^{(k)} &= V^{(k)}(U^TU + tM_{k,k}I) \\ &= V^{(k)}(Q\Sigma^2Q^T + tM_{k,k}I) \\ &= V^{(k)}Q(\Sigma^2 + tM_{k,k}I)Q^T. \end{aligned}$$

It is a simple linear equation with respect to the rows of V. Notice that Q is orthogonal and $\Sigma^2 + tM_{k,k}I$ is diagonal. The solution is trivial:

$$\begin{aligned} V^{(k)} &= N^{(k)}Q(\Sigma^2 + tM_{k,k})^{-1}Q^T \\ &= [X^TU + tC^T(W - Z/t)]^kQ(\Sigma^2 + tM_{k,k})^{-1}Q^T. \end{aligned}$$

Therefore, ADMM requires SVD for the decomposition matrix U. It is used to update all rows of V. The computation complexity for this SVD solution is comparable with proximal operator since it only requires one SVD operation if the matrix C^TC is diagonal.

Forward-Backward Splitting

If $\tilde{f}^*$ is differentiable, its gradient evaluated at point $-C^TZ$ is

$$\begin{aligned} \nabla\tilde{f}^*(-C^TZ) &= \arg\min_V \left\{\tilde{f}(V) - \langle V, -C^TZ\rangle\right\} \\ &= \arg\min_V \left\{\tilde{f}(V) - \mathbf{tr}(V, -C^TZ)\right\} \\ &= (-C^TZ + X^TU)(U^TU)^{-1}, \end{aligned}$$

for a non-singular matrix U^TU [Van12]. Using this gradient evaluation as the forward step for the operator $A(Z)$, the proximal gradient method on the dual with a backward step over $B(Z)$ is

$$\begin{aligned} Z^+ &= \operatorname{prox}_{t\tilde{g}^*}(Z + tC\nabla\tilde{f}^*(-C^T Z)) \\ &= \operatorname{prox}_{t\tilde{g}^*}(Z + tC(-C^T Z + X^T U)(U^T U)^{-1}). \end{aligned} \tag{3.36}$$

In this algorithm, the forward operator $C\nabla\tilde{f}^*(-C^T Z)$ is Lipschitz continuous with $L = \|C\|_2^2/\mu$, where μ is the strong convexity modulus parameter of function $\tilde{f}$. This modulus parameter of $\tilde{f}$ is related to the smallest eigenvalue of matrix U: $\mu = \min\lambda(U^T U)$. The step size t is limited by $t < 1/L$. This is a fairly easy proximal gradient method applied to the dual problem.

The goal of SSPCA is to find the structured sparse principal components. Generally the matrix $U^T U$ at the local optimum is non-singular. However, during the alternating descent algorithm, the matrix $U^T U$ might become singular or close to singular. An ill-conditioned matrix $U^T U$ will still affect the speed of convergence by limiting the step size t with a large Lipschitz constant. On the other hand, the benefit of using proximal gradient method is the use of accelerated methods for better convergence rate.

3.5 Experiments

We first consider the 1-dimensional and 2-dimensional randomly generated structured sparsity patterns discussed by Jenatton et al. [JOB10]. Then we consider the SSPCA analysis on the AR Face Database [MB98] as unsupervised learning problem for face recognition application. In these experiments, first we compare the efficiency and convergence rate between block coordinate descent and splitting methods on one of the alternating descent convex sub-problem discussed in Section 3.4. The other problem discussed in Section 3.3 is much easier in terms of formulation, which is not discussed in these experiments. Then we focus on applying these methods in alternating descent algorithm (Eq. 3.3) and discuss the overall results of the sparse matrix factorization problems.

Our experiments are carried out in MATLAB on a system with an Intel Core i7 Qual-core processor running at 2.3 GHz with CPU performance throttle off and 8Gb of memory.

3.5.1 1-D Group Sparsity Recovery

In the randomly generated structured sparsity example, we use an element-wise uniform distribution to generate r basis vectors V^i with 1-D continuous pattern, where all the non-zero elements in vectors V^i are in a random consecutive region. Then we use these basis vectors to generate n data points s,

$$s = \sum_{i=1}^{r} u_i V^i + \epsilon \in \mathbb{R}^p. \tag{3.37}$$

The parameters u_i are the decomposition weights generated randomly through a uniform distribution as well. The parameter ϵ is an independent identical distributed (IID) Gaussian noise such that the generated signal satisfies a predetermined signal-to-noise ratio (SNR).

Convex Sub-problem Minimization

Consider using splitting methods to solve (Eq. 3.5), one of the two sub-problems in (Eq. 3.3). We formulate our experiment with $r = 10$ continuous basis vectors of dimension $p = 50$ and $n = 100$ data points with an SNR of 0.5 dB. Since the objective function for (Eq. 3.5) changes every iteration due to the alternating descent algorithm, we compare the splitting methods with BCD in two scenarios: Initial Phase, where the parameters are randomly generated starting points for

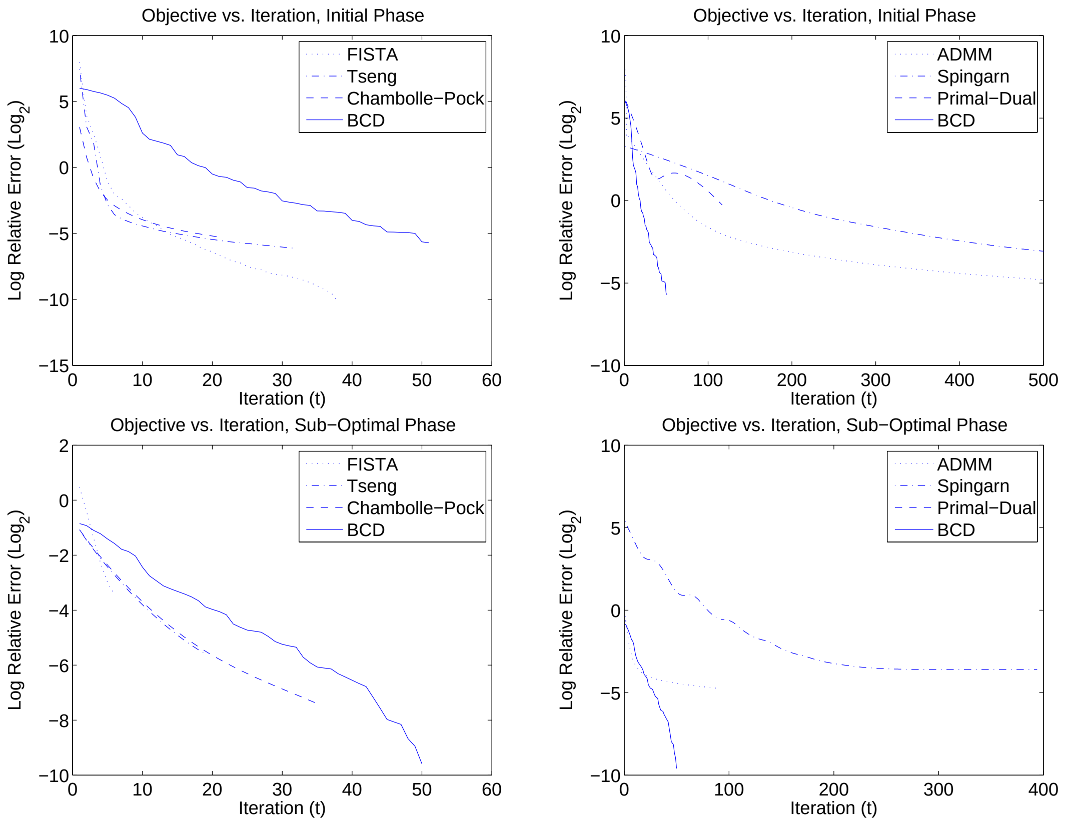


Figure 3.1: *Iterations for the relative objective of the convex sub-problem in 1-D randomly generated sparsity pattern experiment. Splitting methods are separated into two categories by there speeds of convergence for a better comparison with BCD.*

the alternating descent algorithm, and Sub-optimal Phase, where the parameters are determined after approximately 10 alternating descent iterations.

Figure 3.1 shows the relative objective versus iteration plot for these methods. Most splitting methods perform similarly, or better than BCD. Namely, FISTA on the dual, Chambolle-Pock semi-implicit, and Tseng's forward-backward methods perform much better than BCD in both initial and sub-optimal phase. For Douglas-Rachford type methods such as ADMM or primal-dual Douglas-Rachford splitting, the step size can only be roughly estimated due to numerical instability.

Methods	ADMM	Tseng	FISTA	Chambolle-Pock	BCD
Runtime (ms)	11.7	9.95	9.40	9.42	9.07

Table 3.1: *Average runtime per iteration in 1-D randomly generated sparsity pattern experiment, averaging over the entire alternating descent algorithm. We only compare four splitting methods faster than or equivalent to BCD during the alternating descent.*

Table 3.1 shows the average runtime per iteration for these methods. The closed form solution of BCD is cheaper than the evaluation of proximal operator used by the splitting methods. However, a preprocessing step is necessary for each complete BCD cyclic iteration. This preprocessing step is highly complex. Overall performance for BCD is at the same scale as the splitting methods in smaller problems.

Alternating Descent

Although the experiments for the convex sub-problem (Eq. 3.11) show that the performance of splitting methods is better than BCD, it is still important to look at the result in the alternating descent algorithm (Eq. 3.3). Since BCD method does not have a good stopping criterion, we implemented two BCD instants: one with the relative improvement of objective per iteration as a stopping criterion (Soft), and the other with a fixed number of iteration (Hard). We only included splitting methods that perform better or equivalent to BCD in the following experiment.

Figure 3.2 shows the results of the alternating descent for different methods. Each column of an image indicates a 1-D structured sparse principal component, or basis vector. The color gray $(128, 128, 128)$ stands for zero, or sparse elements in

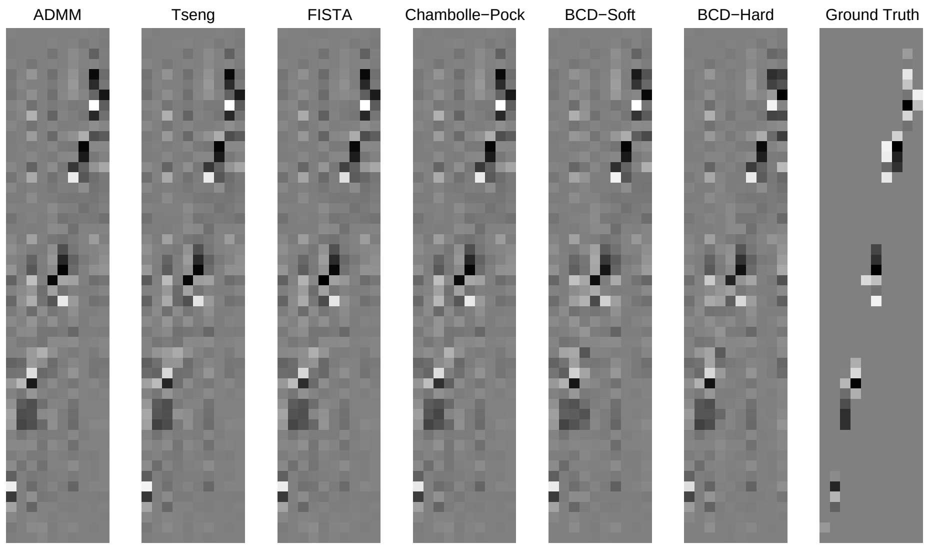


Figure 3.2: *SSPCA results of four splitting methods and two BCD instants comparing with the ground truth in 1-D randomly generated sparsity pattern experiment. Each column of an image corresponds to one principal component.*

the variables. Color black $(0, 0, 0)$ and white $(255, 255, 255)$ stand for the negative and positive maximum of the principal components. Comparing to the ground truth, these methods all give a very close estimation of structured sparse basis vectors and most importantly, preserve the structured sparsity patterns. Notice that some of the structured sparse basis vectors are the negative of the ground truth because it is allowed in our experiment setup.

Figure 3.3 shows the relative objective versus iteration plot for the alternating descent algorithm. Together with the results shown in Figure 3.2, we can conclude that the splitting methods and BCD all converge to the same local minimum. Moreover, these methods follow almost the exact same path converging to the local minimum.

Table 3.2 shows the average iteration per alternating descent step for these

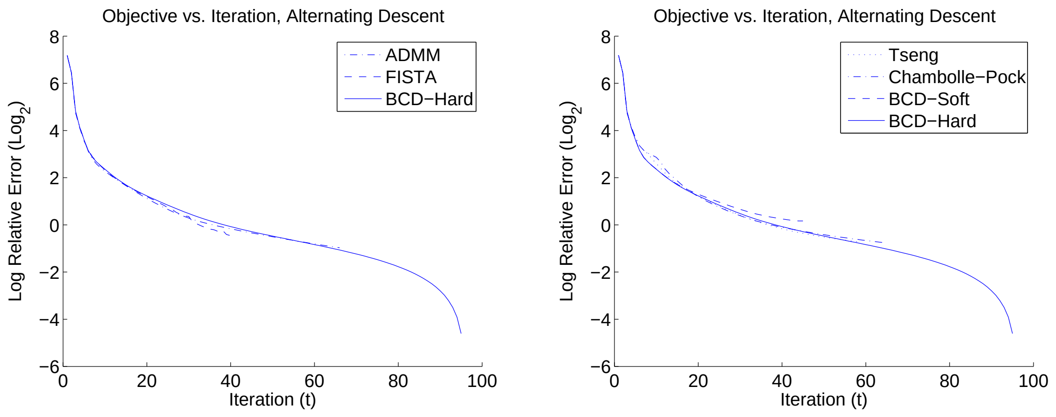


Figure 3.3: *Iterations for the relative objective of the alternating descent in 1-D randomly generated sparsity pattern experiment. Splitting methods are separated into two categories: dual optimality methods (left) and primal-dual optimality methods (right).*

Methods	ADMM	Tseng	FISTA	Chambolle-Pock	BCD-Soft	BCD-Hard
Iteration	72.5	22.8	17.1	47.3	33.0	70

Table 3.2: *Average iteration count per alternating descent step in 1-D randomly generated sparsity pattern experiment. The numbers of inner iterations necessary for these methods to solve a convex sub-problem.*

methods. FISTA and Tseng's forward-backward method use less than half of the number of iteration per alternating descent than BCD does, with almost the same performance each iteration.

To summarize our observations from the 1-D group sparsity pattern experiments, the splitting methods give the same results through alternating descent as BCD does. Most splitting methods perform equivalently or better than BCD in this small-scaled problem.

3.5.2 2-D Group Sparsity Recovery

In the 2-D continuous structured sparsity experiment, we randomly generate r basis vectors with 2-D rectangular/diamond-shape pattern and use these basis vectors to generate n data points s using the same formula in (Eq. 3.37). More precisely, we randomly choose the top-left and bottom-right coordinates drawn from a uniform distribution and form a rectangular region within an image. This region is assigned a non-zero value also following a uniform distribution and the rest of the image is zero, or sparse.

Convex Sub-problem Minimization

Consider using the splitting methods to solve the same convex sub-problem (Eq. 3.11). We formulate our experiment with $r = 10$ basis vectors of dimension $p = 18 \times 18$ with 2-D continuous rectangular pattern. We generate $n = 1000$ data points with an SNR of 0.5 dB. This is a relatively large-scale problem. We still compare all methods in two scenarios, Initial Phase and Sub-optimal Phase.

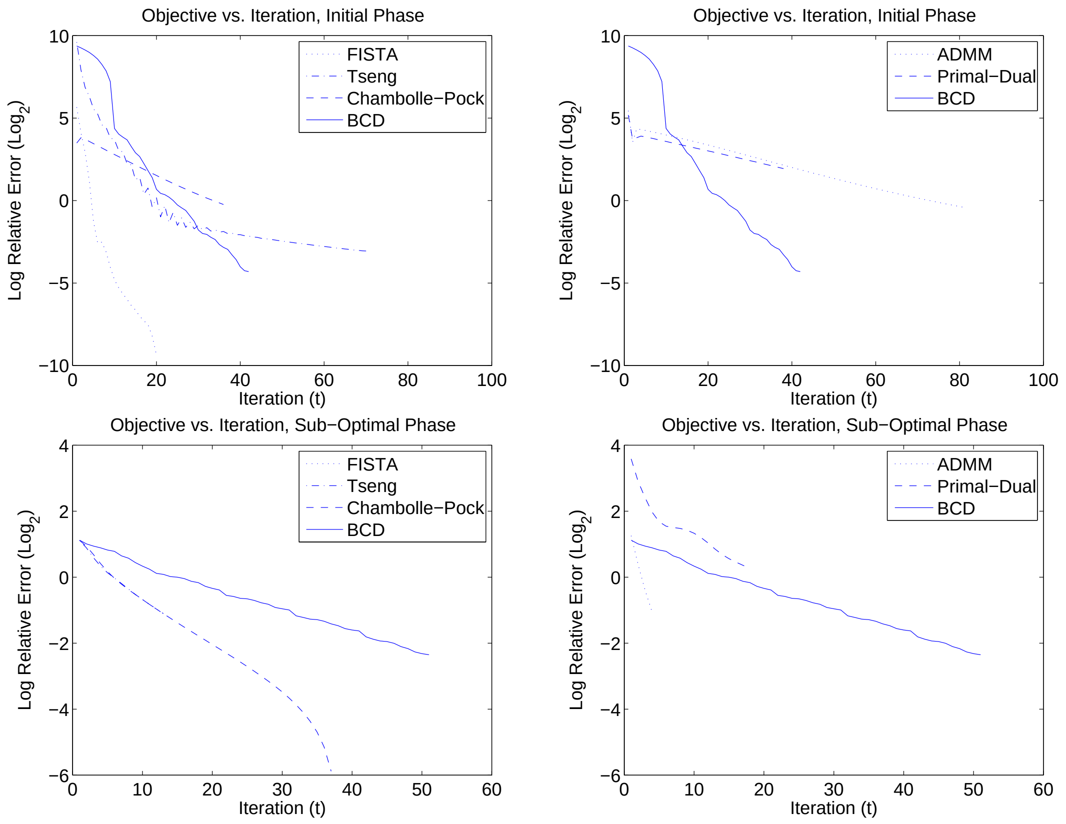


Figure 3.4: *Iterations for the relative objective of the convex sub-problem in 2-D randomly generated sparsity pattern experiment. Splitting methods are separated into two categories.*

Figure 3.4 shows the relative objective versus iteration plot for these methods. We did not include Spingarn's method and Chen-Teboulle semi-implicit method because their convergence speeds are much slower than the other methods. FISTA on the dual, Chambolle-Pock semi-implicit, Tseng's forward-backward methods perform better than BCD in both phases similarly to the 1-D experiment. Comparing with BCD, these splitting methods are only slightly better in terms of the convergence rate per iteration because The 2-D experiment is much larger than the 1-D experiment. ADMM still performs as well as BCD and in some cases better.

Methods	ADMM	Tseng	FISTA	Chambolle-Pock	BCD
Runtime (ms)	35.7	35.0	36.3	31.2	182

Table 3.3: *Average runtime per iteration in 2-D randomly generated sparsity pattern experiment, averaging over the entire alternating descent algorithm. We only compare four splitting methods faster than BCD during the alternating descent.*

Table 3.3 shows the average runtime for one iteration of these methods. In terms of convergence speed, BCD is much more complex than all splitting methods. In a larger problem, the preprocessing step necessary for BCD becomes much more complex. Furthermore, the proximal operator evaluations are highly suitable for parallel computing and can be easily optimized automatically in MATLAB. Therefore, these splitting methods are much better than BCD, especially Chambolle-Pock semi-implicit method, in terms of efficiency per iteration.

Alternating Descent

We consider the 2-D alternating descent experiment in a similar way of the 1-D experiment. In this experiment, we did not include BCD method with a fixed number of iteration as a stopping criterion. In a large scale problem, to fix a large number of iteration for BCD is almost impossible to compute in a reasonable time.

Figure 3.5 shows the results of alternating descent for these methods. Each row of image contains all 2-D structured sparse basis vectors for one method in descending order of their average absolute magnitude. The color gray $(128, 128, 128)$ stands for zero, or sparse elements in the variables. Color black $(0, 0, 0)$ and white $(255, 255, 255)$ stand for the negative and positive maximum of the principal components. Comparing to the ground truth in the last row, all the splitting methods

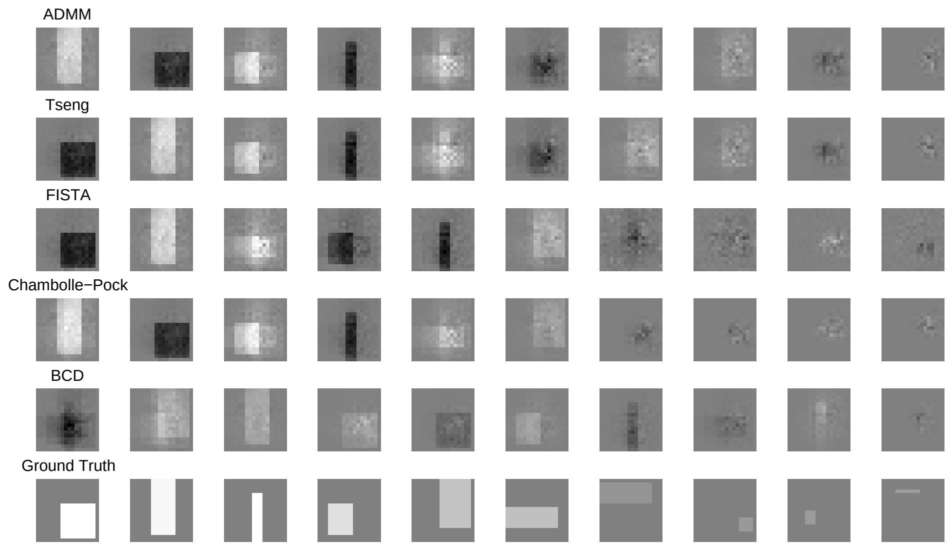


Figure 3.5: *SSPCA results of four splitting methods and BCD comparing with the ground truth in 2-D randomly generated sparsity pattern experiment. Each image in one row corresponds to one principal component. Each row corresponds to the result of alternating descent with that particular method for the convex sub-problem.*

capture the structured sparsity patterns with larger magnitudes (Basis vectors 1 through 5) while the smaller ones are most likely to be lost due to the high SNR (Basis vectors 6 through 10). Basically, the larger the magnitude, the cleaner the structured sparsity pattern is. BCD failed to capture the patterns most likely because it converges to a different local minimum.

Figure 3.6 shows the relative objective versus iteration plot for the alternating descent algorithm. Together with results from Figure 3.5, we further confirmed that most of the splitting methods converge to the same local minimum while BCD converges to a different local minimum. Interestingly, FISTA is not performing as well as other splitting methods, most likely because the step size for FISTA is limited by a large Lipschitz continuity parameter. This small step size might introduce numerical problems.

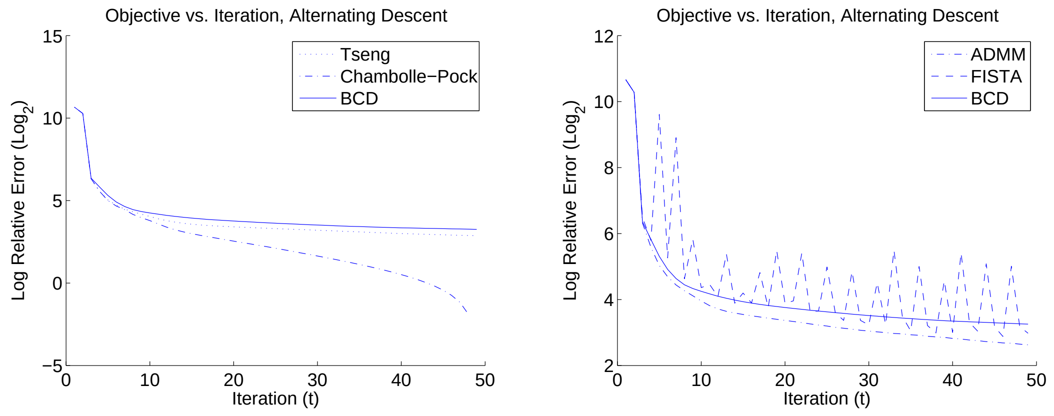


Figure 3.6: *Iterations for the relative objective of the alternating descent in 2-D randomly generated sparsity pattern experiment. Splitting methods are separated into two categories.*

Methods	ADMM	Tseng	FISTA	Chambolle-Pock	BCD
Iterations	7.51	19.3	23.3	57.0	49.0

Table 3.4: *Average iteration count per alternating descent step in 2-D randomly generated sparsity pattern experiment. The numbers of inner iterations necessary for these methods to solve a convex sub-problem.*

Table 3.4 shows the average iteration per alternating descent step for these methods. Most of the splitting methods use much less iteration per alternating descent than BCD. Except for Chambolle-Pock semi-implicit method.

To summarize our observations from the 2-D group sparsity pattern experiments, splitting methods presented above not only give better results capturing more structured sparsity patterns than BCD. Most of the splitting methods are significantly better than BCD in terms of speed and rate of convergence.

3.5.3 AR Face Recognition

We apply these splitting methods to the SSPCA application for the AR Face Database[1] created by Aleix Martinez and Robert Benavente [MB98]. This database consists of face images of 100 different individuals. There are 14 non-occluded and 12 occluded images per subject. The goal is to find r leading principal components, such that the principal components for different subjects are separated with 2-D continuous structure sparsity pattern. This experiment is a good example to show how these splitting methods performed in real application.

Intuitively, choosing the correct the number of columns r, or the number of principal components is very important in sparse matrix factorization problem. It will slow down speed of convergence if it is not correctly estimated. In previous experiments, number of principal components r are predetermined by the ground truth. However, in real SSPCA application, the data points are considered very close to each other and hence making it difficult to estimate the correct number of principal components. The starting point of the alternating descent is also a key factor to speed of convergence. If the starting point is randomly generated without any selection, it might generate a close-to-singular decomposition matrix during the initial phase.

In our experiment, first we downscale the face images into 20×20 images. Then we use 14 non-occluded face image as training data and obtain $r = 7$ principal components for each subject. These principal components can be used to test over the remaining 12 occluded face images to verify the recognition results.

Figure 3.7 shows the results of the alternating descent for the one of the

[1]The AR Face was created by Aleix Martinez and Robert Benavente, which can be found on `http://www2.ece.ohio-state.edu/~aleix/ARdatabase.html`.

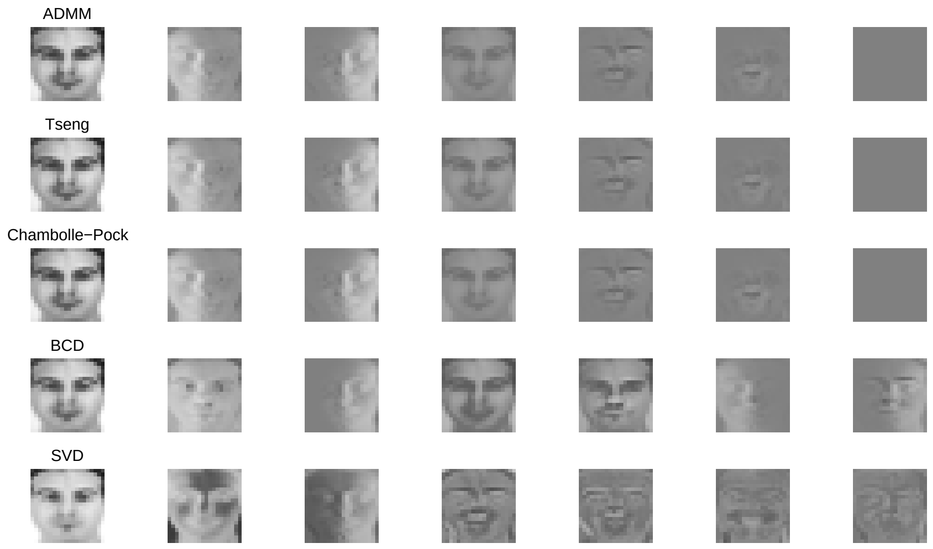


Figure 3.7: *SSPCA results of three splitting methods and BCD comparing with the SVD in AR Face database application. Each image in one row corresponds to one principal component. Each row corresponds to the result of alternating descent with that particular method for the convex sub-problem.*

subjects. It is similar to the 2-D continuous pattern experiment. There is no ground truth in general. Instead, we use the SVD of the original data as a reference. Comparing the results from BCD, the splitting methods (ADMM, Tseng's forward-backward and Chambolle-Pock semi-implicit) successfully capture the left and right half of the faces (Image 2 and 3) due to shadowing, and the emotion changes like laughing (Image 6 on the mouths) and smiling (Image 5 on the eyes and mouths). However, the BCD method failed to capture the emotions, and the left and right shadowing image are separated in different principal components.

Figure 3.8 shows the relative objective versus iteration plot for the alternating descent algorithm. Together with the results from Figure 3.7, we conclude that BCD converges to a different local minimum. The reason is similar to the 2-D randomly generated sparsity pattern experiment: BCD does not have good stopping criterion and a trivial stopping condition will cause early termination when

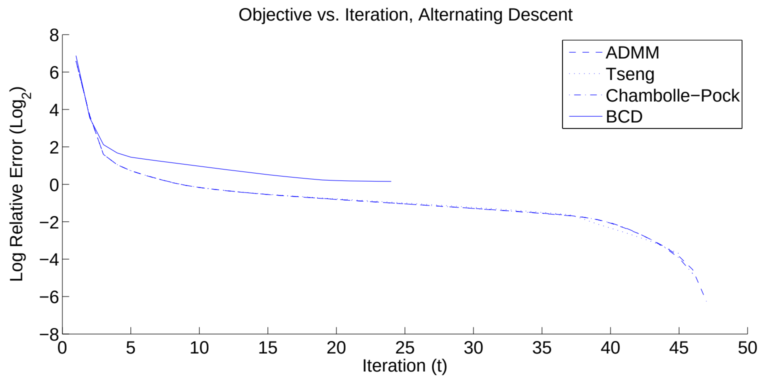


Figure 3.8: *Iterations for the relative objective of the alternating descent in AR Face database application.*

Methods	ADMM	Tseng	Chambolle-Pock	BCD
Runtime (ms)	52.1	51.1	41.3	288

Table 3.5: *Average runtime per iteration in AR Face database application.*

solving the convex sub-problems. All three splitting methods presented above perform similarly. However, FISTA does not work in this experiment due to the ill-conditioned convex sub-problem during the alternating descent. The other splitting methods are considerably slower than the three methods presented. They are not included in our discussion of experimental results.

Table 3.5 and 3.6 show the average runtime per iteration and the average iteration count per alternating descent step for these methods. Although BCD spends much more time per iteration due to the size of the problem, BCD uses much

Methods	ADMM	Tseng	Chambolle-Pock	BCD
Iterations	362	152	230	17

Table 3.6: *Average iteration count per alternating descent step in AR Face database application.*

less iterations per alternating descent step than other splitting methods. This is because of the early termination discussed before, which is also the main reason that BCD converges to a wrong local minimum.

To summarize our observations from the AR-Face recognition application, BCD does not converge to the desired local minimum while the selected splitting methods provided the desired results and performed much faster than BCD. Therefore, to summarize the experiments, choosing the appropriate splitting methods for the alternating descent algorithm makes it much faster than using the existing methods such as BCD. Most importantly, using splitting methods for alternating descent algorithm achieves the desired result of the sparse matrix factorization problem.

CHAPTER 4

Sparse Hierarchical Dictionary Learning

4.1 Problem Description

In many applications in signal processing, the observed signal can be represented by a decomposition of a series of dictionary components. One famous example is the Fourier Transformation. Instead of using a predefined dictionary, e.g., Fourier basis or Wavelet basis, the dictionary learning problem tries find such a dictionary from the training data. As an example, consider a discrete signal $x \in \mathbb{R}^n$ and a finite dictionary D with components $D^i \in \mathbb{R}^n$, $i = 1, 2, \ldots, r$. The signal x can be represented exactly or approximately by a decomposition v of this dictionary:

$$\begin{aligned} \hat{x} &= Dv \\ &= \sum_{i=1}^{r} v_i D^i, \end{aligned}$$

such that $\hat{x} \cong x$. Mathematically, it is very similar to the PCA problem. Namely, given a set of training signals in a matrix $X \in \mathbb{R}^{n\times m}$, the goal of the dictionary learning problem is to find a finite dictionary $D \in \mathbb{R}^{n\times r}$ and the corresponding decompositions $V \in \mathbb{R}^{r\times m}$, such that these training signals can be reconstructed by $\hat{X} = DV$, and the total reconstruction error is minimized: $\hat{X} \cong X$. The dictionary learning problem has been adapted to various applications such as unsupervised learning, pattern classification, and compressive sensing.

In some applications, it is interesting to limit the number of components in the dictionary used to reconstruct the signal while minimizing the overall reconstruction error. In other words, the decomposition matrix V is column-wise sparse.

This is known as sparse dictionary learning (SDL). The SDL problem is known to be computationally challenging. Many algorithms have been developed for SDL problems. Olshausen et al. [OF97] proposed a second-order iterative batch procedure algorithm for general SDL problems. Lee et al. [LBR07] proposed a general first-order gradient descent method specifically for sparse coding.

Unlike the predefined dictionaries, dictionary components acquired from SDL might be meaningless since the learning process is highly unsupervised. Components in the dictionary have no explicit connections with each other. Therefore, the structure of the decomposition matrix V, which indirectly promotes relationships among dictionary components, is as important as the sparsity condition. Jenatton et al. [JMO10] proposed a hierarchically structured sparse decomposition of V to further exploit relationships among dictionary components and maintain the sparsity in the decomposition. With properly chosen constraints and penalty functions, this sparse hierarchical dictionary learning (SHDL) problem can be formulated as a sparse matrix factorization problem.

4.2 Matrix Factorization Formulation

In this thesis, we represent the SHDL problem based on the formulation from Jenatton et al. [JMO10]:

$$\begin{array}{ll} \text{minimize} & \frac{1}{2}\|X - DV\|_F^2 + \lambda \sum_{i=1}^{m} g(V^i) \\ \text{subject to} & h(D^j) \leq 1, \quad j = 1, 2, \ldots, p, \end{array}$$

The m columns of the data matrix $X \in \mathbb{R}^{n \times m}$ correspond to m sample signals in $\mathbb{R}^n$. The p columns of the dictionary matrix $D \in \mathbb{R}^{n \times p}$ correspond to p different dictionary components in $\mathbb{R}^n$. The m columns of the decomposition matrix

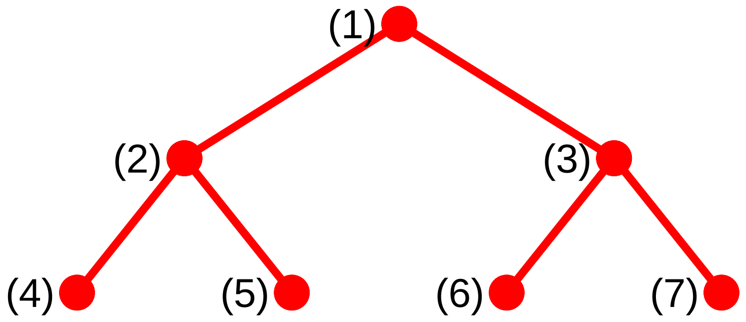


Figure 4.1: *Example of a hierarchical tree with 7 variables*

$V \in \mathbb{R}^{p\times m}$ correspond to m decomposition vectors in $\mathbb{R}^p$. This is a variation of the sparse matrix factorization problem (Eq. 1.2). The major difference would be the row-wise penalties over decomposition matrix V^T. It will affect the formulation and separability of the problem as well as the optimality conditions. However, it is still recognized as a sparse matrix factorization problem (Eq. 1.1).

The hierarchical group sparsity inducing norm $g(V^k)$ in SHDL, proposed by Jenatton et al. [JMO10], is known to exploit sparsity in a hierarchical architecture. To exploit such sparsity, Jenatton et al. proposed the hierarchical group sparsity inducing norm in the format of (Eq. 2.2):

$$\begin{aligned} g(x) &= \sum_{i\in\Delta} \|x_{\delta_i}\|_2 \\ &= \sum_{i\in\Delta} \sqrt{\sum_{j\in\delta_i} x_j^2}. \end{aligned} \tag{4.1}$$

In (Eq. 4.1), Δ is a hierarchical sparsity tree where δ_i is a subtree of Δ rooted at node i. For example, consider a hierarchical tree Δ for $x = (x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \mathbb{R}^7$. Figure 4.1 shows the hierarchical tree Δ for this example. There are 7 subtrees in Δ. The subtree rooted at node 2 is $\delta_2 = \{2, 4, 5\}$. The subtree rooted at node 4 contains only one node, $\delta_4 = \{4\}$. The subtree rooted at node 1 is the entire tree. The hierarchical group sparsity norm (Eq. 4.1) for x with this particular tree structure is:

$$\begin{aligned}
g(x) &= \sum_{i\in\Delta}\sqrt{\sum_{j\in\delta_i} x_j^2} \\
&= |x_4| + |x_5| + |x_6| + |x_7| + \\
&\quad \sqrt{x_2^2 + x_4^2 + x_5^2} + \sqrt{x_3^2 + x_6^2 + x_7^2} + \\
&\quad \sqrt{x_1^2 + x_2^2 + x_3^2 + x_4^2 + x_5^2 + x_6^2 + x_7^2}.
\end{aligned}$$

Other constraints and penalties on the dictionary D and decomposition V might also apply in SHDL application. They typically depend on the specific application. Jenatton et al. [JMO10] proposed a general constraint for the dictionary components D^k:

$$h(D^k) = (1-\mu)\|D^k\|_2^2 + \mu\|D^k\|_1 \leq 1, \tag{4.2}$$

where $\mu \in (0,1)$. For example, Jenatton et al. proposed to use $\mu = 0$ or the l_2 norm for image processing and $\mu = 1$ or the l_1 norm for text mining.

Additionally, the decomposition vectors and/or dictionary components might be restricted in the non-negative orthant $\mathbb{R}_+$ in some applications. It can be represented as element-wise constraints $D \geq 0$ and/or $V \geq 0$.

Consider the typical SHDL formulation,

$$\begin{aligned}
\text{minimize} \quad & \frac{1}{2}\|X - DV\|_F^2 + \lambda\sum_{k=1}^{m} g(V^k) \\
\text{subject to} \quad & \|D^k\|_2 \leq 1, \quad k = 1, 2, \ldots, p, \\
& V \geq 0.
\end{aligned} \tag{4.3}$$

Here we choose $\mu = 0$ in (Eq. 4.2) to formulate the l_2 norm constraint in dictionary components. We also choose an element-wise non-negative constraint on the decomposition matrix. This formulation is the same as the example discussed by

Jenatton et al. [JMO10]. It can be formulated as two convex sub-problems through alternating descent algorithm (Eq. 1.6): The convex minimization problem over the dictionary matrix D with a fixed decomposition matrix V,

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|X - DV\|_F^2 \\ &\text{subject to} \quad \|D^k\|_2 \leq 1, \quad k = 1, 2, \ldots, p, \end{aligned} \tag{4.4}$$

and the convex minimization problem over the decomposition matrix V with a fixed dictionary matrix D,

$$\begin{aligned} &\text{minimize} \quad \frac{1}{2}\|X - DV\|_F^2 + \lambda \sum_{k=1}^{m} g(V^k) \\ &\text{subject to} \quad V \geq 0. \end{aligned} \tag{4.5}$$

These two problems are almost identical to the two convex sub-problems in (Eq. 3.4, 3.5). Therefore, most of the analysis for SSPCA apply to SHDL as well. Section 4.3 and Section 4.4 discuss the major differences between SHDL and SSPCA.

4.3 Minimization over Dictionary Matrix

Consider the convex minimization problem in (Eq. 4.4). It can be written as

$$\text{minimize} \quad \tilde{f}(D) + \tilde{h}(D)$$

with the two parts of the objective function as:

$$\tilde{f}(D) = \frac{1}{2}\|X - DV\|_F^2 \tag{4.6}$$

$$\tilde{h}(D) = \sum_{k=1}^{p} I_{\|\cdot\|_2 \leq 1}(D^k), \tag{4.7}$$

This convex problem (Eq. 4.4) is very similar to the problem (Eq. 3.4) for SSPCA. The proximal operators necessary for splitting methods are almost the same as those discussed in Section 3.3. The best method applied to this convex sub-problem of SHDL is also a projected gradient methods with the proximal operator as a projection onto a l_2 norm ball. Therefore, the proximal gradient methods and accelerated proximal gradient methods discussed in Section 3.3 for SSPCA can be applied to SHDL.

4.4 Minimization over Sparse Decomposition Matrix

The problem (Eq. 4.5) is a penalized least squares problem with $g(V^j)$ as the hierarchical sparsity inducing norm in the format of (Eq. 2.2) . It is similar to the problem discussed in Section 3.4 for SSPCA. The hierarchical sparsity inducing norm (Eq. 4.1) can be represented in the exact same non-overlapping group norm in (Eq. 3.8), such that

$$\tilde{g}(x) = \sum_{i=1}^{|\Delta|} \|x_{(i)}\|_2 \tag{4.8}$$

with $x_{(i)} = V^k_{\delta_i}$ and $x = (x^T_{(1)}, x^T_{(2)}, \dots, x^T_{(|A|)})^T$, where $x_{(i)}$ is the vector containing all variables in the hierarchical subtree δ_i. Therefore, the vector x is also the linear transform of V^k identical to (Eq. 3.9):

$$x = CV^k = \begin{bmatrix} V^k_{\delta_1} \\ V^k_{\delta_2} \\ \vdots \\ V^k_{\delta_{|V|}} \end{bmatrix}, \quad \forall \delta_i \in \Delta. \tag{4.9}$$

The weighted version of the linear transform (Eq. 3.10) can be adapted in a similar manner, which we do not discuss in detail here. The problem (Eq. 4.5) can be reformulated as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X - DV\|_F^2 + \lambda \sum_{k=1}^{m} \sum_{i=1}^{|\Delta|} \|W_{(i)}^k\|_2 \\ \text{subject to} \quad & W = CV, \\ & V \geq 0. \end{aligned}$$

The major difference between SHDL and SSPCA is the additional non-negative constraint on the decomposition matrix $V \geq 0$. Fortunately, this constraint $V \geq 0$ is equivalent to $W \geq 0$ because of the property of matrix C discussed in Section 3.4. The problem (Eq. 4.5) can be further reformulated as

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2}\|X - DV\|_F^2 + \lambda \sum_{k=1}^{m} \sum_{i=1}^{|\Delta|} \|W_{(i)}^k\|_2 \\ \text{subject to} \quad & W = CV, \\ & W \geq 0. \end{aligned} \tag{4.10}$$

Denote the two parts of the objective in (Eq. 4.10) as:

$$\tilde{f}(V) = \frac{1}{2}\|X - DV\|_F^2 \tag{4.11}$$

$$\tilde{g}(W) = \lambda \sum_{k=1}^{m} \sum_{i=1}^{|\Delta|} \|W_{(i)}^k\|_2, \quad W \geq 0. \tag{4.12}$$

Then the problem can be written as

$$\begin{aligned} \text{minimize} \quad & \tilde{f}(V) + \tilde{g}(W) \\ \text{subject to} \quad & W = CV. \end{aligned}$$

This is the exact same format as in SSPCA application with different functions $\tilde{f}$ and $\tilde{g}$. Most of the splitting methods discussed for SSPCA are suitable for SHDL as well. The formulation (Eq. 4.10) for SHDL is separable over the columns of V, which makes it more suitable for parallel computing. Both the Frobenius norm term and the column-wise hierarchical sparsity inducing norm are column-wise independent. Solving the problem (Eq. 4.10) is equivalent to solving m individual regularized least squares problems for every columns of X^k, $k \in (1, 2, \ldots, m)$:

$$\begin{aligned} \text{minimize} \quad & \tilde{f}(V^k) + \tilde{g}(W^k) \\ \text{subject to} \quad & W^k = CV^k, \end{aligned}$$

We use the same notation $\tilde{f}$ and $\tilde{g}$ for the functions because the vectors V^k and W^k can be treated as a matrix with only one column.

4.4.1 Proximal Operators

The properties of the linear mapping $W = CV$ is the same as in SSPCA application. The first resolvent is the proximal operator for quadratic loss function (Eq. 4.11) with respect to vector V^k. It is very similar to the analysis in Section 3.4 as well. In this section, we focus on the second proximal operator with the extra non-negative constraint.

The second resolvent is the proximal operator for the non-overlapping l_1-l_2 norm function (Eq. 4.12) with an element-wise non-negative domain:

$$\begin{aligned} & W^k = \operatorname{prox}_{t\tilde{g}}(\hat{W}^k) \\ \Longleftrightarrow \quad & W^k = \underset{W^k \geq 0}{\arg\min} \left\{ \lambda \sum_{i=1}^{|\Delta|} \|W^k_{(i)}\|_2 + \frac{1}{2t} \|W^k - \hat{W}^k\|_2^2 \right\} \\ \Longleftrightarrow \quad & W^k_{(i)} = \underset{W^k_{(i)} \geq 0}{\arg\min} \left\{ \lambda \|W^k_{(i)}\|_2 + \frac{1}{2t} \|W^k_{(i)} - \hat{W}^k_{(i)}\|_2^2 \right\} \\ \Longleftrightarrow \quad & W^k_{(i)} = \underset{W^k_{(i)} \geq 0}{\arg\min} \left\{ \|W^k_{(i)}\|_2 + \frac{1}{2\lambda t} \|W^k_{(i)} - \hat{W}^k_{(i)}\|_2^2 \right\}. \end{aligned}$$

Each $W^k_{(i)}$ equals to the proximal operator of a scaled l_2 norm function with the domain constraint. It has a closed form solution:

$$W_{(i)}^k = \begin{cases} \left(1 - \dfrac{\lambda t}{\|\hat{W}_{(i)+}^k\|_2}\right)\hat{W}_{(i)+}^k & \|\hat{W}_{(i)+}^k\|_2 \geq \lambda t \\ 0 & \text{otherwise,} \end{cases}$$

where $\hat{W}_{(i)+}^k = \max\{0, \hat{W}_{(i)}^k\}$, the non-negative part of the vector $\hat{W}_{(i)}^k$.

The proximal operators for both parts of the objective are easy to calculate with closed form solution. Therefore, most of the methods discussed in Section 3.4 are also suitable for SHDL. However, some characteristics and properties of SHDL problem are different from those of SSPCA. We will discuss the difference for each method in detail in the rest of this section. For all these splitting methods, we denote v and w as the primal variable, z as the dual multiplier for the equality constraint $w = Cv$. Lower case variables indicate vector operation since columns of V is separable. The variable x represents the corresponding column of data from the original formulation.

4.4.2 Primal Methods

Consider reformulating the equality constraint of the problem (Eq. 4.10) into the subspace representation:

$$\begin{array}{ll} \text{minimize} & \tilde{f}(v) + \tilde{g}(w) \\ \text{subject to} & \begin{bmatrix} -I & C \end{bmatrix} \begin{bmatrix} w \\ v \end{bmatrix} = 0, \end{array}$$

This is identical to the problem formulation (Eq. 3.14) for the Spingarn's method.

Douglas-Rachford Splitting

The Spingarn's method discussed in SSPCA can be directly applied to SHDL application with the appropriate proximal operators and the linear transformation matrix C. The formulations are identical to those in (Eq. 3.15-3.17).

4.4.3 Primal-Dual Methods

Consider the primal-dual optimality condition in (Eq. 2.17),

$$0 \in \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} v \\ z \end{bmatrix} + \begin{bmatrix} \partial \tilde{f}(v) \\ \partial \tilde{g}^*(z) \end{bmatrix}.$$

We consider the trivial splitting of the monotone inclusion as $0 \in A(v, z) + B(v, z)$ with

$$A(v, z) = \begin{bmatrix} 0 & C^T \\ -C & 0 \end{bmatrix} \begin{bmatrix} v \\ z \end{bmatrix}, \qquad B(v, z) = \begin{bmatrix} \partial \tilde{f}(v) \\ \partial \tilde{g}^*(z) \end{bmatrix}.$$

Douglas-Rachford Splitting

In SHDL, the optimality condition and the splitting of operator are identical to those in SSPCA. The detail of the Douglas-Rachford splitting methods is not discussed again. The formulations are identical and discussed in (Eq. 3.18-3.20).

Forward-Backward Splitting

The modified forward-backward method for SHDL application has the exact same formulation as SSPCA. The step size is limited by the inverse of the norm of matrix C, $t \leq 1/\|C\|_2$ for Tseng's forward-backward splitting algorithm. It does not involve any of the proximal operators for the objective function. Fortunately, the matrix C for SHDL application has a small Lipschitz continuity parameter just like SSPCA. The formulations are identical to those in (Eq. 3.21-3.24).

Semi-Implicit Splitting

In SHDL, both of the semi-implicit splitting method from Chambolle and Pock, and method from Chen and Teboulle are identical to the formulation in SSPCA. The step size for both methods are also limited by an upper bound proportional to the inverse of the norm of matrix C. The formulations are identical to those in (Eq. 3.25-3.27) and (Eq. 3.28-3.31).

4.4.4 Dual Methods

Consider the dual optimality condition,

$$0 \in -C\partial\tilde{f}^*(-C^T z) + \partial\tilde{g}^*(z).$$

We also consider the trivial splitting of the monotone inclusion as $0 \in A(z) + B(z)$ with

$$A(z) = -C\partial\tilde{f}^*(-C^T z), \qquad B(z) = \partial\tilde{g}^*(z).$$

Douglas-Rachford Splitting

The Douglas-Rachford splitting method applying to the dual optimality condition is the ADMM method as discussed in SSPCA. The primal formulation are similar to those in (Eq. 3.32-3.34):

$$\begin{aligned}
v^+ &= \arg\min_v \left\{ \tilde{f}(v) + \frac{t}{2}\|Cv - w + z/t\|_2^2 \right\} \\
w^+ &= \arg\min_w \left\{ \tilde{g}(w) + \frac{t}{2}\|Cv^+ - w + z/t\|_2^2 \right\} \\
&= \mathrm{prox}_{\tilde{g}/t}(Cv^+ + z/t) \\
z^+ &= z + t(Cv^+ - w^+).
\end{aligned}$$

The first update requires the solution for

$$(D^T D + tC^T C)v = D^T x + tC^T(w - z/t),$$

which has a closed form solution:

$$v = (D^T D + tC^T C)^{-1}(D^T x + tC^T(w - z/t)). \tag{4.13}$$

The major difference between SHDL and SSPCA is that this convex sub-problem for SHDL application is column-wise independent. Instead of solving the general Sylvester equation, the explicit solution to the first update is just the solution for the linear equations. Therefore, ADMM for SHDL application is easier to formulate than for SSPCA application. However, there is only a marginal difference in terms of calculation complexity.

The dictionary components in SHDL generally require extra redundancy in order to make the decomposition matrix sparse and structured. Therefore, the matrix $D^T D$ is most likely to be singular or close to singular. It will cause numerical problem in solving the linear equation (Eq. 4.13). Therefore, the correct estimation of the size of the dictionary is very important for ADMM since an over-redundant dictionary could lead to singularity condition in $D^T D$, which limits the step size.

Forward-Backward Splitting

The forward-backward splitting method applied to the dual optimality condition is the proximal gradient method (Eq. 3.36). It requires $\tilde{f}^*$ to be differentiable and the gradient at $-C^T z$ is

$$\begin{aligned} \nabla \tilde{f}^*(-C^T z) &= \arg\min_v \left\{ \tilde{f}(v) - < v, -C^T z > \right\} \\ &= (D^T D)^{-1}(-C^T z + D^T x), \end{aligned}$$

for a non-singular matrix $D^T D$. In SSPCA, the matrix $D^T D$ is usually non-singular. However, this matrix $D^T D$ is most likely to be singular in SHDL application. Therefore, the proximal gradient method is not suitable for SHDL application.

4.5 Experiments

We first consider the 2-dimensional randomly generated images from certain basis vectors with colored rectangular region. Then we consider using SHDL to extract the dictionary for some images randomly selected from the Berkeley Segmentation Dataset [MFT01] to see the performance of the splitting methods in a practical application.

In these experiments, we first compare the efficiency of the splitting methods on one of the convex sub-problems of alternating descent discussed in Section 4.4). Then we compare them in the alternating descent algorithm (Eq. 4.3). We did not include BCD method from Jenatton et al. [JMO10] in the comparison. This BCD method uses a proximal algorithm to solve the dual problem for each independent column in the decomposition matrix. Since the problem is completely column-wise independent, it is almost identical to ADMM method applied to one

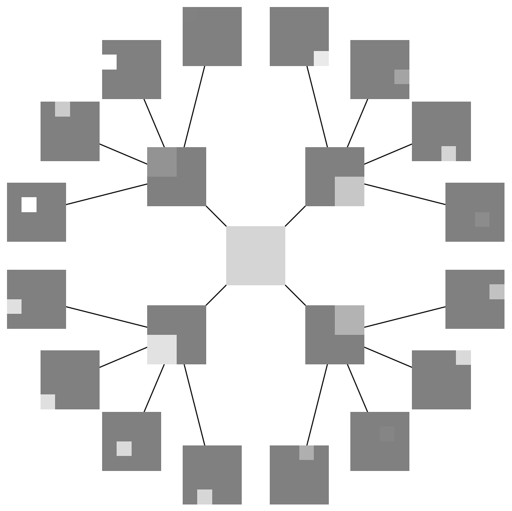


Figure 4.2: *Example of a randomly generated hierarchical dictionary tree. The image in the middle is the root of the tree and each non-leaf image has* 4 *child images.*

column of the decomposition matrix at a time.

4.5.1 Randomly Generated Image Dictionary Learning

In the randomly generated image example, we first generate a random hierarchical dictionary tree. This hierarchical dictionary tree is organized as follows: Each component of this hierarchical tree corresponds to a uniformly colored rectangular region within the image. The color of this region is randomly selected through a uniform distribution in $[0, 255]$. The rectangular region for the root node covers the entire image. Each non-leaf node in this tree has 4 children and their rectangular region cover the top-left, top-right, bottom-left, bottom-right part of their parent. Figure 4.2 shows one example of this dictionary tree with a maximum depth of 3.

Now we randomly generate training data images s using the hierarchical dictionary tree:

$$s = u_0 D_{(0)} + u_1 D_{(1)} + u_2 D_{(2)} + \epsilon \in \mathbb{R}^n. \tag{4.14}$$

The parameters u_i are randomly selected through a uniform distribution. The vectors $D_{(i)}$ are the 3 dictionary components in a randomly selected path from root node to one of the leaf nodes. The parameter ϵ is an independent identical distributed (IID) Gaussian noise such that the generated signal satisfies a predetermined signal-to-noise ratio (SNR).

Convex Sub-problem Minimization

Consider using the splitting methods to solve (Eq. 4.10). We generate $m = 200$ images in the size of 8×8 pixels. We assume the hierarchical tree has a maximum depth of 4 with a structure such that each node in level 0, 1, 2 has 10, 2, 2 children, respectively. These splitting methods perform very differently through the process of alternating descent algorithm. We also compare the splitting methods in two scenarios: Initial Phase and Sub-optimal Phase, which are consistent with the definition in SSPCA application.

Figure 4.3 shows the relative objective versus iteration plot for the convex subproblem discussed in Section 4.4. At the initial phase, the methods with respect to the primal-dual optimality conditions: Tseng's forward-backward and Chambolle-Pock semi-implicit algorithm works much better than methods using primal or dual optimality condition, such as ADMM and Spingarn's. Chen-Teboulle semi-implicit method still converges much slower than the other methods and is not included in the experiments. At the sub-optimal phase, it further confirms that methods with respect to the primal-dual optimality condition is much better than the other methods.

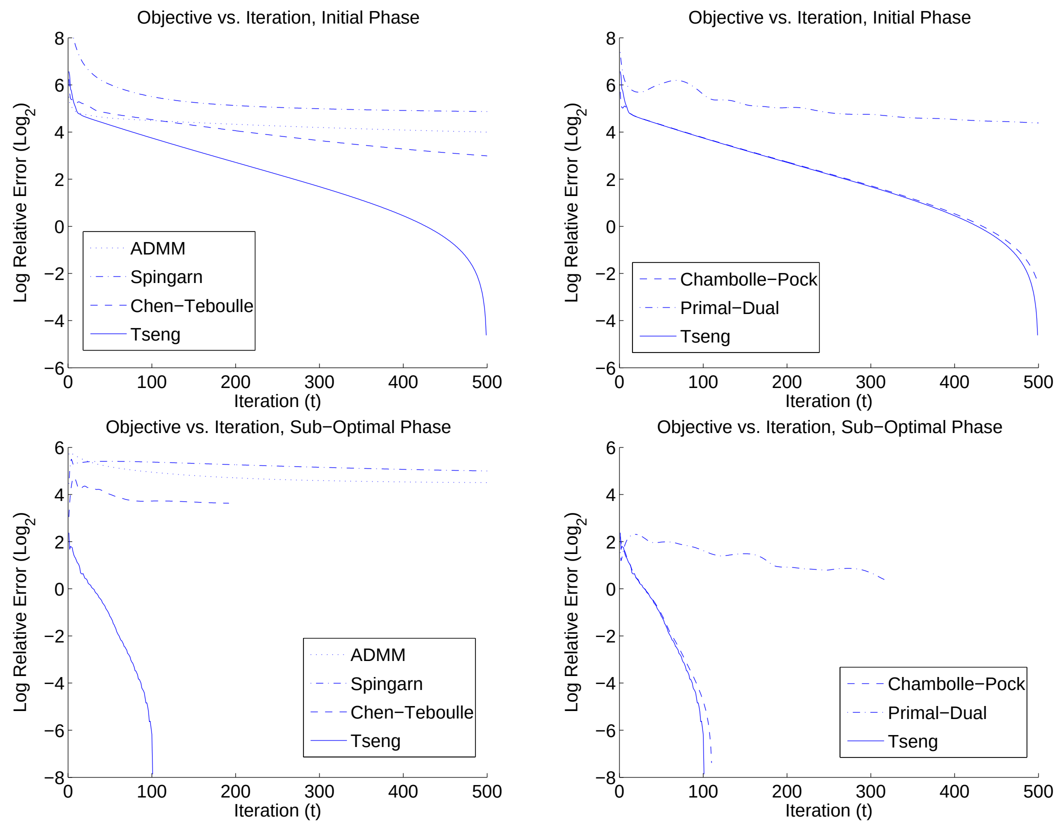


Figure 4.3: *Iterations for the relative objective of the convex sub-problem in randomly generated image experiment. Splitting methods are separated into two categories by there speeds of convergence.*

Methods	ADMM	Tseng	PD-DR	Chambolle-Pock
Runtime (ms)	11.2	11.6	13.4	17.4

Table 4.1: *Average runtime per iteration in randomly generated image experiment, averaging over the entire alternating descent algorithm. We only compare four splitting methods: primal-dual optimality methods and ADMM.*

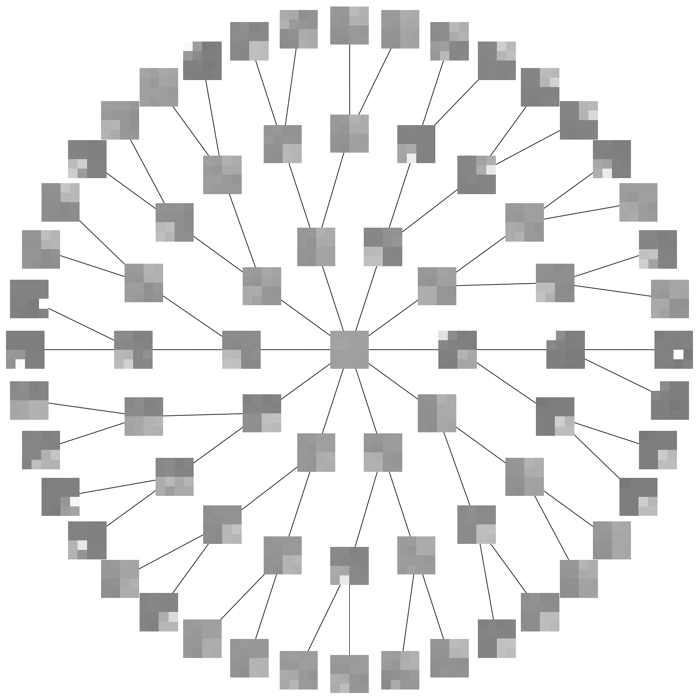
Table 4.1 shows the average runtime per iteration for these methods. Chambolle-Pock semi-implicit method takes a little bit more time each iteration that the other methods.

Alternating Descent

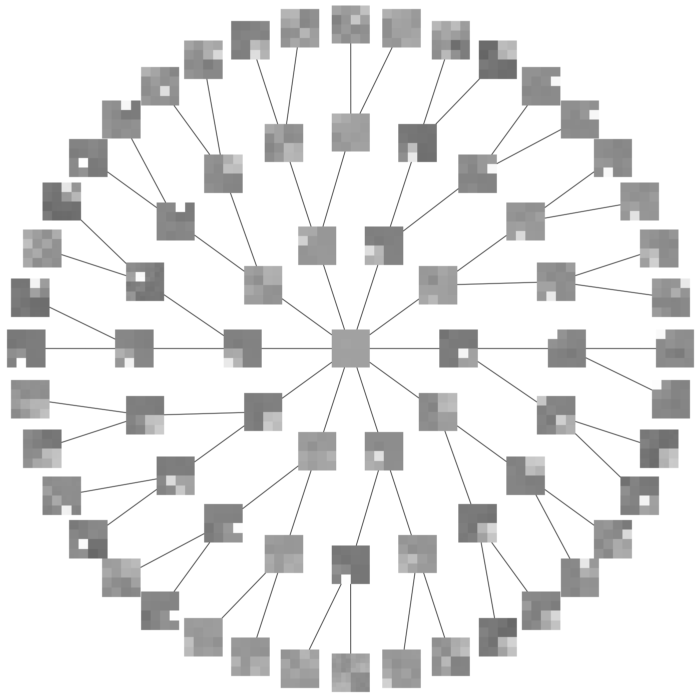
Consider applying these splitting methods to the alternating descent algorithm (Eq. 4.3). We assume the hierarchical tree structure and the other problem settings are the same as the convex sub-problem experiment. We only include the 3 methods based on primal-dual optimality and ADMM in the following experiments because the other methods are considerably slower.

Figure 4.4 shows the dictionaries obtained by the alternating descent algorithm for all these methods in a hierarchical tree structure. We can see that the result from ADMM is very similar to the result from Jenatton et al. [JMO10], which further confirms the similarity between ADMM and their BCD method. The decomposition for the training data is evenly spread over the entire dictionary on each level using ADMM. However, the training data images are randomly generated from 17 ground truth dictionary components as shown in Figure 4.2. It is not a good approximation of the ground truth because there are too many

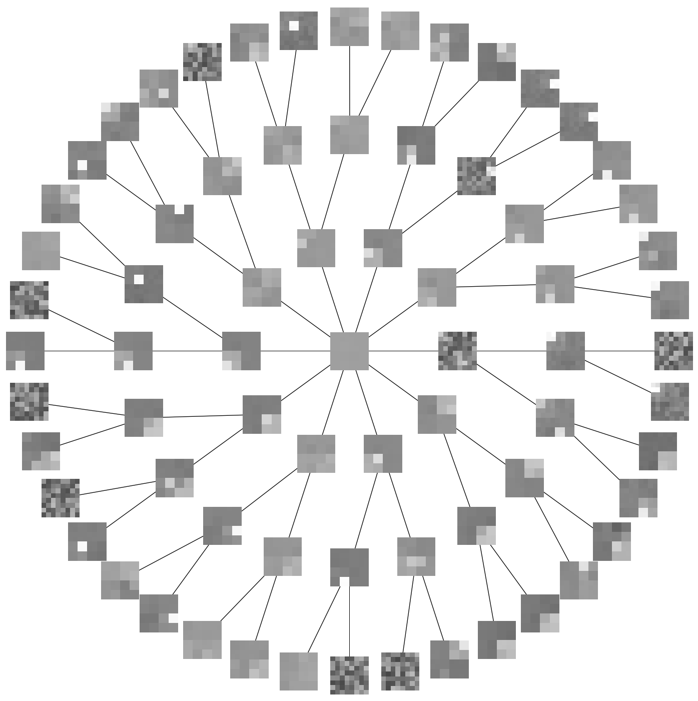
ADMM



Primal−Dual Douglas−Rachord



Tseng's Forward−Backward



Chambolle−Pock

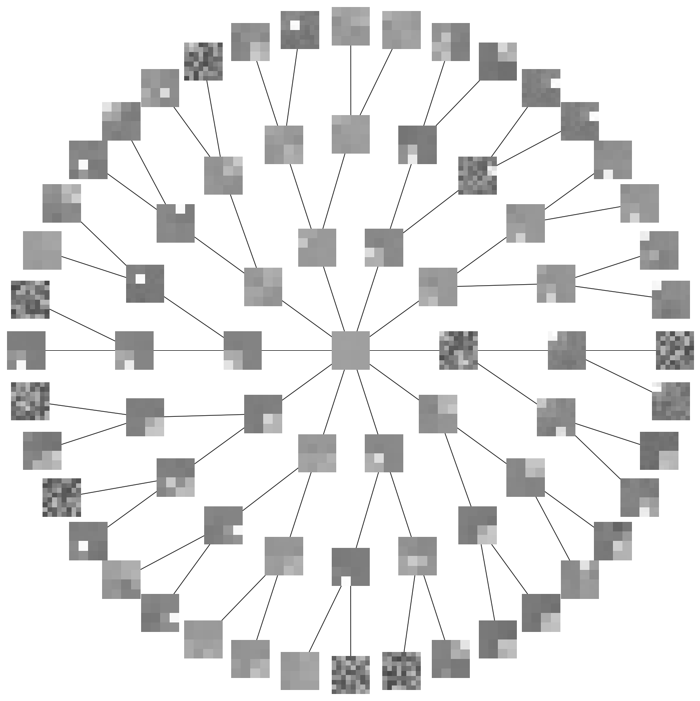


Figure 4.4: *SHDL results for the four methods in randomly generated image experiment. The root of the hierarchical tree is in the middle of each figure. The most outer circle of images is the leaf nodes of the hierarchical tree.*

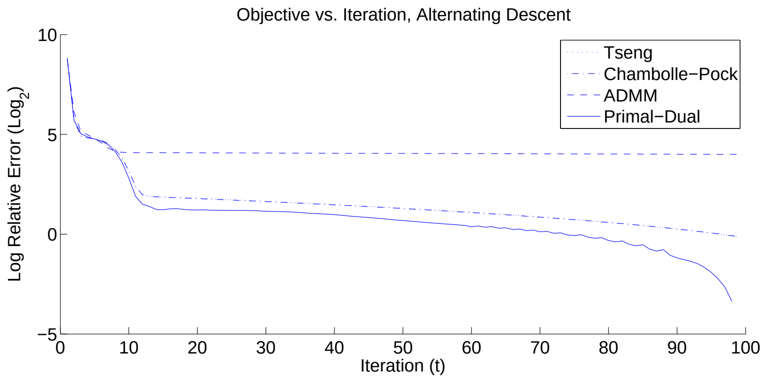


Figure 4.5: *Iterations for the relative objective of the alternating descent in randomly generated image experiment.*

redundant dictionary components.

The results for Tseng's forward-backward method and Chambolle-Pock semi-implicit method are very close to each other. They both have components in part of the tree as noisy images. These noisy dictionary components have almost negligible decomposition weights. For the primal-dual Douglas-Rachford method, we can make the same observation, except that the noisy images are more regularized. All of these methods produce a good hierarchical structure in the dictionary. Naturally these dictionary components are organized in groups of sectors, components closer to the root have lower frequencies or smoother changes while components closer to the leaf nodes have higher frequencies or sharper changes. For those primal-dual based methods, most of the noise images are located on the leaf nodes due to an overestimated number of dictionary components.

Figure 4.5 shows the relative objective versus iteration plot for alternating descent algorithm. It is consistent with the dictionary results shown in Figure 4.4. ADMM converges to a different local minimum from other methods. The Douglas-Rachford splitting method applied to the primal-dual optimality condition per-

Methods	ADMM	Tseng	PD-DR	Chambolle-Pock
Iteration	123	23.5	67.1	23.8

Table 4.2: *Average iteration count per alternating descent step in randomly generated image experiment. The numbers of inner iterations necessary for these methods to solve a convex sub-problem.*

forms slightly better than the other two methods. However, they all converge to the same local minimum.

Table 4.2 shows the average iteration count per alternating descent step for these methods. Tseng's forward-backward method performs the fastest overall, following by Chambolle-Pock semi-implicit method. Although the Douglas-Rachford splitting method has a faster convergence rate in Figure 4.5. It takes more iterations per alternating descent to solve the convex sub-problems. ADMM is the slowest. It takes much more iteration to compute one convex sub-problem on average.

To summarize the randomly generated image dictionary learning experiments, ADMM descent to a different local minimum from the other splitting methods presented in our experiments with a higher objective value. Also, ADMM takes much more time than the other splitting methods, and the resulting dictionary is over redundant. The other three methods based on primal-dual optimality condition perform similarly with each other. They are much faster and better than ADMM.

4.5.2 Berkeley Segmentation Dataset Dictionary Learning

We apply the splitting methods to solve a practical SHDL application. The training data used in this experiment are from the Berkeley Segmentation Dataset (BSD) [MFT01][1]. These data images are downscaled to a size of 8×8 pixels. They are highly uncorrelated and obtained in very different scenarios. It is interesting to see the dictionary learning results for these splitting methods in such a complicated dataset.

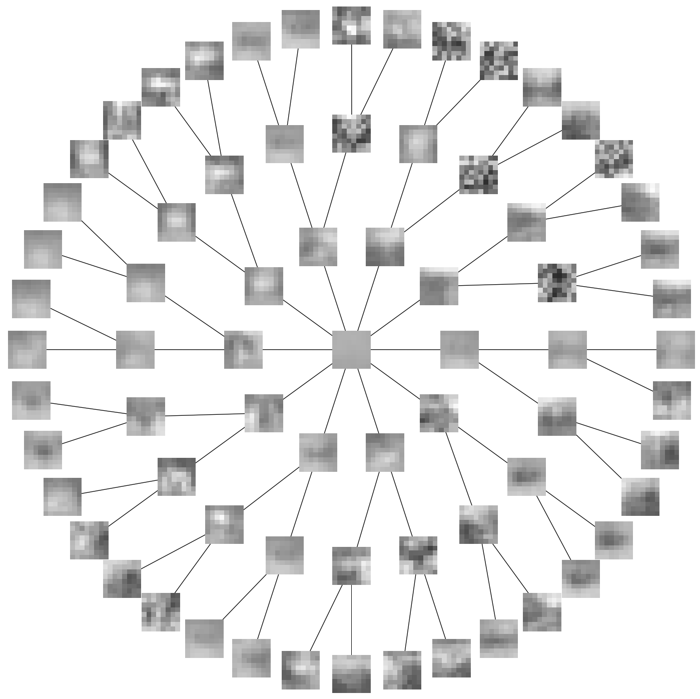
Figure 4.6 shows the dictionaries obtained by the alternating descent algorithm for all these splitting methods. All of these methods produce a fair hierarchical structure in the dictionary. Dictionary components with lower frequencies locate closer to the root while components with higher frequencies stay closer to the leaf nodes. Overall, the resulting dictionary from ADMM still has less details recovered and the dictionary components on the leaf nodes are not sharp enough. The results from the other three methods have more details (high frequency components) recovered on the leaf nodes.

Figure 4.7 shows the relative objective versus iteration plot for alternating descent algorithm. The most obvious observation is still that ADMM converges to a different local minimum than the other three methods based on the primal-dual optimality condition. Tseng's forward-backward method performs slightly better than Chambolle-Pock semi-implicit method. The primal-dual Douglas-Rachford splitting method has the slowest rate of convergence among the three.

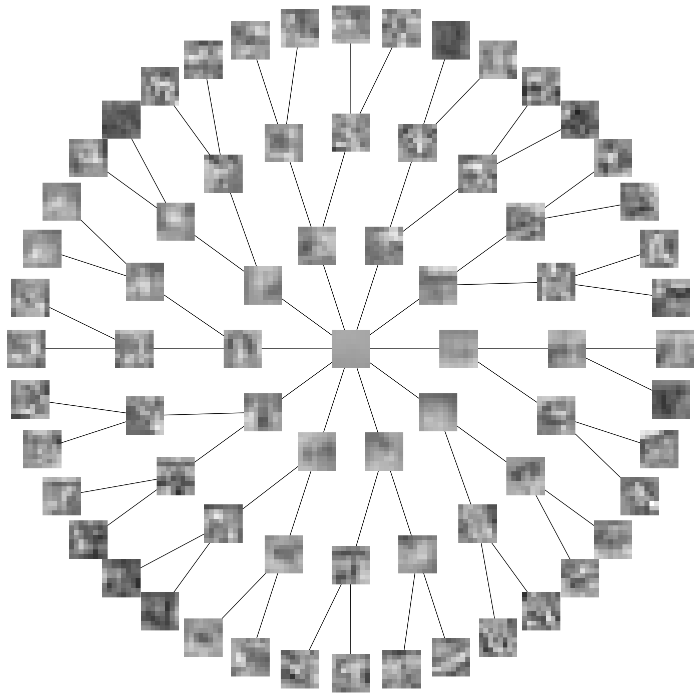
Table 4.3 and Table 4.4 show the average iteration count per alternating descent and average runtime per iteration for these splitting methods. ADMM is

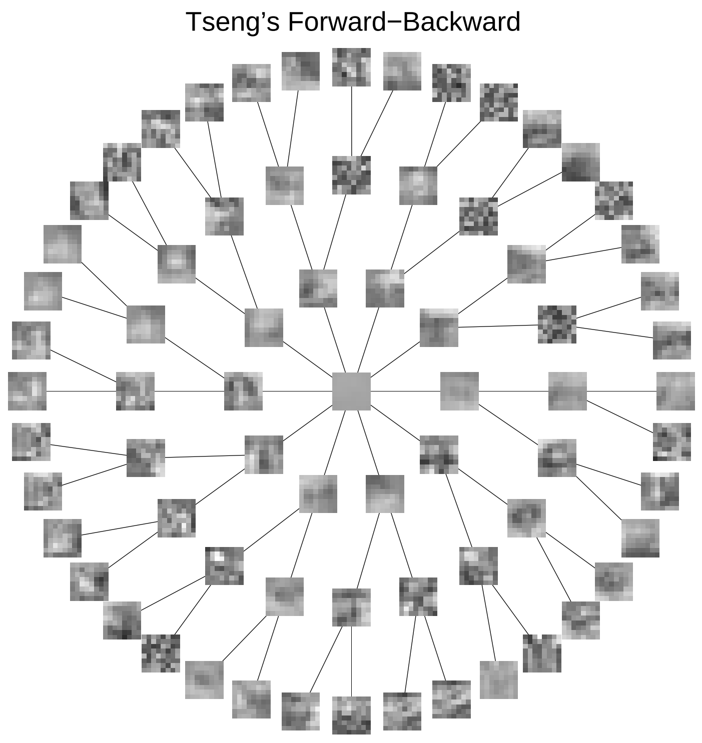
[1]The Berkeley Segmentation Dataset can be found at `http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/`

ADMM



Primal−Dual Douglas−Rachford





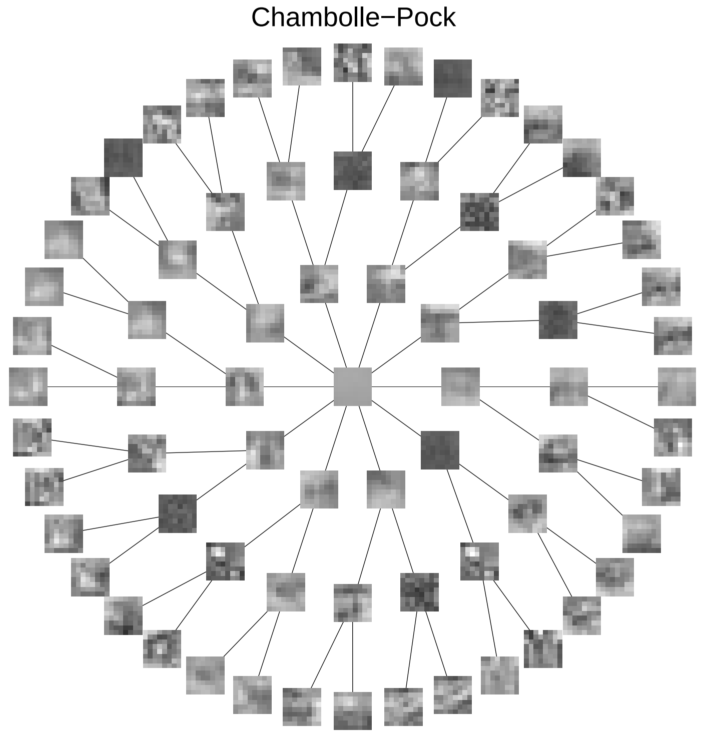


Figure 4.6: *SHDL results for the four methods in the BSD application. The root of the hierarchical tree is in the middle of each figure. The most outer circle of images is the leaf nodes of the hierarchical tree.*

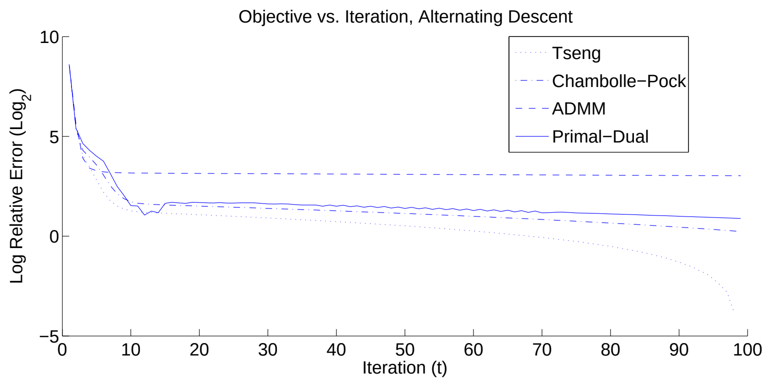


Figure 4.7: *Iterations for the relative objective of the alternating descent in the BSD application*

Methods	ADMM	Tseng	PD-DR	Chambolle-Pock
Iteration	11.2	11.9	12.5	18.4

Table 4.3: *Average iteration count per alternating descent step in the BSD application*

Methods	ADMM	Tseng	PD-DR	Chambolle-Pock
Runtime (ms)	139	12.5	60.3	13.0

Table 4.4: *Average runtime per iteration in the BSD application, averaging over the entire alternating descent algorithm.*

still the slowest among the four methods. This observation is similar to the randomly generated image experiment. The other three methods are much faster than ADMM by using less runtime per iteration. Tseng's forward-backward method is the overall best choice in terms of performance.

To summarize the experiments for SHDL application, the splitting methods applied to the primal-dual optimality condition are much faster, more accurate in terms of objective value than splitting methods applied to the dual optimality condition. The primal-dual based splitting methods recover more detail or high frequency dictionary components than ADMM does. It is extremely important to recover more details within a finite dictionary.

CHAPTER 5

Summary

Sparse matrix factorization and many other matrix factorization problems are usually solved by the alternating descent algorithm. We demonstrated this alternating descent algorithm applied to two applications of the sparse matrix factorization problem in machine learning: Structured sparsity principal component analysis and sparse hierarchical dictionary learning.

In this thesis we presented various splitting methods to solve the convex sub-problems of the alternating descent algorithm. We presented a linear transformation technique to reformulate the convex sub-problem with overlapping group norm so that the splitting methods can be applied. We also presented a novel singular value decomposition approach for one of the proximal operators used in the splitting methods. Our experimental results showed that most of the splitting methods performed much better than block coordinating descent methods in these two applications in terms of performance and stability. These splitting methods are capable of solving large-scale sparse matrix factorization problems with the proximal operators and resolvents for skew-symmetric linear operator presented in this thesis.

Most of the splitting methods discussed in this thesis are widely adaptable for other applications through the alternating descent algorithm. One example will be the sparse non-negative matrix factorization by adding non-negative constraints to both of the matrix factors. Besides the column-wise sparsity pattern, these splitting methods may also be applied to other type of sparsity penalties as long as the underlying proximal operators and resolvents are simple to calculate.

References

[BBL07] M. W. Berry, M. Browne, A. N. Langvilleb, V. P. Paucac, and R. J. Plemmonsc. "Algorithms and applications for approximate nonnegative matrix factorization." *Computational Statistics and Data Analysis*, **52**:155–173, 2007.

[Ber82] D. P. Bertsekas. "Constrained optimization and Lagrange multiplier methods." *Computer Science and Applied Mathematics, Boston: Academic Press*, 1982.

[Bis06] M Bishop. *Pattern Recognition and Machine Learning.* Springer, 2006.

[BMP08] F. Bach, J. Mairal, and J. Ponce. "Convex sparse matrix factorizations." *arXiv*, **CS**(LG):0812–1869, 2008.

[BPC11] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. "Distributed optimization and statistical learning via the alternating direction method of multipliers." *Journal Foundations and Trends in Machine Learning*, **3**(1):1–122, 2011.

[BT09] A. Beck and M. Teboulle. "A fast iterative shrinkage-thresholding algorithm for linear inverse problems." *SIAM Journal on Imaging Sciences*, **2**(1):183–202, 2009.

[BV04] S. Boyd and L. Vandenberghe. *Convex Optimization.* Cambridge University Press, 2004.

[CDS98] S. S. Chen, D. L. Donoho, and M. A. Saunders. "Atomic decomposition by basis pursuit." *SIAM Journal on Scientific Computing*, **20**(1):33–61, 1998.

[CP10] P. L. Combettes and J. C. Pesquet. "Proximal splitting methods in signal processing." *arXiv*, **0912.3522v4 [math.OC]**, 2010.

[CP11] A. Chambolle and T. Pock. "A first-order primal-dual algorithm for convex problems with applications to imaging." *Journal of Mathematical Imaging and Vision*, **40**(1):120–145, 2011.

[CT94] G. Chen and M. Teboulle. "Proximal-based decomposition method for convex minimization problems." *Mathematical Programming*, **64**(1-3):81–101, 1994.

[dBG08] A. d'Aspremont, F. Bach, and L Ghaoui. "Optimal solutions for sparse principal component analysis." *Journal of Machine Learning Research*, **9**:1269–1294, 2008.

[DLJ06] C. Ding, T. Li, and M. Jordan. "Convex and semi-nonnegative matrix factorizations." *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **32**(1):45–55, 2006.

[EA06] M. Elad and M. Aharon. "Image denoising via sparse and redundant representations over learned dictionaries." *IEEE Transactions on Image Processing*, **15**(12):3736–3745, 2006.

[EB92] J. Eckstein and D. Bertsekas. "On the Douglas-Rachford splitting method and the proximal algorithm for maximal monotone operators." *Mathematical Programming*, **55**(1-3):293–318, 1992.

[Eld07] L. Eldën. *Matrix Methods in Data Mining and Pattern Recognition.* Society for Industrial and Applied Mathematics, 2007.

[Fen49] W. Fenchel. "On the conjugate convex functions." *Canadian Journal of Mathematics*, **1**:73–77, 1949.

[FHT10] J. Friedman, T. Hastie, and R. Tibshirani. "A note on the group lasso and a sparse group lasso." *arXiv*, **1001.0736 [math.ST]**, 2010.

[GLA92] J. D. Gardiner, A. J. Laub, J. J. Amato, and C. B. Moler. "Solution of the Sylvester matrix equation $AXB^T + CXD^T = E$." *ACM Transactions on Mathematical Software (TOMS)*, **18**(2):223–231, 1992.

[G91] O. Gler. "On the convergence of the proximal point algorithm for convex minimization." *SIAM Journal on Control and Optimization*, **29**(2):403–419, 1991.

[JMO10] R. Jenatton, J. Mairal, G. Obozinski, and F. Bach. "Proximal Methods for Hierarchical Sparse Coding." *arXiv*, **arXiv:1009.2139 [stat.ML]**, 2010.

[JOB10] R. Jenatton, G. Obozinski, and F. Bach. "Structured sparse principal component analysis." *arXiv*, **0909.1440 [stat.ML]**, 2010.

[Jol02] I. T. Jolliffe. *Principal Component Analysis.* Springer, 2002.

[LBR07] H. Lee, A. Battle, R. Raina, and A. Ng. "Efficient sparse coding algorithms." *Advances in Neural Information Processing Systems*, **19**:801–808, 2007.

[LHZ01] S. Z. Li, X. Hou, H. Zhang, and Q. Cheng. "Learning spatially localized, parts-based representation." *Proceedings of the 2001 IEEE Computer Society Conference*, **1**:207–212, 2001.

[LS86] J. Lawrence and J. E. Spingarn. "On fixed points of non-expansive piecewise isometric mappings." *Proceedings London Mathematical Society*, **3**(3):605–624, 1986.

[MB98] A. M. Martinez and R. Benavente. "The AR face database." *CVC Tech.*, **Report 24**, 1998.

[MBP09] J Mairal, F. Bach, J. Ponce, and G. Sapiro. "Online learning for matrix factorization and sparse coding." *International Conference on Machine Learning*, **11**:19–60, 2009.

[MFT01] D. Martin, C. Fowlkes, D. Tal, and J. Malik. "A database of human segmented natural images and its application to evaluating segmentation algorithms and measuring ecological statistics." *Proc. 8th Int'l Conf. Computer Vision*, **2**:416–423, 2001.

[Nes07] Y. Nesterov. "Gradient methods for minimizing composite objective function." *CORE report*, 2007.

[OF97] B. A. Olshausen and D. J. Field. "Sparse coding with an overcomplete basis set: A strategy employed by V1?" *Vision Research*, **37**:3311–3325, 1997.

[Tib96] R Tibshirani. "Regression shrinkage and selection via the lasso." *Journal of the Royal Statistical Society. Series B (Methodological)*, **58**(1):267–288, 1996.

[Tse00] P. Tseng. "A modified forward-backward splitting method for maximal monotone mappings." *SIAM Journal on Control and Optimization*, **38**(2):431–446, 2000.

[Van12] L. Vandenberghe. "Seminar Lecture Series of Splitting Algorithms." *EE296 Seminar Lecture*, 2012.

[WTH09] D. Witten, R. Tibshirani, and T. Hastie. "A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis." *Biostatistics*, **10**(3):515–534, 2009.

[XLG03] W. Xu, X. Liu, and Y. Gong. "Document clustering based on non-negative matrix factorization." *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, **1**:267–273, 2003.

[ZHT06] H. Zou, T. Hastie, and R. Tibshirani. "Sparse principal component analysis." *Journal of Computational and Graphical Statistics*, **15**(2):265–286, 2006.