POGS – Proximal Operator Graph Solver

Christopher Fougner
EE364b: Convex Optimization II Class Project

Introduction
As data sets continue to increase in size, it becomes increasingly important to design algorithms that scale with the size of the data. Typical interior point solvers are able to handle a wide range of problems and produce solutions to high accuracy. However, they do not scale to the problem sizes that one encounters in “Big Data” settings. Beyond a certain point, only parallel and distributed algorithms remain competitive. The Alternating Direction Method of Multipliers (ADMM) is a method to solve optimization problems in a distributed fashion. Although it generally doesn’t achieve the same accuracy as an interior point solver, it can be applied to much larger problems.

We have created an ADMM-based solver targeted at a class of optimization problems, for which the proximal operators of all terms in the objective are known. We implemented both CPU and GPU versions, with MATLAB wrappers.

Problem formulation – graph form
The problem we wish to solve is

\[
\min_{y \in \mathbb{R}^n} f(y) + g(y)
\]

subject to \( y = Ax \),

where \( x \in \mathbb{R}^m \) and \( y \in \mathbb{R}^n \) are the optimization variables. The terms \( A \in \mathbb{R}^{m \times n} \) and \( f, g : \mathbb{R}^n \to \mathbb{R} \) encode the problem data. We further require that \( f \) and \( g \) be convex and separable, meaning that it must be possible to express \( f \) (resp. \( g \)) as \( f(y) = \sum_{i=1}^m f_i(y_i) \), where \( y_i \in \mathbb{R} \), and \( f_i : \mathbb{R} \to \mathbb{R} \).

This formulation is known as graph form, since \( (x, y) \) are constrained to lie in the graph \( \{(x, y) \in \mathbb{R}^{m+n} : y = Ax \} \) of \( A \).

Algorithm
Using ADMM, we can formulate an iteration for solving the problem

\[
\begin{align*}
& x^{k+1} \leftarrow \text{prox}_f (y^k - \rho \nabla g) \\
& y^{k+1} \leftarrow \text{prox}_g (y^k - \rho \nabla f)
\end{align*}
\]

where \( x^k \in \mathbb{R}^m \) and \( y^k \in \mathbb{R}^n \) are dual variables. The operator \( \Pi_A \) is the orthogonal projection onto the graph of \( A \) and the proximal operator of a function \( f \) at a point \( v \) is defined as

\[
\text{prox}_f (v) = \arg\min_w \{ f(w) + \frac{\rho}{2} \| w - v \|^2 \}
\]

where \(\rho \) is a penalty parameter for deviating from \( v \). Since we require that \( f \) and \( g \) be separable, the proximal step can be done component-wise and, therefore, in parallel. The orthogonal projection consists of solving a system of equations involving either the matrix \( I + A^T A \) or \( I + A A^T \). Both matrices are independent of \( k \), and, as a result, the factorization can be computed once and cached for subsequent iterations.

Solver
We created both a CPU/OpenMP/C++ and a GPU/CUDA/C++ implementation. The interface to the solver is a single struct, which fully specifies the inputs, outputs and parameters

```
template<typename T>
struct POGSData {
  // Input
  std::vector<FunctionObj<T>> f, g;
  const T A;
  size_t m, n;
  // Output
  T *x, *y, optval;
  // Parameters
  T rho; // default: 1.0
  unsigned int max_iter; // default: 1000
  T rel_tol, abs_tol; // default: (1e-3, 1e-4)
  bool quiet; // default: true
};
```

We also added a MATLAB wrapper to simplify interfacing with the solver.

Proximal operator library
To make the solver as general as possible, each term \( \{f_1, \ldots, f_m, g_1, \ldots, g_n\} \) in the objective must be specified separately. The functions \( f_i \) and \( g_j \) are each described by a set of five parameters \( a, b, c, d, h \), which together define the function

\[
f_i(x) = c(h(x - b) + d x),
\]

where \( a, b, d \in \mathbb{R} \) and \( c \) are parameters, \( x \in \mathbb{R} \) is the variable and \( h : \mathbb{R} \to \mathbb{R} \) is one of 12 convex functions

\[
\begin{align*}
  h(x) &= \|x\|_1, & h(x) &= \max(0, x), & h(x) &= \log(1 + e^x), \\
  h(x) &= \log(x), & h(x) &= \log(1 + \exp(x)), & h(x) &= \exp(x), \\
  h(x) &= \exp(x) / x, & h(x) &= \exp(x) - 1, & h(x) &= x.
\end{align*}
\]

where \( I(x \in C) \) is the indicator function of the convex set \( C \). By knowing the proximal operator of \( h \), we can compute the proximal operator of \( f_i \), using the formula

\[
\text{prox}_f (v) = \frac{1}{a} \text{prox}_{\rho p, \rho (I \circ h)} (a(v - d/\rho) - b).
\]

The advantage of this parametric formulation is that the use of proximal operators is hidden from the user. We encapsulate these five parameters in a FunctionObj, which we define as

```
template<typename T>
struct FunctionObj {
  Function h; // enum { khbs, kIndEq0, ... } 
  T a, b, c, d; // default: (1.0, 0.0, 1.0, 0.0)
};
```

Lasso example
Consider the Lasso problem

\[
f(x) = \frac{1}{2} \| Ax - b \|^2 + \lambda \| x \|_1
\]

with \( A \in \mathbb{R}^{m \times n} \), \( b \in \mathbb{R}^m \), \( \lambda \in \mathbb{R} \). Entries of \( A \) are generated as independent samples from a normal distribution with standard deviation \( 1/n \) and \( b \) was generated as \( b = Ax + \varepsilon \), where each entry in \( \varepsilon \) was drawn from a standard normal distribution with probability 0.2 and is identical to zero with probability 0.8. The vector \( \varepsilon \) is Gaussian noise with standard deviation 1/2.

Using the MATLAB interface, this problem can be specified as

```
f.b = klsquare; 
for h = klsquare; 
  g.b = klsquare; 
  g.c = lambda; 
  [x, y, optval] = pogs(A, f, g);
```

Results
On this numerical example, \( n \) was fixed and \( m \) was varied. Default values were used for all parameters. The problem was also solved with CVX as a reference.

```
for m = 500:500:10000 
    xh = xh; 
    yh = yh; 
    % CVX code to solve Lasso
    % CVX code to solve Lasso
    % CVX code to solve Lasso
end
```

For large values of \( m \), the CPU implementation was up to 30x times faster than CVX. Additionally, for \( n = 500 \), the GPU implementation outperformed the CPU implementation by a factor of 5 to 6x. For \( n = 10,000 \), the improvement factor was between 13x to 30x.

Conclusion
We have used ADMM to create an open source solver for a large class of convex problems. The advantage of using such a solver over interior point based solvers has been shown empirically.