Parameter Selection and Pre-Conditioning for a Graph Form Solver

Christopher Fougner Stephen Boyd

Original version March 2015. Updated version Feb 2017.

Abstract

In a recent paper, Parikh and Boyd describe a method for solving a convex optimization problem, where each iteration involves evaluating a proximal operator and projection onto a subspace. In this paper we address the critical practical issues of how to select the proximal parameter in each iteration, and how to scale the original problem variables, so as the achieve reliable practical performance. The resulting method has been implemented as an open-source software package called POGS (Proximal Graph Solver), that targets multi-core and GPU-based systems, and has been tested on a wide variety of practical problems. Numerical results show that POGS can solve very large problems (with, say, more than a billion coefficients in the data), to modest accuracy in a few tens of seconds. As just one example, a radiation treatment planning problem with around 100 million coefficients in the data can be solved in a few seconds, as compared to around one hour with an interior-point method.

1 Introduction

We consider the convex optimization problem

$$\begin{array}{ll} \text{minimize} & f(y) + g(x) \\ \text{subject to} & y = Ax, \end{array} \tag{1}$$

where $x \in \mathbf{R}^n$ and $y \in \mathbf{R}^m$ are the variables, and the (extended-real-valued) functions $f : \mathbf{R}^m \to \mathbf{R} \cup \{\infty\}$ and $g : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ are convex, closed and proper. The matrix $A \in \mathbf{R}^{m \times n}$, and the functions f and g are the problem data. Infinite values of f and g allow us to encode convex constraints on x and y, since any feasible point (x, y) must satisfy

$$x \in \{x \mid g(x) < \infty\}, \qquad y \in \{y \mid f(y) < \infty\}.$$

We will be interested in the case when f and g have simple proximal operators, but for now we do not make this assumption. The problem form (1) is known as *graph form* [PB13a],

since the variable (x, y) is constrained to lie in the graph $\mathcal{G} = \{(x, y) \in \mathbf{R}^{n+m} \mid y = Ax\}$ of A. We denote $p^\star$ as the optimal value of (1), which we assume is finite.

The graph form includes a large range of convex problems, including linear and quadratic programming, general conic programming [BV04, §11.6], and many more specific applications such as logistic regression with various regularizers, support vector machine fitting [HTF09], portfolio optimization [BV04, §4.4.1] [GM75] [BMOW13], signal recovery [CW05], and radiation treatment planning [OW06], to name just a few.

In [PB13a], Parikh and Boyd described an operator splitting method for solving (a generalization of) the graph form problem (1), based on the alternating direction method of multipliers (ADMM) [BPC+11]. Each iteration of this method requires a projection (either exactly or approximately via an iterative method) onto the graph $\mathcal{G}$, and evaluation of the proximal operators of f and g. Theoretical convergence was established in those papers, and basic implementations demonstrated. However it has been observed that practical convergence of the algorithm depends very much on the choice of algorithm parameters (such as the proximal parameter ρ), and scaling of the variables (*i.e.*, pre-conditioning).

The purpose of this paper is to explore these issues, and to add some critical variations on the algorithm that make it a relatively robust general purpose solver, at least for modest accuracy levels. The algorithm we propose, which is the same as the basic method described in [PB13a], with modified parameter selection, diagonal pre-conditioning, and modified stopping criterion, has been implemented in an open-source software project called POGS (for **Pro**ximal **G**raph **S**olver), and tested on a wide variety of problems. Our CUDA implementation reliably solves (to modest accuracy) problems $10^3\times$ larger than those that can be handled by interior-point methods; and for those that can be handled by interior-point methods, $100\times$ faster. As a single example, a radiation treatment planning problem with more than 100 million coefficients in A can be solved in a few seconds; the same problem takes around one hour to solve using an interior-point method.

1.1 Outline

In §1.2 we describe related work. In §2 we derive the graph form dual problem, and the primal-dual optimality conditions, which we use to motivate the stopping criterion and to interpret the iterates of the algorithm. In §3 we describe the ADMM-based graph form algorithm, and analyze the properties of its iterates, giving some results that did not appear in [PB13a]. In §4 we address the topic of pre-conditioning, and suggest novel pre-conditioning and parameter selection techniques. In §5 we describe our implementation POGS, and in §6 we report performance results on various problem families.

1.2 Related work

Many generic methods can be used to solve the graph form problem (1), including projected gradient descent [CM87], projected subgradient methods [Pol87, Chap. 5] [Sho98], operator splitting methods [LM79] [ES08], interior-point methods [NW99, Chap. 19] [BTN01, Chap. 6] and many more. (Of course many of these methods can only be used when additional

assumptions are made on f and g, *e.g.*, differentiability or strong convexity.) For example, if f and g are separable and smooth for their epigraphs), the problem (1) can be solved by Newton's method, with each iteration requiring the solution of a set of linear equations that requires $O(\max\{m,n\}\min\{m,n\}^2)$ floating point operations (flops) when A is dense. If f and g are separable and have smooth barrier functions for their epigraphs, the problem (1) can be solved by an interior-point method, which in practice always takes no more than a few tens of iterations, with each iteration involving the solution of a system of linear equations that requires $O(\max\{m,n\}\min\{m,n\}^2)$ flops when A is dense [BV04, Chap. 11][NW99, Chap. 19].

We now turn to first-order methods for the graph form problem (1). In [BAC11] Briceño-Arias and Combettes describe methods for solving a generalized version of (1), including a forward-backward-forward algorithm and one based on Douglas-Rachford splitting [DR56]. Their methods are especially interesting in the case when A represents an abstract operator, and one only has access to A through Ax and A^Ty. In [OV14] O'Connor and Vandenberghe propose a primal-dual method for the graph form problem where A is the sum of two structured matrices. They contrast it with methods such as Spingarn's method of partial inverses [Spi85], Douglas-Rachford splitting, and the Chambolle-Pock method [CP11a].

Davis and Yin [DY14] analyze convergence rates for different operator splitting methods, and in [Gis15] Giselsson proves the tightness of linear convergence for the operator splitting problems considered [GB14b]. Goldstein et al. [GOSB14] derive Nesterov-type acceleration, and show $O(1/k^2)$ convergence for problems where f and g are both strongly convex.

Nishihara et al. [NLR+15] introduce a parameter selection framework for ADMM with over relaxation [EB92]. The framework is based on solving a fixed-size semidefinite program (SDP). They also make the assumption that f is strongly convex. Ghadimi et al. [GTSJ13] derive optimal parameter choices for the case when f and g are both quadratic. In [GB14b] Giselsson and Boyd show how to choose metrics to optimize the convergence bound, and in [GB14a] Giselsson and Boyd suggest a diagonal pre-conditioning scheme for graph form problems based on semidefinite programming. This scheme is primarily relevant in small to medium scale problems, or situations where many different graph form problems, with the same matrix A, are to be solved. It is clear from these papers (and indeed, a general rule) that the practical convergence of first-order methods depends heavily on algorithm parameter choices.

GPUs are used extensively for training neural networks [NCL+11, CMM+11, KSH12, CHW+13], and they are slowly gaining popularity in convex optimization as well [PC11, COPB13, WB14].

2 Optimality conditions and duality

2.1 Dual graph form problem

The Lagrange dual function of (1) is given by

$$\inf_{x,y} f(y) + g(x) + \nu^T(Ax - y) = -f^*(\nu) - g^*(-A^T\nu)$$

where $\nu \in \mathbf{R}^n$ is the dual variable associated with the equality constraint, and f^* and g^* are the conjugate functions of f and g respectively [BV04, Chap. 4]. Introducing the variable $\mu = -A^T\nu$, we can write the dual problem as

$$\begin{array}{ll} \text{maximize} & -f^*(\nu) - g^*(\mu) \\ \text{subject to} & \mu = -A^T\nu. \end{array} \tag{2}$$

The dual problem can be written as a graph form problem if we negate the objective and minimize rather than maximize. The dual graph form problem (2) is related to the primal graph form problem (1) by switching the roles of the variables, replacing the objective function terms with their conjugates, and replacing A with $-A^T$.

The primal and dual objectives are $p(x, y) = f(y) + g(x)$ and $d(\mu, \nu) = -f^*(\nu) - g^*(\mu)$ respectively, giving us the duality gap

$$\eta = p(x, y) - d(\mu, \nu) = f(y) + f^*(\nu) + g(x) + g^*(\mu). \tag{3}$$

We have $\eta \geq 0$, for any primal and dual feasible tuple (x, y, μ, ν). The duality gap η gives a bound on the suboptimality of (x, y) (for the primal problem) and also (μ, ν) for the dual problem:

$$f(y) + g(x) \leq p^\star + \eta, \qquad -f^*(\nu) - g^*(\mu) \geq p^\star - \eta.$$

2.2 Optimality conditions

The optimality conditions for (1) are readily derived from the dual problem. The tuple (x, y, μ, ν) satisfies the following three conditions if and only it is optimal.

Primal feasibility:

$$y = Ax. \tag{4}$$

Dual feasibility:

$$\mu = -A^T\nu. \tag{5}$$

Zero gap:

$$f(y) + f^*(\nu) + g(x) + g^*(\mu) = 0. \tag{6}$$

If both (4) and (5) hold, then the zero gap condition (6) can be replaced by the Fenchel equalities

$$f(y) + f^*(\nu) = \nu^T y, \quad g(x) + g^*(\mu) = \mu^T x. \tag{7}$$

We refer to a tuple (x, y, μ, ν) that satisfies (7) as *Fenchel feasible*. To verify the statement, we add the two equations in (7), which yields

$$f(y) + f^*(\nu) + g(x) + g^*(\mu) = y^T\nu + x^T\mu = (Ax)^T\nu - x^TA^T\nu = 0.$$

The Fenchel equalities (7) are is also equivalent to

$$\nu \in \partial f(y), \quad \mu \in \partial g(x), \tag{8}$$

where ∂ denotes the subdifferential, which follows because

$$\nu \in \partial f(y) \Leftrightarrow \sup_z \left(z^T\nu - f(z)\right) = \nu^T y - f(y) \Leftrightarrow f(y) + f^*(\nu) = \nu^T y.$$

In the sequel we will assume that strong duality holds, meaning that there exists a tuple $(x^\star, y^\star, \mu^\star, \nu^\star)$ which satisfies all three optimality conditions.

3 Algorithm

3.1 Graph projection splitting

In [PB13a] Parikh et al. apply ADMM [BPC+11, §5] to the problem of minimizing $f(y) + g(x)$, subject to the constraint $(x, y) \in \mathcal{G}$. This yields the *graph projection splitting* algorithm 1.

Algorithm 1 Graph projection splitting

Input: A, f, g

1: Initialize $(x^0, y^0, \tilde{x}^0, \tilde{y}^0) = 0,\ k = 0$
2: **repeat**
3: $\quad (x^{k+1/2},\ y^{k+1/2}) := \left(\mathbf{prox}_g(x^k - \tilde{x}^k),\ \mathbf{prox}_f(y^k - \tilde{y}^k)\right)$
4: $\quad (x^{k+1}, y^{k+1}) := \Pi(x^{k+1/2} + \tilde{x}^k,\ y^{k+1/2} + \tilde{y}^k)$
5: $\quad (\tilde{x}^{k+1}, \tilde{y}^{k+1}) := (\tilde{x}^k + x^{k+1/2} - x^{k+1},\ \tilde{y}^k + y^{k+1/2} - y^{k+1})$
6: $\quad k := k + 1$
7: **until** converged

The variable k is the iteration counter, $x^{k+1}, x^{k+1/2} \in \mathbf{R}^n$ and $y^{k+1}, y^{k+1/2}, \in \mathbf{R}^m$ are primal variables, $\tilde{x}^{k+1} \in \mathbf{R}^n$ and $\tilde{y}^{k+1} \in \mathbf{R}^m$ are scaled dual variables, Π denotes the (Euclidean) projection onto the graph $\mathcal{G}$,

$$\mathbf{prox}_f(v) = \operatorname*{argmin}_y \left(f(y) + (\rho/2)\left\|y - v\right\|_2^2\right)$$

is the proximal operator of f (and similarly for g), and $\rho > 0$ is the proximal parameter. The projection Π can be explicitly expressed as the linear operator

$$\Pi(c, d) = K^{-1}\begin{bmatrix} c + A^T d \\ 0 \end{bmatrix}, \qquad K = \begin{bmatrix} I & A^T \\ A & -I \end{bmatrix}. \tag{9}$$

Roughly speaking, in steps 3 and 5, the x (and $\tilde{x}$) and y (and $\tilde{y}$) variables do not mix; the computations can be carried out in parallel. The projection step 4 mixes the $x, \tilde{x}$ and $y, \tilde{y}$ variables.

General convergence theory for ADMM [BPC+11, §3.2] guarantees that (with our assumption on the existence of a solution)

$$(x^{k+1}, y^{k+1}) - (x^{k+1/2}, y^{k+1/2}) \to 0, \quad f(y^k) + g(x^k) \to p^\star, \quad (\tilde{x}^k, \tilde{y}^k) \to (\tilde{x}^\star, \tilde{y}^\star), \tag{10}$$

as $k \to \infty$.

3.2 Extensions

We discuss three common extensions that can be used to speed up convergence in practice: over-relaxation, approximate projection, and varying penalty.

Over-relaxation. Replacing $x^{k+1/2}$ by $\alpha x^{k+1/2} + (1-\alpha)x^k$ in the projection and dual update steps is known as over-relaxation if $\alpha > 1$ or under-relaxation if $\alpha < 1$. The algorithm is guaranteed to converge [EB92] for any $\alpha \in (0, 2)$; it is observed in practice [OSB13] [AHW12] that using an over-relaxation parameter in the range [1.5, 1.8] can improve practical convergence.

Approximate projection. Instead of computing the projection Π exactly one can use an approximation $\tilde{\Pi}$, with the only restriction that

$$\textstyle\sum_{k=0}^{\infty} \|\Pi(x^{k+1/2}, y^{k+1/2}) - \tilde{\Pi}(x^{k+1/2}, y^{k+1/2})\|_2 < \infty$$

must hold. This is known as approximate projection [OSB13], and is guaranteed to converge [BAC11]. This extension is particularly useful if the approximate projection is computed using an indirect or iterative method.

Varying penalty. Large values of ρ tend to encourage primal feasibility, while small values tend to encourage dual feasibility [BPC+11, §3.4.1]. A common approach is to adjust or vary ρ in each iteration, so that the primal and dual residuals are (roughly) balanced in magnitude. When doing so, it is important to re-scale $(\tilde{x}^{k+1}, \tilde{y}^{k+1})$ by a factor ρ^k/ρ^{k+1}.

3.3 Feasible iterates

In each iteration, algorithm 1 produces sets of points that are either primal, dual, or Fenchel feasible. Define

$$\mu^k = -\rho\tilde{x}^k, \quad \nu^k = -\rho\tilde{y}^k, \quad \mu^{k+1/2} = -\rho(x^{k+1/2} - x^k + \tilde{x}^k), \quad \nu^{k+1/2} = -\rho(y^{k+1/2} - y^k + \tilde{y}^k).$$

The following statements hold.

1. The pair (x^{k+1}, y^{k+1}) is primal feasible, since it is the projection onto the graph $\mathcal{G}$.

2. The pair (μ^{k+1}, ν^{k+1}) is dual feasible, as long as (μ^0, ν^0) is dual feasible and (x^0, y^0) is primal feasible. Dual feasibility implies $\mu^{k+1} + A^T\nu^{k+1} = 0$, which we show using the update equations in algorithm 1:

$$\begin{aligned}\mu^{k+1} + A^T\nu^{k+1} &= -\rho(\tilde{x}^k + x^{k+1/2} - x^{k+1} + A^T(\tilde{y}^k + y^{k+1/2} - y^{k+1})) \\ &= -\rho(\tilde{x}^k + A^T\tilde{y}^k + x^{k+1/2} + A^T y^{k+1/2} - (I + A^TA)x^{k+1}),\end{aligned}$$

where we substituted $y^{k+1} = Ax^{k+1}$. From the projection operator in (9) it follows that $(I + A^TA)x^{k+1} = x^{k+1/2} + A^Ty^{k+1/2}$, therefore

$$\mu^{k+1} + A^T\nu^{k+1} = -\rho(\tilde{x}^k + A^T\tilde{y}^k) = \mu^k + A^T\nu^k = \mu^0 + A^T\nu^0,$$

where the last equality follows from an inductive argument. Since we made the assumption that (μ^0, ν^0) is dual feasible, we can conclude that (μ^{k+1}, ν^{k+1}) is also dual feasible.

3. The tuple $(x^{k+1/2}, y^{k+1/2}, \mu^{k+1/2}, \nu^{k+1/2})$ is Fenchel feasible. From the definition of the proximal operator,

$$\begin{aligned}x^{k+1/2} = \operatorname*{argmin}_x \left(g(x) + (\rho/2)\left\|x - x^k + \tilde{x}^k\right\|_2^2\right) &\Leftrightarrow 0 \in \partial g(x^{k+1/2}) + \rho(x^{k+1/2} - x^k + \tilde{x}^k) \\ &\Leftrightarrow \mu^{k+1/2} \in \partial g(x^{k+1/2}).\end{aligned}$$

By the same argument $\nu^{k+1/2} \in \partial f(y^{k+1/2})$.

Applying the results in (10) to the dual variables, we find $\nu^{k+1/2} \to \nu^\star$ and $\mu^{k+1/2} \to \mu^\star$, from which we conclude that $(x^{k+1/2}, y^{k+1/2}, \mu^{k+1/2}, \nu^{k+1/2})$ is primal and dual feasible in the limit.

3.4 Stopping criteria

In §3.3 we noted that either (4, 5, 6) or (4, 5, 7) are sufficient for optimality. We present two different stopping criteria based on these conditions.

Residual based stopping. The tuple $(x^{k+1/2}, y^{k+1/2}, \mu^{k+1/2}, \nu^{k+1/2})$ is Fenchel feasible in each iteration, but only primal and dual feasible in the limit. Accordingly, we propose the residual based stopping criterion

$$\|Ax^{k+1/2} - y^{k+1/2}\|_2 \leq \epsilon^{\text{pri}}, \quad \|A^T\nu^{k+1/2} + \mu^{k+1/2}\|_2 \leq \epsilon^{\text{dual}}, \tag{11}$$

where the ϵ^{pri} and ϵ^{dua} are positive tolerances. These should be chosen as a mixture of absolute and relative tolerances, such as

$$\epsilon^{\text{pri}} = \epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|y^{k+1/2}\|_2, \quad \epsilon^{\text{dual}} = \epsilon^{\text{abs}} + \epsilon^{\text{rel}}\|\mu^{k+1/2}\|_2.$$

Reasonable values for ϵ^{abs} and ϵ^{rel} are in the range $[10^{-4}, 10^{-2}]$.

Gap based stopping. The tuple (x^k, y^k, μ^k, ν^k) is primal and dual feasible, but only Fenchel feasible in the limit. We propose the gap based stopping criteria

$$\eta^k = f(y^k) + g(x^k) + f^*(\nu^k) + g^*(\mu^k) \leq \epsilon^{\text{gap}},$$

where ϵ^{gap} should be chosen relative to the current objective value, *i.e.*,

$$\epsilon^{\text{gap}} = \epsilon^{\text{abs}} + \epsilon^{\text{rel}}|f(y^k) + g(x^k)|.$$

Here too, reasonable values for ϵ^{abs} and ϵ^{rel} are in the range $[10^{-4}, 10^{-2}]$.

Although the gap based stopping criteria is very informative, since it directly bounds the suboptimality of the current iterate, it suffers from the drawback that f, g, f^* and g^* must all have full domain, since otherwise the gap η^k can be infinite. Indeed, the gap η^k is almost always infinite when f or g represent constraints.

3.5 Implementation

Projection. There are different ways to evaluate the projection operator Π, depending on the structure and size of A.

One simple method that can be used if A is sparse and not too large is a direct sparse factorization. The matrix K is quasi-definite, and therefore the LDL^T decomposition is well defined [Van95]. Since K does not change from iteration to iteration, the factors L and D (and the permutation or elimination ordering) can be computed in the first iteration (*e.g.*, using CHOLMOD [CDHR08]) and re-used in subsequent iterations. This is known as *factorization caching* [BPC+11, §4.2.3] [PB13a, §A.1]. With factorization caching, we get a (potentially) large speedup in iterations, after the first one.

If A is dense, and $\min(m, n)$ is not too large, then block elimination [BV04, Appendix C] can be applied to K [PB13a, Appendix A], yielding the reduced update

$$\begin{aligned} x^{k+1} &:= (A^T A + I)^{-1}(c + A^T d) \\ y^{k+1} &:= Ax^{k+1} \end{aligned}$$

if $m \geq n$, or

$$\begin{aligned} y^{k+1} &:= d + (AA^T + I)^{-1}(Ac - d) \\ x^{k+1} &:= c - A^T(d - y^{k+1}) \end{aligned}$$

if $m < n$. Both formulations involve forming and solving a system of $\min(m, n)$ equations with $\min(m, n)$ unknowns. Since the coefficient matrix is symmetric positive definite, we can use the Cholesky decomposition. Forming the coefficient matrix $A^T A + I$ or $AA^T + I$ dominates the computatation. Here too we can take advantage of factorization caching.

The regular structure of dense matrices allows us to analyze the computational complexity of each step. We define $q = \min(m, n)$ and $p = \max(m, n)$. The first iteration involves the factorization and the solve step; subsequent iterations only require the solve step. The

computational cost of the factorization is the combined cost of computing A^TA (or AA^T, whichever is smaller), at a cost of pq^2 flops, in addition to the Cholesky decomposition, at a cost of $(1/3)q^3$ flops. The solve step consists of two matrix-vector multiplications at a cost of $4pq$ flops and solving a triangular system of equations at a cost of q^2 flops. The total cost of the first iteration is $O(pq^2)$ flops, while each subsequent iteration only costs $O(pq)$ flops, showing that we obtain a savings by a factor of q flops, after the first iteration, by using factorization caching.

For very large problems direct methods are no longer practical, at which point indirect (iterative) methods can be used. Fortunately, as the primal and dual variables converge, we are guaranteed that $(x^{k+1/2}, y^{k+1/2}) \to (x^{k+1}, y^{k+1})$, meaning that we will have a good initial guess we can use to initialize the iterative method to (approximately) evaluate the projection. One can either apply CGLS (conjugate gradient least-squares) [HS52] or LSQR [PS82] to the reduced update or apply MINRES (minimum residual) [PS75] to K directly. It can be shown the latter requires twice the number of iterations as compared to the former, and is therefore not recommended.

Proximal operators. Since the $x, \tilde{x}$ and $y, \tilde{y}$ components are decoupled in the proximal step and dual variable update step, both of these can be done separately, and in parallel for x and y. If either f or g is separable, then the proximal step can be parallelized further. [CP11b, §10.2] contains a table of proximal operators for a wide range of functions, and the monograph [PB13b] details how proximal operators can be computed efficiently, in particular for the case where there is no analytical solution. Typically the cost of computing the proximal operator will be negligible compared to the cost of the projection. In particular, if f and g are separable, then the cost will be $O(m+n)$, and completely parallelizable.

4 Pre-conditioning and parameter selection

The practical convergence of the algorithm (*i.e.*, the number of iterations required before it terminates) can depend greatly on the choice of the proximal parameter ρ, and the scaling of the variables. In this section we analyze these, and suggest a method for choosing ρ and for scaling the variables that (empirically) speeds up practical convergence.

4.1 Pre-conditioning

Consider scaling the variables x and y in (1), by E^{-1} and D respectively, where $D \in \mathbf{R}^{m\times m}$ and $E \in \mathbf{R}^{n\times n}$ are non-singular matrices. We define the scaled variables

$$\hat{y} = Dy, \quad \hat{x} = E^{-1}x,$$

which transforms (1) into

$$\begin{array}{ll} \text{minimize} & f(D^{-1}\hat{y}) + g(E\hat{x}) \\ \text{subject to} & \hat{y} = DAE\hat{x}. \end{array} \tag{12}$$

This is also a graph form problem, and for notational convenience, we define

$$\hat{A} = DAE, \quad \hat{f}(\hat{y}) = f(D^{-1}\hat{y}), \quad \hat{g}(\hat{x}) = g(E\hat{x}),$$

so that the problem can be written as

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

We refer to this problem as the pre-conditioned version of (1). Our goal is to choose D and E so that (a) the algorithm applied to the pre-conditioned problem converges in fewer steps in practice, and (b) the additional computational cost due to the pre-conditioning is minimal.

Graph projection splitting applied to the pre-conditioned problem (12) can be interpreted in terms of the original iterates. The proximal step iterates are redefined as

$$\begin{aligned} x^{k+1/2} &= \operatorname*{argmin}_x \left(g(x) + (\rho/2)\|x - x^k + \tilde{x}^k\|^2_{(EE^T)^{-1}} \right), \\ y^{k+1/2} &= \operatorname*{argmin}_y \left(f(y) + (\rho/2)\|y - y^k + \tilde{y}^k\|^2_{(D^TD)} \right), \end{aligned}$$

and the projected iterates are the result of the weighted projection

$$\begin{array}{ll} \text{minimize} & (1/2)\|x - x^{k+1/2}\|^2_{(EE^T)^{-1}} + (1/2)\|y - y^{k+1/2}\|^2_{(D^TD)} \\ \text{subject to} & y = Ax, \end{array}$$

where $\|x\|_P = \sqrt{x^TPx}$ for a symmetric positive-definite matrix P. This projection can be expressed as

$$\Pi(c,d) = \hat{K}^{-1}\begin{bmatrix} (EE^T)^{-1}c + A^TD^TDd \\ 0 \end{bmatrix}, \qquad \hat{K} = \begin{bmatrix} (EE^T)^{-1} & A^TD^TD \\ D^TDA & -D^TD \end{bmatrix}.$$

Notice that graph projection splitting is invariant to orthogonal transformations of the variables x and y, since the pre-conditioners only appear in terms of D^TD and EE^T. In particular, if we let $D = U^T$ and $E = V$, where $A = U\Sigma V^T$, then the pre-conditioned constraint matrix $\hat{A} = DAE = \Sigma$ is diagonal. We conclude that any graph form problem can be pre-conditioned to one with a diagonal non-negative constraint matrix Σ. For analysis purposes, we are therefore free to assume that A is diagonal. We also note that for orthogonal pre-conditioners, there exists an analytical relationship between the original proximal operator and the pre-conditioned proximal operator. With $\phi(x) = \varphi(Qx)$, where Q is any orthogonal matrix ($Q^TQ = QQ^T = I$), we have

$$\mathbf{prox}_\phi(v) = Q^T\mathbf{prox}_\varphi(Qv).$$

While the proximal operator of ϕ is readily computed, orthogonal pre-conditioners destroy separability of the objective. As a result, we can not easily combine them with other pre-conditioners.

Multiplying D by a scalar α and dividing E by the same scalar has the effect of scaling ρ by a factor of α^2. It however has no effect on the projection step, showing that ρ can be thought of as the relative scaling of D and E.

In the case where f and g are separable and both D and E are diagonal, the proximal step takes the simplified form

$$\begin{aligned} x_j^{k+1/2} &= \underset{x_j}{\operatorname{argmin}} \left(g_j(x_j) + (\rho_j^E/2)(x_j - x_j^k + \tilde{x}_j^k)^2 \right) && j = 1, \ldots, n \\ y_i^{k+1/2} &= \underset{y_i}{\operatorname{argmin}} \left(f_i(y_i) + (\rho_i^D/2)(y_i - y_i^k + \tilde{y}_i^k)^2 \right) && i = 1, \ldots, m, \end{aligned}$$

where $\rho_j^E = \rho / E_{jj}^2$ and $\rho_i^D = \rho D_{ii}^2$. Since only ρ is modified, any routine capable of computing $\mathbf{prox}_f$ and $\mathbf{prox}_g$ can also be used to compute the pre-conditioned proximal update.

4.1.1 Effect of pre-conditioning on projection

For the purpose of analysis, we will assume that $A = \Sigma$, where Σ is a non-negative diagonal matrix. The projection operator simplifies to

$$\Pi(c, d) = \begin{bmatrix} (I + \Sigma^T\Sigma)^{-1} & (I + \Sigma^T\Sigma)^{-1}\Sigma^T \\ (I + \Sigma\Sigma^T)^{-1}\Sigma & (I + \Sigma\Sigma^T)^{-1}\Sigma\Sigma^T \end{bmatrix} \begin{bmatrix} c \\ d \end{bmatrix},$$

which means the projection step can be written explicitly as

$$\begin{aligned} x_i^{k+1} &= \frac{1}{1+\sigma_i^2}(x_i^{k+1/2} + \tilde{x}_i^k + \sigma_i(y_i^{k+1/2} + \tilde{y}_i^k)) && i = 1, \ldots, \min(m, n) \\ x_i^{k+1} &= x_i^{k+1/2} + \tilde{x}_i^k && i = \min(m, n) + 1, \ldots, n \\ y_i^{k+1} &= \frac{\sigma_i}{1+\sigma_i^2}(x_i^{k+1/2} + \tilde{x}_i^k + \sigma_i(y_i^{k+1/2} + \tilde{y}_i^k)) && i = 1, \ldots, \min(m, n) \\ y_i^{k+1} &= 0 && i = \min(m, n) + 1, \ldots, m, \end{aligned}$$

where σ_i is the ith diagonal entry of Σ and subscripted indices of x and y denote the ith entry of the respective vector. Notice that the projected variables x_i^{k+1} and y_i^{k+1} are equally dependent on $(x_i^{k+1/2} + \tilde{x}_i^k)$ and $\sigma_i(y_i^{k+1/2} + \tilde{y}_i^k)$. If σ_i is either significantly smaller or larger than 1, then the terms x_i^{k+1} and y_i^{k+1} will be dominated by either $(x_i^{k+1/2} + \tilde{x}_i^k)$ or $(y_i^{k+1/2} + \tilde{y}_i^k)$. However if $\sigma_i = 1$, then the projection step exactly averages the two quantities

$$x_i^{k+1} = y_i^{k+1} = \frac{1}{2}(x_i^{k+1/2} + \tilde{x}_i^k + y_i^{k+1/2} + \tilde{y}_i^k) \qquad i = 1, \ldots, \min(m, n).$$

As pointed out in §3, the projection step mixes the variables x and y. For this to approximately reduce to averaging, we need $\sigma_i \approx 1$.

4.1.2 Choosing D and E

The analysis suggests that the algorithm will converge quickly when the singular values of DAE are all near one, *i.e.*,

$$\mathbf{cond}\big(DAE\big) \approx 1, \quad \|DAE\|_2 \approx 1. \tag{13}$$

(This claim is also supported in [GB14c], and is consistent with our computational experience.) Pre-conditioners that exactly satisfy these conditions can be found using the singular value decomposition of A. They will however be of little use, since such pre-conditioners generally destroy our ability to evaluate the proximal operators of $\hat{f}$ and $\hat{g}$ efficiently.

So we seek choices of D and E for which (13) holds (very) approximately, and for which the proximal operators of $\hat{f}$ and $\hat{g}$ can still be efficiently computed. We now specialize to the special case when f and g are separable. In this case, diagonal D and E are candidates for which the proximal operators are still easily computed. (The same ideas apply to block separable f and g, where we impose the further constraint that the diagonal entries within a block are the same.) So we now limit ourselves to the case of diagonal pre-conditioners.

Diagonal matrices that minimize the condition number of DAE, and therefore approximately satisfy the first condition in (13), can be found exactly, using semidefinite programming [BEGFB94, §3.1]. But this computation is quite involved, and may not be worth the computational effort since the conditions (13) are just a heuristic for faster convergence. (For control problems, where the problem is solved many times with the same matrix A, this approach makes sense; see [GB14a].)

A heuristic that tends to minimize the condition number is to equilibrate the matrix, *i.e.*, choose D and E so that the rows all have the same p-norm, and the columns all have the same p-norm. (Such a matrix is said to be equilibrated.) This corresponds to finding D and E so that

$$|DAE|^p\mathbf{1} = \alpha\mathbf{1}, \qquad \mathbf{1}^T|DAE|^p = \beta\mathbf{1}^T,$$

where $\alpha, \beta > 0$. Here the notation $|\cdot|^p$ should be understood in the elementwise sense. Various authors [OSB13], [COPB13], [Bra10] suggest that equilibration can decrease the number of iterations needed for operator splitting and other first order methods. One issue that we need to address is that not every matrix can be equilibrated. Given that equilibration is only a heuristic for achieving $\sigma_i(DAE) \approx 1$, which is in turn a heuristic for fast convergence of the algorithm, partial equilibration should serve the same purpose just as well.

Sinkhorn and Knopp [SK67] suggest a method for matrix equilibration for $p < \infty$, and A is square and has full support. In the case $p = \infty$, the Ruiz algorithm [Rui01] can be used. Both of these methods fail (as they must) when the matrix A cannot be equilibrated. We give below a simple modification of the Sinkhorn-Knopp algorithm, modified to handle the case when A is non-square, or cannot be equilibrated.

Choosing pre-conditioners that satisfy $\|DAE\|_2 = 1$ can be achieved by scaling D and E by $\sigma_{\max}(DAE)^{-q}$ and $\sigma_{\max}(DAE)^{q-1}$ respectively for $q \in \mathbf{R}$. The quantity $\sigma_{\max}(DAE)$ can be approximated using power iteration, but we have found it is unnecessary to exactly enforce $\|DAE\|_2 = 1$. A more computationally efficient alternative is to replace $\sigma_{\max}(DAE)$

by $\|DAE\|_F/\sqrt{\min(m,n)}$. This quantity coincides with $\sigma_{\max}(DAE)$ when $\mathbf{cond}(DAE) = 1$. If DAE is equilibrated and $p = 2$, this scaling corresponds to $(DAE)^T(DAE)$ (or $(DAE)(DAE)^T$ when $m < n$) having unit diagonal.

4.2 Regularized equilibration

In this section we present a self-contained derivation of our matrix-equilibration method. It is similar to the Sinkhorn-Knopp algorithm, but also works when the matrix is non-square or cannot be exactly equilibrated.

Consider the convex optimization problem with variables u and v,

$$\text{minimize} \quad \sum_{i=1}^{m}\sum_{j=1}^{n} |A_{ij}|^p e^{u_i+v_j} - n\mathbf{1}^T u - m\mathbf{1}^T v + \gamma\left[n\sum_{i=1}^{m} e^{u_i} + m\sum_{j=1}^{n} e^{v_j}\right], \tag{14}$$

where $\gamma \geq 0$ is a regularization parameter. The objective is bounded below for any $\gamma > 0$. The optimality conditions are

$$\sum_{j=1}^{n} |A_{ij}|^p e^{u_i+v_j} - n + n\gamma e^{u_i} = 0, \quad i = 1,\ldots,m,$$
$$\sum_{i=1}^{m} |A_{ij}|^p e^{u_i+v_j} - m + m\gamma e^{v_j} = 0, \quad j = 1,\ldots,n.$$

By defining $D_{ii} = e^{u_i/p}$ and $E_{jj}^p = e^{v_j/p}$, these conditions are equivalent to

$$|DAE|^p\mathbf{1} + n\gamma D\mathbf{1} = n\mathbf{1}, \quad \mathbf{1}^T|DAE|^p + m\gamma\mathbf{1}^T E = m\mathbf{1}^T,$$

where $\mathbf{1}$ is the vector with all entries one. When $\gamma = 0$, these are the conditions for a matrix to be equilibrated. The objective may not be bounded when $\gamma = 0$, which exactly corresponds to the case when the matrix cannot be equilibrated. As $\gamma \to \infty$, both D and E converge to the scaled identity matrix $(1/\gamma)I$, showing that γ can be thought of as a regularizer on the elements of D and E. If D and E are optimal, then the two equalities

$$\mathbf{1}^T|DAE|^p\mathbf{1} + n\gamma\mathbf{1}^T D\mathbf{1} = mn, \qquad \mathbf{1}^T|DAE|^p\mathbf{1} + m\gamma\mathbf{1}^T E\mathbf{1} = mn$$

must hold. Subtracting the one from the other, and dividing by γ, we find the relationship

$$n\mathbf{1}^T D\mathbf{1} = m\mathbf{1}^T E\mathbf{1},$$

implying that the average entry in D and E is the same.

There are various ways to solve the optimization problem (14), one of which is to apply coordinate descent. Minimizing the objective in (14) with respect to u_i yields

$$\sum_{j=1}^{n} e^{u_i^k+v_j^{k-1}}|A_{ij}|^p + n\gamma e^{u_i^k} = n \Leftrightarrow e^{u_i^k} = \frac{n}{\sum_{j=1}^{n} e^{v_j^{k-1}}|A_{ij}|^p + n\gamma}$$

and similarly for v_j

$$e^{v_i^k} = \frac{m}{\sum_{i=1}^n e^{u_i^{k-1}} |A_{ij}|^p + m\gamma}.$$

Since the minimization with respect to u_i^k is independent of u_{i-1}^k, the update can be done in parallel for each element of u, and similarly for v. Repeated minimization over u and v will eventually yield values that satisfy the optimality conditions.

Algorithm 2 summarizes the equilibration routine. The inverse operation in steps 4 and 5 should be understood in the element-wise sense.

Algorithm 2 Regularized Sinkhorn-Knopp

Input: $A, \epsilon > 0$, $\gamma > 0$

1: Initialize $e^0 := \mathbf{1}$, $k := 0$
2: **repeat**
3: $\quad k := k + 1$
4: $\quad d^k := n\ (|A|^p e^{k-1} + n\gamma \mathbf{1})^{-1}$
5: $\quad e^k := m\ (|A^T|^p d^k + m\gamma \mathbf{1})^{-1}$
6: **until** $\|e^k - e^{k-1}\|_2 \leq \epsilon$ and $\|d^k - d^{k-1}\|_2 \leq \epsilon$
7: **return** $D := \mathbf{diag}(d^k)^{1/p}$, $E := \mathbf{diag}(e^k)^{1/p}$

4.3 Adaptive penalty update

The projection operator Π does not depend on the choice of ρ, so we are free to update ρ in each iteration, at no extra cost. While the convergence theory only holds for fixed ρ, it still applies if one assumes that ρ becomes fixed after a finite number of iterations [BPC+11].

As a rule, increasing ρ will decrease the primal residual, while decreasing ρ will decrease the dual residual. The authors in [HYW00],[BPC+11] suggest adapting ρ to balance the primal and dual residuals. We have found that substantially better practical convergence can be obtained using a variation on this idea. Rather than balancing the primal and dual residuals, we allow either the primal or dual residual to approximately converge and only then start adjusting ρ. Based on this observation, we propose the following adaptive update scheme.

Once either the primal or dual residual converges, the algorithm begins to steer ρ in a direction so that the other residual also converges. By making small adjustments to ρ, we will tend to remain approximately primal (or dual) feasible once primal (dual) feasibility has been attained. Additionally by requiring a certain number of iterations between an increase in ρ and a decrease (and vice versa), we enforce that changes to ρ do not flip-flop between one direction and the other. The parameter τ determines the relative number of iterations between changes in direction.

Algorithm 3 Adaptive ρ update

Input: $\delta > 1$, $\tau \in (0, 1]$,

Initialize $l := 0$, $u := 0$
repeat
  Apply graph projection splitting
  if $\|A^T\nu^{k+1/2} + \mu^{k+1/2}\|_2 < \epsilon^{\text{dual}}$ and $\tau k > l$ **then**
    $\rho^{k+1} := \delta\rho^k$
    $u := k$
  else if $\|Ax^{k+1/2} - y^{k+1/2}\|_2 < \epsilon^{\text{pri}}$ and $\tau k > u$ **then**
    $\rho^{k+1} := (1/\delta)\rho^k$
    $l := k$
until $\|A^T\nu^{k+1/2} + \mu^{k+1/2}\|_2 < \epsilon^{\text{dual}}$ and $\|Ax^{k+1/2} - y^{k+1/2}\|_2 < \epsilon^{\text{pri}}$

5 Implementation

Proximal Graph Solver (POGS) is an open-source (BSD-3 license) implementation of graph projection splitting, written in C++. It supports both GPU and CPU platforms and includes wrappers for C, MATLAB, and R. POGS handles all combinations of sparse/dense matrices, single/double precision arithmetic, and direct/indirect solvers, with the exception (for now) of sparse indirect solvers. The only dependency is a tuned BLAS library on the respective platform (*e.g.*, cuBLAS or the Apple Accelerate Framework). The source code is available at

`https://github.com/foges/pogs`

In lieu of having the user specify the proximal operators of f and g, POGS contains a library of proximal operators for a variety of different functions. It is currently assumed that the objective is separable, in the form

$$f(y) + g(x) = \sum_{i=1}^{m} f_i(y_i) + \sum_{j=1}^{n} g_j(x_j),$$

where $f_i, g_j : \mathbf{R} \to \mathbf{R} \cup \{\infty\}$. The library contains a set of base functions, and by applying various transformations, the range of functions can been greatly extended. In particular we use the parametric representation

$$f_i(y_i) = c_i h_i(a_i y_i - b_i) + d_i y_i + (1/2)e_i y_i^2,$$

where $a_i, b_i, d_i \in \mathbf{R}$, $c_i, e_i \in \mathbf{R}_+$, and $h_i : \mathbf{R} \to \mathbf{R} \cup \{\infty\}$. The same representation is also used for g_j. It is straightforward to express the proximal operators of f_i in terms of the proximal operator of h_i using the formula

$$\mathbf{prox}_f(v) = \frac{1}{a}\left(\mathbf{prox}_{h,(e+\rho)/(ca^2)}\left(a\left(v\rho - d\right)/(e+\rho) - b\right) + b\right),$$

where for notational simplicity we have dropped the index i in the constants and functions. It is possible for a user to add their own proximal operator function, if it is not in the current library. We note that the separability assumption on f and g is a simplification, rather than a limitation of the algorithm. It allows us to apply the proximal operator in parallel using either CUDA or OpenMP (depending on the platform).

The constraint matrix is equilibrated using algorithm 2, with a choice of $p = 2$ and $\gamma = \frac{m+n}{mn}\sqrt{\epsilon^{\text{cmp}}}$, where ϵ^{cmp} is machine epsilon. Both D and E are rescaled evenly, so that they satisfy $\|DAE\|_F/\sqrt{\min(m,n)} = 1$. The projection Π is computed as outlined in §3.5. We work with the reduced update equations in all versions of POGS. In the indirect case, we chose to use CGLS. The parameter ρ is updated according to algorithm 3. Empirically, we found that $(\delta, \tau) = (1.05, 0.8)$ works well. We also use over-relaxation with $\alpha = 1.7$.

POGS supports warm starting, whereby an initial guess for x^0 and/or ν^0 may be supplied by the user. If only x^0 is provided, then ν^0 will be estimated, and vice-versa. The warm-start feature allows any cached matrices to be used to solve additional problems with the same matrix A.

POGS returns the tuple $(x^{k+1/2}, y^{k+1/2}, \mu^{k+1/2}, \nu^{k+1/2})$, since it has finite primal and dual objectives. The primal and dual residuals will be non-zero and are determined by the specified tolerances.

Future plans for POGS include extension to block-separable f and g (including general cone solvers), additional wrappers for Julia and Python, support for a sparse direct solver, and a multi-GPU extension.

6 Numerical results

To highlight the robustness and general purpose nature of POGS, we tested it on 9 different problem classes using random data, as well as a radiation treatment planning problem using real-world data.

All experiments were performed in single precision arithmetic on a machine equipped with an Intel Core i7-870, 16GB of RAM, and an Nvidia Titan X (Maxwell) GPU. Timing results include the data copy from CPU to GPU.

We compare POGS to SDPT3 [TTT99], an open-source solver that handles linear, second-order, and positive semidefinite cone programs. Since SDPT3 uses an interior-point algorithm, the solution returned will be of high precision, allowing us to verify the accuracy of the solution computed by POGS. Problems that took SDPT3 more than 200 seconds (of which there were many) were aborted.

6.1 Random problem classes

We considered the following 9 problem classes: Basis pursuit, Entropy maximization, Huber fitting, Lasso, Logistic regression, Linear programming, Non-negative least-squares, Portfolio optimization, and Support vector machine fitting. For each problem class, reasonable random instance were generated and solve; details about problem generation can be found

in Appendix A. For each problem class the number of non-zeros in A was varied on a logarithmic scale from 100 to 1 Billion. The aspect ratio of A also varied from 1:10 to 10:1, with the orientation (wide or tall) chosen depending on what was reasonable for each problem. We report running time averaged over all aspect ratios.

The maximum number of iterations was set to 10^4, but all problems converged in fewer iterations, with most problems taking a couple of hundred iterations. The relative tolerance was set to 10^{-3}, and where solutions from SDPT3 were available, we verified that the solutions produced by both solvers matched to 3 decimal places. We omit SDPT3 running times for problems involving exponential cones, since SDPT3 does not support them.

Figure 1 compares the running time of POGS versus SDPT3, for problems where the constraint matrix A is dense. We can make several general observations.

- POGS solves problems that are 3 orders of magnitude larger than SDPT3 in the same amount of time.
- Problems that take 200 seconds in SDPT3 take 0.5 seconds in POGS.
- POGS can solve problems with 1 Billion non-zeros in 10-40 seconds.
- The variation in solve time across different problem classes was similar for POGS and SDPT3, around one order of magnitude.

In summary, POGS is able to solve much larger problems, much faster (to moderate precision).

6.2 Radiation treatment planning

Radiation treatment is used to radiate tumor cells in cancer patients. The goal of radiation treatment planning is to find a set of radiation beam intensities that will deliver a specified radiation dosage to tumor cells, while minimizing the impact on healthy cells. The problem can be stated directly in graph form, with x corresponding to the n beam intensities to be found, y corresponding to the radiation dose received at the m voxels, and the matrix A (whose elements are non-negative) giving the mapping from the beams to the received dosages at the voxels. This matrix comes from geometry, including radiation scattering inside the patient [AHIM06]. The objective g is the indicator function of the non-negative orthant (which imposes the constraint that $x_j \geq 0$), and f is a separable function of the form

$$f_i(y_i) = \begin{cases} w_i^+ y_i & i \text{ corresponds to a non-tumor voxel} \\ w_i^- \max(d_i - y_i, 0) + w_i^+ \max(y_i - d_i, 0) & i \text{ corresponds to a tumor voxel,} \end{cases}$$

where $w_i^+ > 0$ is the (given) weight associated with overdosing voxel i, where $w_i^- > 0$ is the (given) weight associated with underdosing voxel i, and $d_i > 0$ is the target dose, given for each tumor voxel. We can also add the redundant constraint $y_i \geq 0$ by defining $f_i(y_i) = \infty$ for $y_i < 0$.

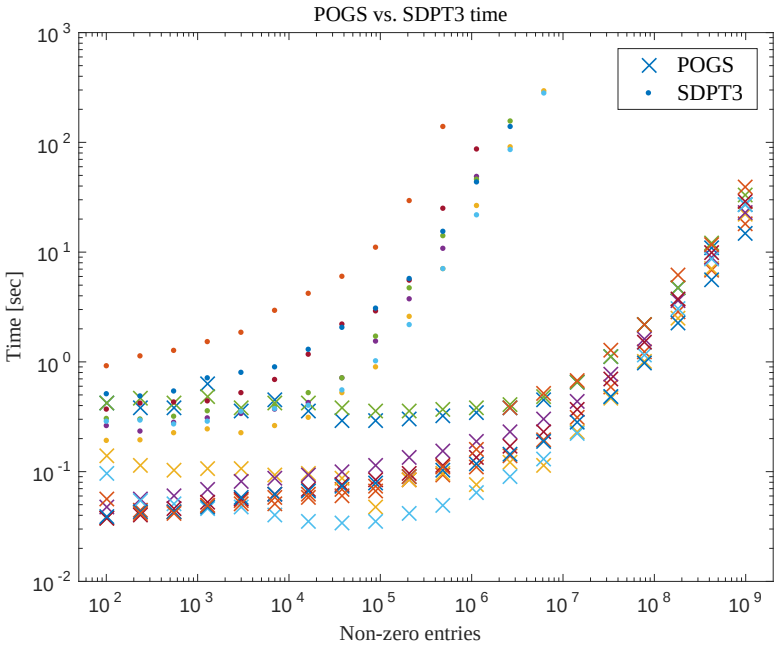

Figure 1: POGS (GPU version) vs. SDPT3 for dense matrices (color represents problem class).

We present results for one instance of this problem, with $m = 360000$ voxels and $n = 360$ beams. The matrix A comes from a real patient, and the objective parameters are chosen to achieve a good clinical plan. The problem is small enough that it can be solved (to high accuracy) by an interior-point method, in around one hour. POGS took a few seconds to solve the problem, producing a solution that was extremely close to the one produced by the interior-point method. In warm start mode, POGS could solve problem instances (obtained by varying the objective parameters) in under one second, allowing for real-time tuning of the treatment plan (by adjusting the objective function weights) by a radiation oncologist.

7 Acknowledgments

We thank Baris Ungun for testing POGS and providing valuable feedback, and the radiation treatment data. We also thank Michael Saunders for numerous discussions about solving large sparse systems, and Patrick Combettes for valuable suggestions and pointing us to very relevant work. This research was funded by DARPA XDATA and Adobe.

A Problem generation details

In this section we describe how the problems in §6.1 were generated.

A.1 Basis pursuit

The basis pursuit problem [CDS98] seeks the smallest vector in the ℓ_1-norm sense that satisfies a set of underdetermined linear equality constraints. The objective has the effect of finding a sparse solution. It can be stated as

$$\begin{array}{ll} \text{minimize} & \|x\|_1 \\ \text{subject to} & b = Ax, \end{array}$$

with equivalent graph form representation

$$\begin{array}{ll} \text{minimize} & I(y = b) + \|x\|_1 \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0, 1)$. To construct b we first generated a vector $v \in \mathbf{R}^n$ as

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 1/2 \\ \mathcal{N}(0, 1/n) & \text{otherwise,} \end{cases}$$

we then let $b = Av$. In each instance we chose $m > n$.

A.2 Entropy maximization

The entropy maximization problem [BV04] seeks a probability distribution with maximum entropy that satisfies a set of m affine inequalities, which can be interpreted as bounds on the expectations of arbitrary functions. It can be stated as

$$\begin{array}{ll} \text{maximize} & -\sum_{i=1}^n x_i \log x_i \\ \text{subject to} & \mathbf{1}^T x = 1, \quad Ax \leq b, \end{array}$$

with equivalent graph form representation

$$\begin{array}{ll} \text{minimize} & I(y_{1:m} \leq b) + I(y_{m+1} = 1) + \sum_{i=1}^n x_i \log x_i \\ \text{subject to} & y = \begin{bmatrix} A \\ \mathbf{1}^T \end{bmatrix} x. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0, n)$. To construct b, we first generated a vector $v \in \mathbf{R}^n$ as $v_i \sim U[0, 1]$, then we set $b = Fv/(\mathbf{1}^T v)$. This ensures that there exists a feasible x. In each instance we chose $m < n$.

A.3 Huber fitting

Huber fitting or robust regression [Hub64] performs linear regression under the assumption that there are outliers in the data. The problem can be stated as

$$\text{minimize} \quad \sum_{i=1}^m \text{huber}(b_i - a_i^T x),$$

where the Huber loss function is defined as

$$\text{huber}(x) = \begin{cases} (1/2)x^2 & |x| \leq 1 \\ |x| - (1/2) & |x| > 1 \end{cases}$$

The graph form representation of this problem is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^{n} \text{huber}(b_i - y_i) \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0, n)$. To construct b, we first generated a vector $v \in \mathbf{R}^n$ as $v_i \sim \mathcal{N}(0, 1/n)$ then we generated a noise vector ε with elements

$$\varepsilon_i \sim \begin{cases} \mathcal{N}(0, 1/4) & \text{with probability } p = 0.95 \\ U[0, 10] & \text{otherwise.} \end{cases}$$

Lastly we constructed $b = Av + \varepsilon$. In each instance we chose $m > n$.

A.4 Lasso

The lasso problem [Tib96] seeks to perform linear regression under the assumption that the solution is sparse. An ℓ_1 penalty is added to the objective to encourage sparsity. It can be stated as

$$\text{minimize} \quad \|Ax - b\|_2^2 + \lambda \|x\|_1,$$

with graph form representation

$$\begin{array}{ll} \text{minimize} & \|y - b\|_2 + \lambda \|x\|_1 \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0, 1)$. To construct b we first generated a vector $v \in \mathbf{R}^n$, with elements

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 1/2 \\ \mathcal{N}(0, 1/n) & \text{otherwise.} \end{cases}$$

We then let $b = Av + \varepsilon$, where ε represents the noise and was generated as $\varepsilon_i \sim \mathcal{N}(0, 1/4)$. The value of λ was set to $(1/5)\|A^T b\|_\infty$. This is a reasonable choice since $\|A^T b\|_\infty$ is the critical value of λ above which the solution of the Lasso problem is $x = 0$. In each instance we chose $m < n$.

A.5 Logistic regression

Logistic regression [HTF09] fits a probability distribution to a binary class label. Similar to the Lasso problem (A.4) a sparsifying ℓ_1 penalty is often added to the coefficient vector. It can be stated as

$$\text{minimize} \quad \sum_{i=1}^m \left(\log(1+\exp(x^T a_i)) - b_i x^T a_i\right) + \lambda\|x\|_1,$$

where $b_i \in \{0,1\}$ is the class label of the ith sample, and a_i^T is the ith row of A. The graph form representation of this problem is

$$\begin{array}{ll} \text{minimize} & \sum_{i=1}^m \left(\log(1+\exp(y_i)) - b_i y_i\right) + \lambda\|x\|_1, \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0,1)$. To construct b we first generated a vector $v \in \mathbf{R}^n$, with elements

$$v_i \sim \begin{cases} 0 & \text{with probability } p = 1/2 \\ \mathcal{N}(0, 1/n) & \text{otherwise.} \end{cases}$$

We then constructed the entries of b as

$$b_i \sim \begin{cases} 0 & \text{with probability } p = 1/(1+\exp(-a_i^T v)) \\ 1 & \text{otherwise.} \end{cases}$$

The value of λ was set to $(1/10)\|A^T((1/2)\mathbf{1} - b)\|_\infty$. ($\|A^T((1/2)\mathbf{1} - b)\|_\infty$ is the critical of λ above which the solution is $x = 0$.) In each instance we chose $m > n$.

A.6 Linear program

Linear programs [BV04] seek to minimize a linear function subject to linear inequality constraints. It can be stated as

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax \leq b, \end{array}$$

and has graph form representation

$$\begin{array}{ll} \text{minimize} & c^T x + I(y \leq b) \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0,1)$. To construct b we first generated a vector $v \in \mathbf{R}^n$, with elements

$$v_i \sim \mathcal{N}(0, 1/n).$$

We then generated b as $b = Av + \varepsilon$, where $\varepsilon_i \sim U[0, 1/10]$. The vector c was constructed in a similar fashion. First we generate a vector $u \in \mathbf{R}^m$, with elements

$$u_i \sim U[0,1],$$

then we constructed $c = -A^T u$. This method guarantees that the problem is bounded. In each instance we chose $m > n$.

A.7 Non-negative least-squares

Non-negative least-squares [CP09] seeks a minimizer of a least-squares problem subject to the solution vector being non-negative. This comes up in applications where the solution represents real quantities. The problem can be stated as

$$\begin{array}{ll} \text{minimize} & \|Ax - b\|_2 \\ \text{subject to} & x \geq 0, \end{array}$$

and has graph form representation

$$\begin{array}{ll} \text{minimize} & \|y - b\|_2^2 + I(x \geq 0) \\ \text{subject to} & y = Ax. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0,1)$. To construct b we first generated a vector $v \in \mathbf{R}^n$, with elements

$$v_i \sim \mathcal{N}(1/n, 1/n).$$

We then generated b as $b = Av + \varepsilon$, where $\varepsilon_i \sim \mathcal{N}(0, 1/4)$. In each instance we chose $m > n$.

A.8 Portfolio optimization

Portfolio optimization or optimal asset allocation seeks to maximize the risk adjusted return of a portfolio. A common assumption is the k-factor risk model [CK93], which states that the return covariance matrix is the sum of a diagonal plus a rank k matrix. The problem can be stated as

$$\begin{array}{ll} \text{maximize} & \mu^T x - \gamma x^T (FF^T + D)x \\ \text{subject to} & x \geq 0, \quad \mathbf{1}^T x = 1 \end{array}$$

where $F \in \mathbf{R}^{n \times k}$ and D is diagonal. An equivalent graph form representation is given by

$$\begin{array}{ll} \text{minimize} & -x^T\mu + \gamma x^T D x + I(x \geq 0) + \gamma y_{1:m}^T y_{1:m} + I(y_{m+1} = 1) \\ \text{subject to} & y = \begin{bmatrix} F^T \\ \mathbf{1}^T \end{bmatrix} x. \end{array}$$

The elements of A were generated as $A_{ij} \sim \mathcal{N}(0,1)$. The diagonal of D was generated as $D_{ii} \sim U[0, \sqrt{k}]$ and the the mean return μ was generated as $\mu_i \sim \mathcal{N}(0,1)$. The risk aversion factor γ was set to 1. In each instance we chose $n > k$.

A.9 Support vector machine

The support vector machine [CV95] problem seeks a separating hyperplane classifier for a problem with two classes. The problem can be stated as

$$\text{minimize} \quad x^T x + \lambda \textstyle\sum_{i=1}^m \max(0, b_i a_i^T x + 1),$$

where $b_i \in \{-1, +1\}$ is a class label and a_i^T is the ith row of A. It has graph form representation

$$\begin{array}{ll} \text{minimize} & \lambda \sum_{i=1}^{m} \max(0, y_i + 1) + x^T x \\ \text{subject to} & y = \mathbf{diag}(b) A x. \end{array}$$

The vector b was chosen to so that the first $m/2$ elements belong to one class and the second $m/2$ belong to the other class. Specifically

$$b_i = \begin{cases} +1 & i \leq m/2 \\ -1 & \text{otherwise.} \end{cases}$$

Similarly, the elements of A were generated as

$$A_{ij} \sim \begin{cases} \mathcal{N}(+1/n, 1/n) & i \leq m/2 \\ \mathcal{N}(-1/n, 1/n) & \text{otherwise.} \end{cases}$$

This choice of A causes the rows of A to form two distinct clusters. In each instance we chose $m > n$.

References

[AHIM06] A. Ahnesjö, B. Hårdemark, U. Isacsson, and A. Montelius. The IMRT information process — Mastering the degrees of freedom in external beam therapy. *Physics in Medicine and Biology*, 51(13):R381–R402, 2006.

[AHW12] M. Annergren, A. Hansson, and B. Wahlberg. An ADMM algorithm for solving ℓ_1 regularized MPC. *arXiv preprint arXiv:1203.4070*, 2012.

[BAC11] L. M. Briceno-Arias and P. L. Combettes. A monotone + skew splitting model for composite monotone inclusions in duality. *SIAM Journal on Optimization*, 21(4):1230–1250, 2011.

[BACPP11] L. M. Briceno-Arias, P. L. Combettes, J. C. Pesquet, and N. Pustelnik. Proximal algorithms for multicomponent image recovery problems. *Journal of Mathematical Imaging and Vision*, 41(1-2):3–22, 2011.

[BEGFB94] S. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15. SIAM, 1994.

[BMOW13] S. Boyd, M. Mueller, B. O'Donoghue, and Y. Wang. Performance bounds and suboptimal policies for multi-period investment. *Foundations and Trends in Optimization*, 1(1):1–69, 2013.

[BPC+11] 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*, 3(1):1–122, 2011.

[Bra10] A. M. Bradley. *Algorithms for the equilibration of matrices and their application to limited-memory quasi-Newton methods*. PhD thesis, Stanford University, 2010.

[BTN01] A. Ben-Tal and A. Nemirovski. *Lectures on modern convex optimization: Analysis, algorithms, and engineering applications*, volume 2. SIAM, 2001.

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

[CDHR08] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate. *ACM Transactions on Mathematical Software*, 35(3):22, 2008.

[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.

[CHW+13] A. Coates, B. Huval, T. Wang, D. Wu, B. Catanzaro, and A. Y. Ng. Deep learning with COTS HPC systems. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1337–1345, 2013.

[CK93] G. Connor and R. A. Korajczyk. The arbitrage pricing theory and multifactor models of asset returns. *Handbooks in Operations Research and Management Science*, 9, 1993.

[CM87] P. H. Calamai and J. J. Moré. Projected gradient methods for linearly constrained problems. *Mathematical Programming*, 39(1):93–116, 1987.

[CMM+11] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. *International Joint Conference on Artificial Intelligence*, 22(1):1237–1242, 2011.

[COPB13] E. Chu, B. O'Donoghue, N. Parikh, and S. Boyd. A primal-dual operator splitting method for conic optimization, 2013.

[CP09] D. Chen and R. J. Plemmons. Nonnegativity constraints in numerical analysis. In A. Bultheel and R. J. Plemons, editors, *The Birth of Numerical Analysis*, pages 109–140. World Scientific, 2009.

[CP11a] 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.

[CP11b] P. L. Combettes and J. C. Pesquet. Proximal splitting methods in signal processing. In *Fixed-Point Algorithms for Inverse Problems in Science and Engineering*, pages 185–212. Springer, 2011.

[CV95] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[CW05] P. L. Combettes and V. R. Wajs. Signal recovery by proximal forward-backward splitting. *Multiscale Modeling & Simulation*, 4(4):1168–1200, 2005.

[DR56] J. Douglas and H. H. Rachford. On the numerical solution of heat conduction problems in two and three space variables. *Transactions of the American Mathematical Society*, pages 421–439, 1956.

[DY14] D. Davis and W. Yin. Convergence rate analysis of several splitting schemes. *arXiv preprint arXiv:1406.4834*, 2014.

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

[ES08] J. Eckstein and B. F. Svaiter. A family of projective splitting methods for the sum of two maximal monotone operators. *Mathematical Programming*, 111(1-2):173–199, 2008.

[GB14a] P. Giselsson and S. Boyd. Diagonal scaling in Douglas-Rachford splitting and ADMM. In *53rd IEEE Conference on Decision and Control*, 2014.

[GB14b] P. Giselsson and S. Boyd. Metric selection in Douglas-Rachford splitting and ADMM. *arXiv preprint arXiv:1410.8479*, 2014.

[GB14c] P. Giselsson and S. Boyd. Preconditioning in fast dual gradient methods. *53rd IEEE Conference on Decision and Control*, 2014.

[Gis15] P. Giselsson. Tight linear convergence rate bounds for Douglas-Rachford splitting and ADMM. *arXiv preprint arXiv:1503.00887*, 2015.

[GM75] R. Glowinski and A. Marroco. Sur l'approximation, par élḿents finis d'ordre un, et la résolution, par pénalisation-dualité d'une classe de problémes de Dirichlet non linéaires. *Mathematical Modelling and Numerical Analysis*, 9(R2):41–76, 1975.

[GOSB14] T. Goldstein, B. O'Donoghue, S. Setzer, and R. Baraniuk. Fast alternating direction optimization methods. *SIAM Journal on Imaging Sciences*, 7(3):1588–1623, 2014.

[GTSJ13] E. Ghadimi, A. Teixeira, I. Shames, and M. Johansson. Optimal parameter selection for the alternating direction method of multipliers (ADMM): quadratic problems. *IEEE Transactions on Automatic Control*, 60:644–658, 2013.

[HS52] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Joural of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[HTF09] T. Hastie, R. Tibshirani, and T. Friedman. *The elements of statistical learning*. Springer, 2009.

[Hub64] P. J. Huber. Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, 35(1):73–101, 1964.

[HYW00] B. S. He, H. Yang, and S. L. Wang. Alternating direction method with self-adaptive penalty parameters for monotone variational inequalities. *Journal of Optimization Theory and Applications*, 106(2):337–356, 2000.

[KSH12] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[LM79] P. L. Lions and B. Mercier. Splitting algorithms for the sum of two nonlinear operators. *SIAM Journal on Numerical Analysis*, 16(6):964–979, 1979.

[NCL+11] J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, Q. V. Le, and A. Y. Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning*, pages 265–272, 2011.

[NLR+15] R. Nishihara, L. Lessard, B. Recht, A. Packard, and M. I. Jordan. A general analysis of the convergence of ADMM. *arXiv preprint arXiv:1502.02009*, 2015.

[NW99] J. Nocedal and S. Wright. *Numerical Optimization*, volume 2. Springer, 1999.

[OSB13] B. O'Donoghue, G. Stathopoulos, and S. Boyd. A splitting method for optimal control. *IEEE Transactions on Control Systems Technology*, 21(6):2432–2442, 2013.

[OV14] D. O'Connor and L. Vandenberghe. Primal-dual decomposition by operator splitting and applications to image deblurring. *SIAM Journal on Imaging Sciences*, 7(3):1724–1754, 2014.

[OW06] A. Olafsson and S. Wright. Efficient schemes for robust IMRT treatment planning. *Physics in Medicine and Biology*, 51(21):5621–5642, 2006.

[PB13a] N. Parikh and S. Boyd. Block splitting for distributed optimization. *Mathematical Programming Computation*, pages 1–26, 2013.

[PB13b] N. Parikh and S. Boyd. Proximal algorithms. *Foundations and Trends in Optimization*, 1(3):123–231, 2013.

[PC11] T. Pock and A. Chambolle. Diagonal preconditioning for first order primal-dual algorithms in convex optimization. In *IEEE International Conference on Computer Vision*, pages 1762–1769, 2011.

[Pol87] B. Polyak. Introduction to optimization. *Optimization Software Inc., Publications Division, New York*, 1987.

[PP12] J. C. Pesquet and N. Pustelnik. A parallel inertial proximal optimization method. *Pacific Journal of Optimization*, 8(2):273–305, 2012.

[PS75] C. C. Paige and M. A. Saunders. Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629, 1975.

[PS82] C. C. Paige and M. A. Saunders. LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71, 1982.

[Rui01] D. Ruiz. A scaling algorithm to equilibrate both rows and columns norms in matrices. Technical report, Rutherford Appleton Laboratory, 2001. Technical Report RAL-TR-2001-034.

[Sho98] N. Z. Shor. *Nondifferentiable optimization and polynomial problems*. Kluwer Academic Publishers, 1998.

[SK67] R. Sinkhorn and P. Knopp. Concerning nonnegative matrices and doubly stochastic matrices. *Pacific Journal of Mathematics*, 21(2):343–348, 1967.

[Spi85] J. E. Spingarn. Applications of the method of partial inverses to convex programming: decomposition. *Mathematical Programming*, 32(2):199–223, 1985.

[Tib96] R. Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society*, pages 267–288, 1996.

[TTT99] K. Toh, M. J. Todd, and R. H. Tütüncü. SDPT3-a MATLAB software package for semidefinite programming, version 1.3. *Optimization Methods and Software*, 11(1-4):545–581, 1999.

[Van95] R. J. Vanderbei. Symmetric quasidefinite matrices. *SIAM Journal on Optimization*, 5(1):100–113, 1995.

[WB14] H. Wang and A. Banerjee. Bregman alternating direction method of multipliers. In *Advances in Neural Information Processing Systems*, pages 2816–2824, 2014.