Project Proposal: AMSC/CMSC 663 and 664

The Alternating Direction Method of Multipliers
An Adaptive Step-size Software Library

Peter Sutor, Jr.
psutor@umd.edu

Project Advisor

Dr. Tom Goldstein
tomg@cs.umd.edu

Assistant Professor
Department of Computer Science
University of Maryland

October 17, 2015

Abstract

The Alternating Direction Method of Multipliers (ADMM) is a method that solves convex optimization problems of the form $\min(f(x) + g(z))$ subject to $Ax + Bz = c$, where A and B are suitable matrices and c is a vector, for optimal points (x_{opt}, z_{opt}). It is commonly used for distributed convex minimization on large scale data-sets. However, it can be technically difficult to implement and there is no known way to automatically choose an optimal step size for ADMM. Our goal in this project is to simplify the use of ADMM by making a robust, easy-to-use software library for all ADMM-related needs, with an adaptive step-size selection algorithm to optimize performance on every iteration. The library will contain a general ADMM method, as well as solvers for common problems that ADMM is used for. It will also implement adaptive step-size selection, utilize the Message Passing Interface (MPI) for parallel computing and have user-friendly options and features.

Introduction

The generalization of ADMM's usage is in solving convex optimization problems where the data can be arbitrarily large. That is, we wish to find $x_{opt} \in X$ such that:

$$f(x_{opt}) = \min \{f(x) : x \in X\}, \tag{1}$$

given some constraint $Ax = b$, where $X \subset \mathbb{R}^n$ is called the *feasible set*, $f(x) : \mathbb{R}^n \longmapsto \mathbb{R}$ is the *objective function*, X and f are convex, matrix $A \in \mathbb{R}^{m \times n}$ and vector $b \in \mathbb{R}^m$. Our input x here may have a huge amount of variables/dimensions, or an associated data matrix A for it can simply be hundreds of millions of entries long. In such extreme cases, the traditional techniques for minimization may be too slow, despite how fast they may be on normal sized problems.

Generally, such issues are solved by using parallel versions of algorithms to distribute the workload across multiple processors, thus speeding up the optimization. But our traditional optimization algorithms are not suitable for parallel computing, so we must use a method that is. Such a method would have to decentralize the optimization; one good way to do this is to use the *Alternating Direction Method of Multipliers* (**ADMM**). This convex optimization algorithm is robust and splits the problem into smaller pieces that can be optimized in parallel.

We will first give some background on ADMM, then describe how it works, with a brief example of how it is used to solve problems in practice. Next, we will discuss some ideas for adaptive step-size selection that we will try to implement and investigate. At the end, we discuss the design of the software library in greater detail, as well as well as how the project will proceed.

Background

In the following sections, we briefly describe the general optimization strategy ADMM uses, and the two algorithms ADMM is a hybrid of. For more information, refer to [1].

The Dual Problem

Consider the following equality-constrained convex optimization problem:

$$\min_x (f(x)) \text{ subject to } Ax = b \tag{2}$$

This is referred to as the *primal problem* (for a *primal function* f) and x is referred to as the *primal variable.* To help us solve this, we formulate a different problem using the Lagrangian and solve that. The *Lagrangian* is defined as

$$L(x, y) = f(x) + y^T(Ax - b). \tag{3}$$

We call the *dual function* $g(y) = \inf_x(L(x, y))$ and the *dual problem* $\max_y(g(y))$, where y is the *dual variable.* With this formulation, we can recover $x_{opt} = \arg\min_x(L(x, y_{opt}))$; f's minimizer.

One method that gives us this solution is the *Dual Ascent Method* (**DAM**), characterized at iteration k by computing until convergence:

1. $x^{(k+1)} := \arg\min_x(L(x, y^{(k)}))$ (minimization for $f(x)$ on x)
2. $y^{(k+1)} := y^{(k)} + \alpha^{(k)}(Ax^{(k+1)} - b)$ (update y for next iteration)

Here, $\alpha^{(k)}$ is a step size for the iteration k and we note that $\nabla g(y^{(k)}) = Ax_{opt} - b$, and $x_{opt} = \arg\min_x(L(x, y^{(k)}))$. If g is differentiable, this algorithm strictly converges and seeks out the gradient of g. If g is not differentiable, then we do not have monotone convergence and the algorithm seeks out the negative of a sub-gradient of $-g$. Note that the term $y^T(Ax - b)$ acts as a penalty function that guarantees minimization occurs on the given constraint.

Dual Decomposition

It's important to realize that for high-dimensional input we may want to parallelize DAM for better performance. The technique for this is described in this section. Suppose that our objective is *separable*; i.e. $f(x) = f_1(x_1) + \cdots + f_n(x_n)$, and $x = (x_1, ..., x_n)^T$. Then we can say the same for the Lagrangian. From (3), we have: $L(x, y) = L_1(x_1, y) + \cdots + L_n(x_n, y) - y^T b$, where $L_i = f(x_i) + y^T A_i x_i$. Thus, our x-minimization step in the DAM is split into n separate minimizations that can be carried out in parallel:

$$x_i^{(k+1)} := \arg\min_{x_i}\left(L_i(x_i, y^{(k)})\right).$$

This leads to a good plan for parallelization: disperse $y^{(k)}$, update x_i in parallel then add up the $A_i x_i^{(k+1)}$. This is called *Dual Decomposition* (**DD**), and was originally proposed by Dantzig and Wolfe [4, 2], and by Benders [3]. However, Dual Decomposition's general idea is primarily due to Everett [5]. The algorithm computes the above x-minimization step for $i = 1, ..., n$, in parallel, then coordinates to update the *dual variable*:

$$y^{(k+1)} := y^{(k)} + \alpha^{(k)}\left(\sum_{i=1}^{n} A_i x_i^{(k+1)} - b\right).$$

Initially, this seems great. But this algorithm requires several big assumptions (sufficiently smooth and decomposible f), and can be slow at times. We need a faster method.

Method of Multipliers

What if we want to make DAM more robust, with faster iterations and convergence? The *Method of Multipliers* (**MM**) can do this, as proposed by Hestenes [7, 8] and Powell [9]. Simply swap the Lagrangian for an *Augmented Lagrangian*:

$$L_\rho(x, y) = f(x) + y^T(Ax - b) + (\rho/2)||Ax - b||_2^2 \text{ , where } \rho > 0. \tag{4}$$

Note the addition of another penalty term that penalizes straying too far from the constraint during minimization over the length of ρ. Now our iteration computes until convergence:

1. $x^{(k+1)} := \arg\min_x (L_\rho(x, y^{(k)}))$ (minimization Lagrangian for x)
2. $y^{(k+1)} := y^{(k)} + \rho(Ax^{(k+1)} - b)$ (update y for next iteraton)

Here, ρ is the *dual update step length*, chosen to be the same as the penalty coefficient ρ in (4). This Augmented Lagrangian can be shown to be differentiable under mild conditions for the primal problem. According to [1], for a differentiable f, the *optimality conditions* are:

$$\left| \begin{array}{ll} \text{Primal Feasibility:} & Ax_{opt} - b = 0 \\ \text{Dual Feasibility:} & \nabla f(x_{opt}) + A^T y_{opt} = 0 \end{array} \right.$$

At each iteration k, $x^{(k+1)}$ minimizes $L_\rho(x, y^{(k)})$, so:

$$\begin{aligned} \nabla_x L_\rho(x^{(k+1)}, y^{(k)}) &= \nabla_x f(x^{(k+1)}) + A^T(y^{(k)} + \rho(Ax^{(k+1)} - b)) \\ &= \nabla_x f(x^{(k+1)}) + A^T y^{(k+1)} = 0 \end{aligned}$$

Thus, our dual update $y^{(k+1)}$ makes $(x^{(k+1)}, y^{(k+1)})$ *dual feasible*; the *primal feasibility* is achieved as $(Ax^{(k+1)} - b) \to 0$ (convergence on constrained solution).

What does all this mean?

Generally, MM is faster, more robust (does not require a smooth f) and has more relaxed convergence conditions than DD. However, MM's quadratic penalty in the Augmented Lagrangian prevents us from being able to parallelize the x-update like in DD. With this set-up, we cannot have the advantages of both MM and DD. This is where ADMM comes into play.

The Alternating Method Of Multipliers (ADMM)

Having covered the background of ADMM, we can now begin discussing the algorithm itself.

The General ADMM Algorithm

ADMM combines the advantages of DD and MM. It solves problems of the form:

$$\min\,(f(x) + g(z)) \text{ subject to } Ax + Bz = c, \tag{5}$$

where f and g are both convex. Note that the objective is separable into two sets of variables. ADMM defines and uses a special Augmented Lagrangian to allow for decomposition:

$$L_\rho(x, z, y) = f(x) + g(x) + y^T(Ax + Bz - c) + \frac{\rho}{2}||Ax + Bz - c||_2^2 \tag{6}$$

The ρ in (6) is the step length. The original ADMM algorithm was proposed by Gabay and Mercier [11], and Glowinski, and Marrocco [10]. Many further findings in ADMM were discovered by Eckstein and Bertsekas [12]. At iteration k, we minimize for x, then z, and finally, update y, keeping the other variables constant during each minimization. This gives us the ADMM algorithm shown in Algorithm 1.

Algorithm 1 The Alternating Direction Method of Multipliers (ADMM)

```
1: procedure ADMM($A$, $B$, $c$, $\rho$, OBJECTIVE, ARGMINX, ARGMINZ, STOPCOND)
2:     Set $x$, $z$ and $y$ to some initial value.
3:     while STOPCOND($x$, $z$, $y$, $A$, $B$, $c$) $\neq 1$ do
4:         $x :=$ ARGMINX$(x, z, y, \rho)$                    (minimize $f(x)$ for $x$)
5:         $z :=$ ARGMINZ$(x, z, y, \rho)$                    (minimize $g(z)$ for $z$)
6:         $y := y + \rho(Ax + Bz - c)$                     (Dual variable update)
7:     return ($x$, $z$, OBJECTIVE($x$, $z$))
```

There are a few things to note about this formulation of ADMM. The way the algorithm works does not require explicitly knowing the objective function $f(x) + g(z)$; it only requires the constraint variables (A, B, and c), minimizing functions ARGMINX and ARGMINZ, and a stopping condition function STOPCOND. The algorithm only cares about the objective function OBJECTIVE for evaluating the final, minimized value. This can even be left up to the user. Another detail to note is that the algorithm decouples the objective function on variables x and z and minimize on them independently. This formulation changes our optimality conditions a little.

ADMM's Optimality Conditions

Assume f and g are differentiable. We now have a second dual feasible condition due to the z-minimization step in ADMM that did not exist in MM. The other conditions are slightly altered from MM for the ADMM constraint:

$$\begin{array}{|ll} \text{Primal Feasibility:} & Ax + Bz - c = 0 \\ \text{Dual Feasibility:} & \nabla f(x) + A^T y = 0, \\ & \nabla g(z) + B^T y = 0 \end{array}$$

Assume $z^{(k+1)}$ minimizes $L_\rho(x^{(k+1)}, z, y^{(k)})$; we want to show that the dual update makes $(x^{(k+1)}, z^{(k+1)}, y^{(k+1)})$ satisfy the dual feasible condition for g. We proceed using the same strategy as was done in MM, using ADMM's Augmented Lagrangian instead:

$$\begin{aligned} 0 &= \nabla g(z^{(k+1)}) + B^T y^{(k)} + \rho B^T (Ax^{(k+1)} + Bz^{(k+1)} - c) \\ &= \nabla g(z^{(k+1)}) + B^T y^{(k+1)} \end{aligned}$$

Thus, the dual update makes $(x^{(k+1)}, z^{(k+1)}, y^{(k+1)})$ satisfy the second dual feasible condition. As for the other dual and primal conditions, they are both achieved as $k \to \infty$.

Convergence of ADMM

What conditions need to be satisfied in order for ADMM to be guaranteed to converge? According to [1, 12], ADMM requires two assumptions:

1. The infinite domain of functions f and g must be closed, proper and convex. In Euclidean space, a set is *closed* if its complement is an *open set*, a generalization of an open interval on the reals in higher dimensions. A set is *convex* in Euclidean space if for every pair of points in the set, all points on the line segment between them lie in the set as well. A convex function f's domain is *proper* if its effective domain (all x such that $f(x) < \infty$) is nonempty and never reaches $-\infty$.

2. The Augmented Lagrangian in (6) has a saddle point for $\rho = 0$.

Assumption 1 essentially states that the subproblems solved by ADMM in the minimization steps for f and g must indeed be solvable, even if not uniquely. This condition allows for f and g to be non-differentiable and take on the value of positive infinity. It also guarantees that for assumption 2, the saddle point is finite.

Thus, the conditions are not to strict for convergence. These two assumptions guarantee that residuals between iterates converge to 0, the objective approaches the optimal value, and that the dual value also approaches the optimal value.

Scaled Dual Form of ADMM

By scaling the dual variable y in ADMM by $1/\rho$, we can rewrite the algorithm in a simpler way. Let *residual* $r = Ax + Bz - c$, then:

$$\begin{aligned} L_\rho(x,z,y) &= f(x) + g(z) + y^T r + (\rho/2)||r||_2^2 \\ &= f(x) + g(z) + (\rho/2)||r + (1/\rho)y||_2^2 - (1/2\rho)||y||_2^2 \\ &= f(x) + g(z) + (\rho/2)||r+u||_2^2 - constant_y \\ &= L_\rho(x,z,u), \end{aligned} \tag{7}$$

where $u = (1/\rho)y$. Now the ADMM can be written in a simpler fashion, as shown in Algorithm 2.

Algorithm 2 Scaled Dual ADMM

```
1: procedure SCALEDADMM($A$, $B$, $c$, $\rho$, OBJECTIVE, ARGMINX, ARGMINZ, STOPCOND)
2:     Set $x$, $z$ and $u$ to some initial value.
3:     while STOPCOND($x$, $z$, $u$, $A$, $B$, $c$) $\neq 1$ do
4:         $x :=$ ARGMINX($x, z, u, \rho$)                    (minimize $f(x)$ for $x$)
5:         $z :=$ ARGMINZ($x, z, u, \rho$)                    (minimize $g(z)$ for $z$)
6:         $u := u + (Ax + Bz - c)$                       (Dual variable update)
7:     return ($x$, $z$, OBJECTIVE($x$, $z$))
```

Writing General Convex Problems in ADMM Form

What if we want to use ADMM for a general convex optimization problem; that is, the generic problem: $\min f(x)$, subject to $x \in \mathbb{S}$, with f and $\mathbb{S}$ convex? One way is to simply write: $\min (f(x) + g(z))$, subject to $x - z = 0$, hence $x = z$. The question is, what do we make g? A good idea is to let $g(z) = \mathbb{I}_{\mathbb{S}}(z)$, the indicator function that z is in the set $\mathbb{S}$ (i.e., $g(z) = 1$ if $x \in \mathbb{S}$, else $g(z) = 0$), as it does not impact the problem we are trying to solve.

Notice that in this formulation, $B = -\mathrm{I}$, so z-minimization boils down to

$$\arg\min\,(g(z) + (\rho/2)||z - v||_2^2) = \mathbf{prox}_{g,\rho}(v), \tag{8}$$

with $v = x^{(k+1)} + u^{(k)}$, where $\mathbf{prox}_g(v)$ is the *proximal operator* of v on function g. The proximal operator is defined as

$$\mathbf{prox}_g(v) = \arg\min_z\,(g(z) + \frac{1}{2}||z - v||_2^2) \tag{9}$$

We add an additional parameter ρ to our version of the proximal operator for the step size. Since we have the special case that $g(z)$ is the indicator function, we can compute the proximal operator by projecting v onto $\mathbb{S}$. However, since the indicator function itself is difficult to minimize for, we instead actually use an alternative $g(z) = \lambda||z||_1$, a proximal mapping of the l_1 norm, an equivalent substitute as far as minimization goes. Then, we can use a technique called *soft-thresholding*. This gives the individual components of z in the minimization by computing:

$$z_i^{(k+1)} := (v_i - \lambda/\rho)_+ - (-v_i - \lambda/\rho)_+ \tag{10}$$

We can do this over all the components at once for our z-minimization step. Notice that both the x and z-minimization steps in ADMM are proximal operators, by definition of the Augmented Lagrangian. Thus, in a general ADMM program, we can ask for functions that compute the proximal operators for both as input functions ARGMINX and ARGMINZ for Algorithms 1 and 2.

This formulation allows us to solve problems of the form in (1). If there is already a constraint like the one in (2), we can still use this formulation to solve it via ADMM. However, the constraint $Ax = b$ cannot simply be ignored; the parameters A and b will be incorporated into the x update step; i.e., they are part of the minimization problem for x (the solutions for which are supplied by the user as a function in generalized ADMM). How to handle the x update is dependent on the problem, though there are solutions for general cases such as in Quadratic Programming.

What about inequality constrained problems, such as $Ax \leq b$? There are some tricks one can do to solve certain inequality constrained convex optimization problems. Using a slack variable z, we can write the problem as $Ax + z = b$, with $z \geq 0$. This is now in ADMM form, but with the additional constraint that $z \geq 0$. The additional constraint primarily affects the z update step in this case, as we need to ensure a non-negative z is chosen. Considering $g(z) = \lambda||z||_1$, which can be solved in general via (10), we can modify the solution to select positive values for z. For example, we can project (10) into the non-negative orthant by setting negative components to zero, for $v = Ax^{(k+1)} + u^{(k)}$.

Stopping Conditions

By [1], we can define the primal (p) and dual (d) residuals in ADMM at step $k+1$ as:

$$p^{k+1} = Ax^{k+1} + Bz^{k+1} - c \tag{11}$$
$$d^{k+1} = \rho A^T B(z^{k+1} - z^k) \tag{12}$$

The primal residual is trivial. However the dual residual stems from the need to satisfy the first dual optimality condition $\nabla f(x) + A^T y = 0$. More generally, for subgradients ∂f and ∂g for f and g, and since x^{k+1} minimizes $L_\rho(x, z^k, y^k)$:

$$\begin{aligned}
0 &\in \partial f(x^{k+1}) + A^T y^k + \rho A^T(Ax^{k+1} + Bz^k - c) \\
&= \partial f(x^{k+1}) + A^T y^k + \rho A^T(r^{k+1} + Bz^k - Bz^{k+1}) \\
&= \partial f(x^{k+1}) + A^T y^k + \rho A^T r^{k+1} + \rho A^T B(z^k - z^{k+1}) \\
&= \partial f(x^{k+1}) + A^T y^k + \rho A^T B(z^k - z^{k+1})
\end{aligned}$$

So, one can say $d^{k+1} = \rho A^T B(z^{k+1} - z^k) \in \partial f(x^{k+1}) + A^T y^k$, and the first dual optimality condition is satisfied by (12). It is reasonable to say that the stopping criteria is based on some sort of primal and dual tolerances that change with the iteration. I.e., $||p^k||_2 \leq \epsilon^{pri}$ and $||d^k||_2 \leq \epsilon^{dual}$. There are many ways to choose these tolerances. One common example, where $p \in \mathbb{R}^{n_1}$ and $d \in \mathbb{R}^{n_2}$:

$$\epsilon^{pri} = \sqrt{n_1}\epsilon^{abs} + \epsilon^{rel} \max(||Ax^k||_2, ||Bz^k||_2, ||c||_2) \tag{13}$$

$$\epsilon^{dual} = \sqrt{n_2}\epsilon^{abs} + \epsilon^{rel}||A^T y^k||_2 \tag{14}$$

where ϵ^{abs} and ϵ^{rel} are chosen constants referred to as *absolute* and *relative* tolerance. In practice, the absolute tolerance specifies the precision of the result, while the relative tolerance specifies the accuracy of the dual problem in relation to the primal.

Parallelizing ADMM

The advantage of ADMM over the methods discussed in the background section is that it has the desired robustness and speed, but doesn't sacrifice the ability to parallelize. But how exactly could a distributed ADMM work?

We can let $A = \mathrm{I}$, $B = -\mathrm{I}$ and $c = 0$ to set the constraint as $x = z$. As a result, with a separable f and x, we can minimize f_i and require each $x_i = z$ at the end. Thus, we optimize each x_i and aggregate their solutions to update our z, so our Augmented Lagrangian looks like:

$$L_\rho(x, z, y) = \sum_i (f_i(x_i) + y^T(x_i - z) + \frac{\rho}{2}||x_i - z||_2^2) \tag{15}$$

where each x_i is a decomposed vector from the original x.

In general, ADMM actually solves a single convex function, which is decomposed into $f(x)+g(z)$. The function $g(z)$ is ideally chosen as something artificial and easy to solve, like the proximal mapping of the l_1 norm with the solution in (10). Thus, the above distributed scheme works for many uses of ADMM. More complicated schemes can be developed should the need arise. See Algorithm 3 for a general distributed ADMM algorithm. Parallelization with ADMM, along with other similar MM methods is discussed in further detail in [2] and [6].

Algorithm 3 Distributed ADMM (Scaled and Unscaled Dual)

```
1: procedure DistributedADMM($A$, $B$, $c$, $\rho$, OBJECTIVE, ARGMINX, ARGMINZ, STOPCOND)
2:     Set $x$, $z$ and $u$ to some initial value.
3:     while STOPCOND($x$, $z$, $u$, $A$, $B$, $c$) $\neq 1$ do
4:         for parallel machine labeled by index $i$ do
5:             $x_i :=$ ARGMINX$(f_i(x) + (y_i)^T(x - z) + \frac{\rho}{2}||x - z||_2^2)$        (distributed $x$ update)
6:         $z :=$ ARGMINZ$((y_i)^T(x_i - z) + \frac{\rho}{2}||x_i - z||_2^2) = \frac{1}{n}\sum_{i=1}^{n}(x_i + \frac{1}{\rho}y_i) = \frac{1}{n}\sum_{i=1}^{n}(x_i + u_i)$
7:         for component index $i$ do
8:             $y_i := y_i + \rho(x_i - z) = \frac{1}{\rho}y_i + x_i - z = u_i + x_i - z$
9:     return ($x$, $z$, OBJECTIVE($x$, $z$))
```

Solving Problems With ADMM

ADMM is capable of solving many general convex optimization problems. To name a few, according to [1]:

- Basis Pursuit
- Sparse Inverse Covariance Selection
- Huber Fitting
- Intersection of Polyhedra
- Lasso Problem
- Least Absolute Deviations
- Linear Programming
- ℓ_1 Regularized Logistic Regression
- Regressor Selection (nonconvex)
- Quadratic Programming
- Support Vector Machines (SVMs)
- Total Variation Minimization (e.g., image denoising)

One can read about the problem statements for all of these optimization problems, as well as ADMM solutions for them in [1]. For purposes of demonstrating how ADMM can solve these problems and to show the need for adaptive step-size selection, we will go over how to solve Total Variation Minimization in one dimension with ADMM.

Total Variation Minimization in 1D

Total Variation (**TV**) measures the integral of the absolute value of the gradient of a signal x. Minimizing this integral eliminates small variations in the signal data but preserves large variations, thus removing noise from a noisy signal. Because of this, it has many applications. One example is image denoising in the 2-dimensional case.

In 1-dimension, TV is defined as $V(x) = \sum_i |x_{i+1} - x_i|$. We want to find a value close to a given signal component b_i that has smaller Total Variation. One way to measure such closeness is the sum of square errors: $E(x,b) = \frac{1}{2}||x-b||_2^2$. So our goal is to minimize for x, over a signal b:

$$E(x,b) + \lambda V(x) \tag{16}$$

The λ here is called the *regularization parameter*, and it dictates how much of an impact the Total Variation has in the minimization by scaling it by λ. The problem, in ADMM form, can be written as:

$$\arg\min_x \left(E(x,b) + \lambda||z||_1\right), \text{ subject to } Dx - z = 0 \tag{17}$$

First, let's figure our the x-minimization step. We see that the gradient can be computed by hand. So, using ADMM's Augmented Lagrangian we minimize x by:

$$\begin{aligned} \nabla_x L_\rho(x, z^{(k)}, u^{(k)}) &= \nabla_x E(x,b) + \nabla_x ||Dx - z^{(k)} + u^{(k)}||_2^2 \\ &= \nabla_x \frac{1}{2}||x-b||_2^2 + \rho D^T(Dx - z^{(k)} + u^{(k)}) \\ &= x - b + \rho D^T Dx - \rho D^T z^{(k)} + \rho D^T u^{(k)} = 0 \end{aligned} \tag{18}$$

We now solve (18) for x:

$$\begin{aligned} x - b + \rho D^T Dx - \rho D^T z^{(k)} + \rho D^T u^{(k)} = 0 \\ x + \rho D^T Dx &= \rho D^T z^{(k)} - \rho D^T u^{(k)} + b \\ (\mathrm{I} + \rho D^T D)x &= \rho D^T (z^{(k)} - u^{(k)}) + b \\ x &= (\mathrm{I} + \rho D^T D)^{-1}(\rho D^T(z^{(k)} - u^{(k)}) + b) \end{aligned} \tag{19}$$

Thus, based off of the results in (19), our x-minimization step is:

$$x^{(k+1)} := (\mathrm{I} + \rho D^T D)^{-1}(\rho D^T(z^{(k)} - u^{(k)}) + b) \tag{20}$$

In real life, we would not form the inverse matrix in (20), but solve a system for x^{k+1}. The z and y minimization steps are as before for the general ADMM form, like in Algorithm 2. Thus, the algorithm for solving TV remains the same as Algorithm 2 but with (20) as the x-minimization step and soft-thresholding for the z step. This is shown in Algorithm 4

Once implemented, the algorithm would return results similar to those shown in Figure 1. As we can see in this figure, both the values of ρ and λ alter the number of iterations required to

Algorithm 4 TV Minimization

```
1: procedure TVM($b$, $\rho$, $\lambda$, STOPCOND)
2:     Set $x$, $z$ and $u$ to some initial value.
3:     Form matrix $D$ using stencil $[1 - 1]$ (circular boundaries)
4:     while STOPCOND($x$, $z$, $u$, $D$ $b$) $\neq 1$ do
5:         $x := (\mathrm{I} + \rho D^T D)^{-1}(\rho D^T(z - u) + b)$
6:         $z := (v - \lambda/\rho)_+ - (-v - \lambda/\rho)_+$, with $v = -Dx - (u = y^{(k)}/\rho)$
7:         $u := u + Dx - z$
8:     return ($x$, $z$, Evaluate (16) for ($x$, $z$))
```

converge. This exemplifies why it is so difficult to choose an optimal step-size ρ for a general ADMM algorithm. The problem being minimized itself often has an effect on convergence, and choosing an optimal ρ might be dependent on parameters inherent to the problem itself.

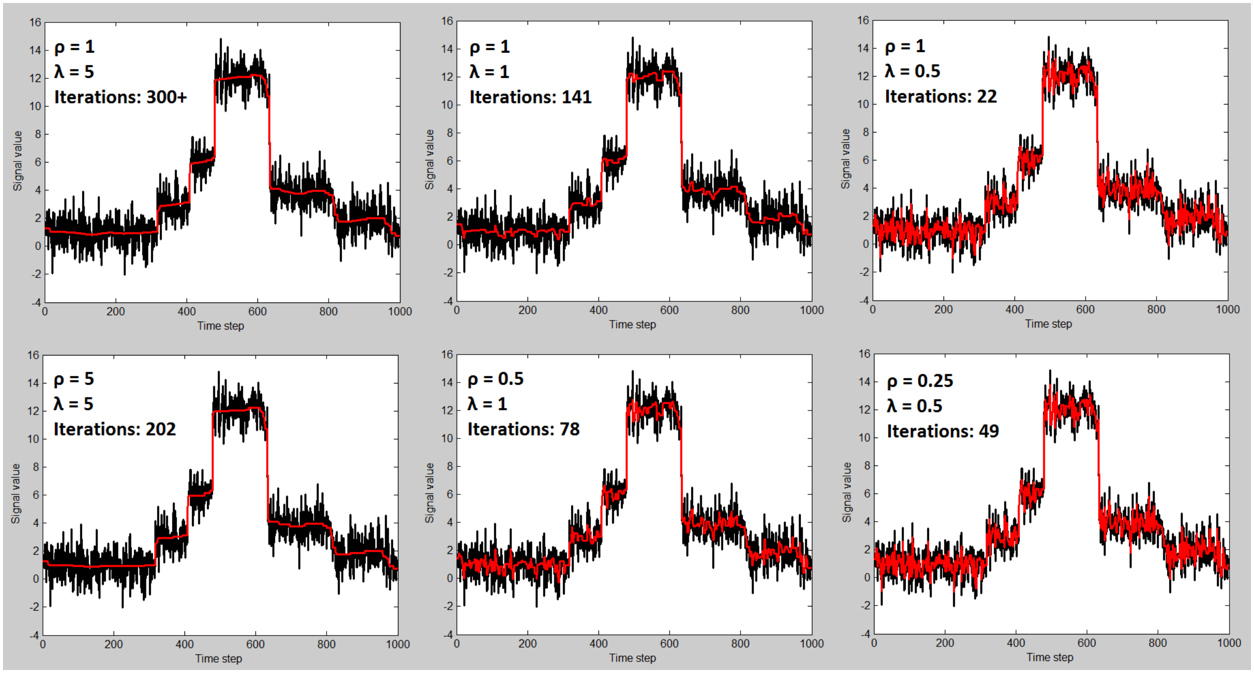

Figure 1: Results for a MATLAB implementation of TV Minimization using ADMM for various values of step-size ρ and λ.

Drawbacks Of ADMM

There are inherent issues with ADMM that can cause problems when it is used. One notable issue is the convergence of ADMM itself. It converges very fast initially, but tends to slow down as it approaches the solution. Thus, it takes a lot of work to get an answer down to machine precision. Generally, ADMM is used to compute the solution to within five or six decimal digits to avoid this issue. As a result, ADMM is often used when we do not require exact precision in our solution.

Another issue is that it is generally not as fast as traditional convex optimization methods.

With careful choices of step size ρ, however, it can be competitive with state-of-the-art techniques. The issue then lies in how to choose the step size. This may require specialized knowledge about each problem being solved - can we automate the optimal choice of ρ? Or better yet, can we automate the optimal choice of ρ at each iteration to converge at optimum speeds? Doing this can make ADMM run in a fraction of the time.

Adaptive Step Size ADMM

There are a few ways ADMM could potentially choose a close-to-optimal step size at each iteration. Solving such a problem is much easier than choosing an overall step size for the entire execution of ADMM. In practice, repeated experiments are performed to determine a good step size ρ for ADMM for a given problem. This process is not very convenient nor efficient. We cover a few ideas for Adaptive ADMM in the next few sections that we would like to try.

Interpolation Between Iterates

Suppose ADMM has already iterated at least two times. Since we started at an arbitrary point, this gives us at least three previous points to work with. From these three points we can form residuals between the last three iterations: call them r^k and r^{k+1}, respectively. We can interpolate for a residual r that should have been between these two residuals:

$$\frac{r - r^k}{\rho} = \frac{r^{k+1} - r^k}{\rho_k} \iff r = r^k + (r^{k+1} - r^k)\frac{\rho}{\rho_k} \tag{21}$$

In (21), the only unknown is ρ, the time step for the interpolated iteration. We can plug this r back into any other relationship between the supposed r residual and r^k and/or r^{k+1}. By doing this, it is possible to write an expression to minimize in terms of ρ. We can now choose the ρ that will minimizing the objective as much possible. This ρ will then be used in the next full iteration as a better step length than the previous.

Determining ρ will likely require approximating our resulting model; for example, to a linear model, and solving that. In any case, the approximation would have to be good enough while still being fast to compute, in order for ρ to be of any use.

Computing Dual Step's Gradient

ADMM can be viewed as a type of Douglas-Rachford Method (**DRM**), which is a further generalization of a class of proximal methods. Using results from [13]'s paper on DRM, one can solve for the optimal step size via computing the gradient on the dual step in DRM. Once the gradient is found, [13] provides a set of equations relating DRM's Lagrange Multiplier to ADMM's which can be solved for the step length.

Optimize Dual Problem's Step Size

It is possible to optimize the dual problem's step size and leave the primal problem's step size alone. This translates into keeping the penalty term's ρ constant and manipulate the dual's step size ρ only. This would greatly simplify solving for a better step length, but may not be effective enough to speed up convergence significantly. Furthermore, changing the dual step length requires checking the additional optimality condition $\nabla(g) + B^T y = 0$.

Project Details

Here we discuss the specifics of the software library that will implement Adaptive ADMM in an easy-to-use way, and how the general project will proceed throughout the year.

Implementation

The goal is to reduce the user's job to the minimum of uploading matrices/matrix functions A, B, and vector c for the constraints, and providing the proximal functions for f and g, if using general ADMM. Initialization of vectors x, z, and u can simply be to let them all be the zero vector, or some other given vector if the user has a better one in mind (in some problems, one can guess good initial vectors).

The software library should be capable of adaptively selecting the step-size ρ or use a given one, have distributed computing functionality built-in via the Message Passing Interface (MPI) using the strategy described in Algorithm 3, and have at least the flexible customization of the FASTA scheme in [14]. There will be optimized helper functions that can solve the most common problems ADMM is used for (listed on pages 6-7). We plan to have a MATLAB and Python implementation so that the library is more accessible to users.

The software library will utilize all the tricks described in the section on ADMM. The primary structure of the algorithm will resemble that of the scaled dual version in Algorithm 2, and will assume proximal operators as given input for solving the objective. Giving an actual objective function as input will be optional. Stopping conditions will at least be as robust as described earlier, but may need to be more varied for special circumstances. From the section on minimizing Total Variation, one can see how solvers for such problems can be developed with ADMM. Implementation details are extensively described in [1] for all common problems solved by ADMM.

Implementation Challenges

The most challenging aspect of this project will be finding an adaptive step size ADMM method. This will likely require much experimentation and will likely take up a significant portion of time. Barring that, another challenge is implementing stopping conditions. As described earlier, there are general stopping conditions, but there may be others to consider for special circumstances. One other challenge will be input checking. For example, how does one know if the given proximal functions for ADMM actually converge or not? Checking for this may not be an easy thing to do, but should be possible from observing how the residuals behave.

Testing

If one or more of the adaptive step-size strategies work, they must be tested in comparison to each other to see how they perform. This can be done by simply running each adaptive method on the same input, for a set of inputs. Furthermore, the convergence speeds can be measured this way directly for iteration counts and overall run-time, averages over multiple trials.

In addition to this, each working adaptive method must be tested in performance to non-adaptive ADMM. The software library will have options for turning on/off adaptive ADMM, thus testing can be done on the same general ADMM function.

The test inputs will be randomly generated, to get more general results on ADMM's performance. This way, the testing described should show whether an adaptive ADMM method is better than the others, and whether it outperforms non-adaptive ADMM, and if so, by how much.

Validation

As in the testing section, the correctness of ADMM can be tested through randomized data. This is the standard for convex optimization, as there is no real model problem to work with. We can compare how correct ADMM is by comparing it to other, working convex solvers and seeing if it gets similar results. For distributed computation, we can compare the run time for random constrained, decomposable f between the distributed and sequential options. The distributed option should give the expected run time improvements, for large enough vectors and matrices in the constraints.

For the ADMM solvers for common problems, the validation stage can use real data in some instances, such as with SVMs (the MNIST handwritten database will be used for that). In such cases, as with MNIST, we can compare how ADMM performs to existing solvers. For two dimensional TV Minimization one could use any set of images with Gaussian noise added to them as a benchmark for validation. Otherwise, we proceed the same way as with general ADMM for the validation of the solvers.

Project Schedule and Milestones

Below is the envisioned schedule for the project. The bold and italicized entries are milestones that can objectively measure the progress of the project.

- **Fall Semester Goals:**
  - **End of October:** ***Implement generic ADMM, solvers for the Lasso problem, TV Minimization, and SVMs.***
  - **Early November:** Implement scripts for general testing, convergence checking, and stopping condition strategies.
  - **End of November:** Try out implementations of all three adaptive step-size selection strategies.
  - **Early December:** Finalize bells and whistles on ADMM options. ***Compile testing and validation data.***
- **Spring Semester Goals:**
  - **End of February:** ***Implement the full library of standard problem solvers.***
  - **End of March:** ***Finish implementing MPI in ADMM library.***
  - **End of April:** Finishing porting code to Python version.
  - **Early May:** ***Compile new testing/validation data.***

Deliverables

Below is a list of expected deliverables at the end of the project:

- **ADMM Library: Python and Matlab versions**
  - Contain general ADMM with adaptive step size routines and standard solvers for common problems ADMM solves.
  - Scripts for generating random test data and results.
  - Scripts for validating performance of adaptive ADMM to regular ADMM for each adaptive strategy.
- Report on observed testing/validation results and on findings with adaptive ADMM - may lead to a paper eventually.
- Datasets used for testing the standard solvers (or references to where to obtain them, if they are too big).

References

[1] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers", *Foundations and Trends in Machine Learning*, vol. 3, no.1, pp. 1-122, 2010.

[2] G. B. Dantzig and P. Wolfe, "Decomposition principle for linear programs", *Operations Research*, vol. 8, pp. 101111, 1960.

[3] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems", *Numerische Mathematik*, vol. 4, pp. 238252, 1962.

[4] G. B. Dantzig, *Linear Programming and Extensions.* RAND Corporation, 1963.

[5] H. Everett, "Generalized Lagrange multiplier method for solving problems of optimum allocation of resources", *Operations Research*, vol. 11, no. 3, pp. 399417, 1963.

[6] A. Nedic and A. Ozdaglar, "Cooperative distributed multi-agent optimization", in *Convex Optimization in Signal Processing and Communications*, (D. P. Palomar and Y. C. Eldar, eds.), Cambridge University Press, 2010.

[7] M. R. Hestenes, "Multiplier and gradient methods", *Journal of Optimization Theory and Applications*, vol. 4, pp. 302320, 1969.

[8] M. R. Hestenes, "Multiplier and gradient methods", in *Computing Methods in Optimization Problems*, (L. A. Zadeh, L. W. Neustadt, and A. V. Balakrishnan, eds.), Academic Press, 1969.

[9] M. J. D. Powell, "A method for nonlinear constraints in minimization problems", in *Optimization*, (R. Fletcher, ed.), Academic Press, 1969.

[10] R. Glowinski and A. Marrocco, "Sur l'approximation, par elements finis d'ordre un, et la resolution, par penalisation-dualit'e, d'une classe de problems de Dirichlet non lineares", *Revue Francaise d'Automatique, Informatique, et Recherche Operationelle*, vol. 9, pp. 4176, 1975.

[11] D. Gabay and B. Mercier, "A dual algorithm for the solution of nonlinear variational problems via finite element approximations", *Computers and Mathematics with Applications*, vol. 2, pp. 1740, 1976.

[12] J. Eckstein and D. P. Bertsekas, "On the Douglas-Rachford splitting method and the proximal point algorithm for maximal monotone operators", *Mathematical Programming*, vol. 55, pp. 293318, 1992.

[13] E. Esser, *Applications of Lagrangian-Based Alternating Direction Methods and Connections to Split Bregman*, April 2009.

[14] T. Goldstein, C. Studer and R. G. Baraniuk, "A Field Guide to Forward-Backward Splitting with a FASTA Implementation", *CoRR*, arXiv:1411.3406, Nov. 2014.