Implementing 2 First-Order Conic Solvers in Python

Caleb Ki & Sanjana Gupta

November 29, 2017

Background

Templates for first-order conic solvers, or TFOCS for short, is a software package that provides efficient solvers for various convex optimization problems [@beck:etal:2011]. With problems in signal processing, machine learning, and statistics in mind, the creators developed a general framework and approach for solving convex cone problems. The standard routine within the TFOCS library is a four-step process. The notation used below is taken from the paper accompanying the software package.

The first step is to express the convex optimization problem with an equivalent conic formulation. The optimization problem should follow the form:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \mathcal{A}(x) + b \in \mathcal{K} \end{aligned} \tag{1}$$

In the problem above, $x \in \mathbb{R}^n$ is the vector we are optimizing under the convex objective function f. It is important to note that f does not have to be a smooth function. $\mathcal{A}$ is a linear operator from $\mathbb{R}^n$ to $\mathbb{R}^m$, $b \in \mathbb{R}^m$ is simply a vector, and $\mathcal{K}$ is a convex cone in $\mathbb{R}^m$.

The two main hurdles to developing efficient first-order solutions for convex optimization problems of this form are that f is not restricted to the set of smooth functions and that finding feasible points under this conic constraint is often an expensive computation. These hurdles are circumvented by finding and solving the dual problem of the given convex optimization problem which leads us to the second step of the process, turning the problem into its dual form. The dual of the form given by equation 1 is as follows:

$$\begin{aligned} \underset{\lambda}{\text{maximize}} \quad & g(\lambda) \\ \text{subject to} \quad & \lambda \in \mathcal{K}^* \end{aligned} \tag{2}$$

g is simply the dual function given by $g(\lambda) = \inf_x f(x) - \sum_{i=1}^m \langle \mathcal{A}(x) + b, \lambda \rangle$, and $\mathcal{K}^* = \{\lambda \in \mathbb{R}^m : \langle \lambda, x \rangle \geq 0 \, \forall x \in \mathcal{K}\}$ is the dual cone.

The dual problem is not directly solved at this step. This is because the dual function is generally not differentiable for the class of problems we are considering. Further, directly using subgradient methods is not efficient since these methods converge very slowly. In order to convert this to a problem that can be efficiently optimized, we apply a smoothing technique which modifies the primal objective function and instead solves the following problem:

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f_\mu(x) \triangleq f(x) + \mu d(x) \\ \text{subject to} \quad & \mathcal{A}(x) + b \in \mathcal{K} \end{aligned} \tag{3}$$

Here μ is a positive scalar and $d(x)$ is a strongly convex function called the *proximity function* which satisfies $d(x) \geq d(x_0) + \frac{1}{2}\|x - x_0\|^2$ for some fixed $x_0 \in \mathbb{R}^n$. The dual of this problem is

$$\begin{aligned} \underset{\lambda}{\text{maximize}} \quad & g_\mu(\lambda) \\ \text{subject to} \quad & \lambda \in \mathcal{K}^* \end{aligned} \tag{4}$$

where g_μ is a smooth approximation of g. In many cases, the dual can be reduced to the following unconstrained problem:

$$\underset{z}{\text{maximize}} \quad -g_{sm}(z) - h(z) \tag{5}$$

This is known as the *composite form*. Here, $z \in \mathbb{R}^m$ is the optimization variable, g_{sm} is a smooth convex function and h is a non-smooth (possibly extended-value) convex function. Finally, we can efficiently solve both the smooth dual and the composite problems using optimal first order methods such as gradient descent.

For both the problems (4), (5), given a sequence of step sizes $\{t_k\}$, the optimization begins with a point $\lambda_0 \in \mathcal{K}^*$ and has the following updating rule respectively:

$$\lambda_{k+1} \leftarrow \underset{\lambda \in \mathcal{K}^*}{\arg\min} \|\lambda_k + t_k \nabla g_\mu(\lambda_k) - \lambda\|_2$$

$$z_{k+1} \leftarrow \arg\min_z \; g_{sm}(z_k) + \langle \nabla g_{sm}(z_k), z - z_k \rangle + \frac{1}{2t_k}\|z - z_k\|^2 + h(z)$$

The approximate primal solution can then be recovered from the optimal value of λ as follows:

$$x(\lambda) \triangleq \arg\min_x \; f(x) + \mu d(x) - \langle \mathcal{A}(x) + b, \lambda \rangle \tag{6}$$

Equation 6 shows that the fundamental computational primitive in this method is the efficient minimization of the sum of a linear term, a proximity function, and a non-smooth function.

Implementation

As stated previously, TFOCS, when given the correct conic formulation of an optimization problem, will apply various first-order solvers to the smoothed dual problem. The goal of this project is to port 2 of the 6 available first-order solvers in TFOCS software package from Matlab into Python. Specifically, we will be implementing the default method in TFOCS, the single-projection method developed by @ausl:tebo:2006, as well as the dual-projection method developed by @lan:etal:2011. Furthermore, we would like our solvers to accept a pythonic version of the calling sequence found in the user manual written by @beck:etal:2014, `[x, out] = tfocs(smoothF, gradF, projectorF, x0m tol, gamma)`. Given a smooth function `smoothF`, the gradient of the smooth function `gradF`, a nonsmooth function `projectorF`, and an initial point `x0`, the solver would return the optimal vector `x` and the primal solution

`out`. For the purposes of this project, we are starting with the assumption that the user has expressed the optimization problem of concern in the correct specific conic form. As stated earlier, once the problem has been wrangled into the correct form, they can be solved through first-order methods.

Motivation

The first-order solvers we implement assume that the optimization problem has been massaged into the following unconstrained form:

$$\text{minimize } \phi(z) \triangleq g(z) + h(z)$$

which is simply problem 5 except we have flipped the signs and turned the maximization problem into a minimization problem. To be notationally consistent we fix the update rule for z_{k+1} provided in the background section to

$$z_{k+1} \leftarrow \arg\min\ g(z_k) + \langle \nabla g(z_k), z - z_k \rangle + \frac{1}{2t_k}\|z - z_k\|^2 + h(z), \tag{7}$$

where t_k is the step size. @beck:etal:2011 assert that to ensure global convergence the following inequality must be satisfied:

$$\phi(z_{k+1}) \leq g(z_k) + \langle \nabla g(z_k), z_{k+1} - z_k \rangle + \frac{1}{2t_k}\|z_{k+1} - z_k\|^2 + h(z_{k+1}). \tag{8}$$

If we assume that the gradient of g satisfies,

$$\|\nabla g(x) - \nabla g(y)\|_* \leq L\|x - y\|\ \forall x, y \in \text{dom}\phi$$

(i.e., it satifies a generalized Lipschitz criterion), then convergence condition 8 holds for all $t_k \leq L^{-1}$.

Implementation

The repeated calls to a generalized projection as described above manifests itself in the algorithm below. We are assuming that the user has provided the smooth function g and its gradient, the nonsmooth function h and its prox function, a tolerance level for convergence, and a starting point x_0.

Lan, Lu, and Monteiro's modification of Nesterove's 2007 algorithm follows the same algorithm except we replace *line 8* with the following call:

$$z_{k+1} \leftarrow \arg\min_z \langle \nabla g(y_k), z \rangle + \frac{1}{2}L_k\|z - y_k\|^2 + h(z).$$

The key difference between these two similar variants is that, Lan, Lu, and Monteiro's method requires 2 projections where Auslender and Teboulle's only requires 1. Of course, two projections per iteration is more computationally taxing, so we only prefer the two projection method in the case that it can reduce the number of iterations significantly.

Algorithm 1 Auslender and Teboulle's Algorithm

Require: $z_0 \in \text{dom}\theta$, $\alpha \in (0,1]$, $\beta \in (0,1)$, γ, tol > 0

1: $\bar{z}_0 \leftarrow z_0$, $\theta_{-1} \leftarrow 1$, $L_{-1} \leftarrow 1$
2: **for** $k = 0, 1, 2, \ldots$ **do**
3: $\quad L_k = \alpha L_{k-1}$
4: $\quad$ **loop**
5: $\quad\quad \theta_k \leftarrow 2/(1 + (1 + 4L_k/\theta_{k-1}^2 L_{k-1})^{1/2})$
6: $\quad\quad y_k \leftarrow (1-\theta_k)z_k + \theta_k \bar{z}_k$
7: $\quad\quad \bar{z}_{k+1} \leftarrow \arg\min_z \langle \nabla g(y_k), z\rangle + \frac{1}{2}\theta_k L_k \|z - \bar{z}_k\|^2 + h(z)$
8: $\quad\quad z_{k+1} \leftarrow (1-\theta_k)z_k + \theta_k \bar{z}_{k+1}$
9: $\quad\quad$ **if** $g(y_k) - g(z_{k+1}) \geq \gamma g(z_{k+1})$ **then**
10: $\quad\quad\quad \hat{L} \leftarrow 2(g(z_{k+1}) - g(y_k) - \langle \nabla g(y_k), z_{k+1} - y_k\rangle)/\|z_{k+1} - y_k\|_2^2$
11: $\quad\quad$ **else**
12: $\quad\quad\quad \hat{L} \leftarrow 2|\langle y_k - z_{k+1}, \nabla g(z_{k+1}) - \nabla g(y_k)\rangle|/\|z_{k+1} - y_k\|_2^2$
13: $\quad\quad$ **end if**
14: $\quad\quad$ **if** $L_k \geq \hat{L}$ **then break**
15: $\quad\quad$ **end if**
16: $\quad\quad L_k \leftarrow \max\{L_k/\beta, \hat{L}\}$
17: $\quad$ **end loop**
18: $\quad$ **if** $\|z_k - z_{k-1}\|/\max\{1, \|z_k\|\} \leq$ tol **then break**
19: $\quad$ **end if**
20: **end for**

Justification

In the following section, we go through several lines in our algorithm to justify and clairfy what the solver is actually doing. We begin with a discussion about step size (*lines 3,9-16*). Generally it's very difficult to calculate L, and the step size $\frac{1}{L}$ is often too conservative. While the performance can be improved by reducing L, reducing L too much can cause the algorithm to diverge. All these problems are simultaneously resolved by using *backtracking*. Backtracking is a technique used to find solutions to optimization problems by building partial candidates to the solution called *backtracks*. Each backtrack is dropped as soon as the algorithm realizes that it can not be extended to a valid solution. Applying this technique to the Lipschitz constant in our problem, we estimate the global constant L by L_k which preserves convergence if the following inequality holds:

$$g(z_{k+1}) \leq g(y_k) + \langle \nabla g(y_k), z_{k+1} - y_k\rangle + \frac{1}{2}L_k\|z_{k+1} - y_k\|^2. \tag{9}$$

It is observed that if $g(z_{k+1})$ is very close to $g(y_k)$, equation (9) suffers from severe cancellation errors. Generally, cancellation errors occur if $g(y_k) - g(z_{k+1}) < \gamma g(z_{k+1})$ where the threshold for γ is around 10^{-8} to 10^{-6}. To be safe, we consider $\gamma = 10^{-4}$. In this case, we use the following inequality to ensure convergence:

$$|\langle y_k - z_{k+1}, \nabla g(z_{k+1}) - \nabla g(y_k)\rangle| \leq \frac{1}{2}L_k\|z_{k+1} - y_k\|^2. \tag{10}$$

Note that the above inequalities (9), (10) automatically hold for $L_k \geq L$. Let's consider what's going on in iteration k. If $L_k \geq L$ then the relevant inequality automatically holds and we do not update L_k further (*i.e.* $L_k = \alpha L_{k-1}$). If however $L_k < L$ we simply increase L_k by

a factor of $\frac{1}{\beta}$ to obtain $\frac{L_k}{\beta}$ (for fixed $\beta \in (0,1)$) which eventually results in $L_k \geq L$. Thus, backtracking preserves global convergence. To combine both the above cases in one step, we introduce $\hat{L}$ which is the smallest value of L_k that satisfies the relevant inequality (9) or (10) at the k^{th} iteration. L_k is then updated as $\max\{\frac{L_k}{\beta}, \hat{L}\}$. Here, $\hat{L}$ is obtained by changing the inequalities (9), (10) to equalities and solving for L_k.

In every iteration we try to reduce the value of the Lipschitz estimate L_k (*line 3*). This is done to improve the performance of the algorithm and is achieved by updating $L_k = \alpha L_{k-1}$ for some fixed $\alpha \in (0,1]$. Reducing L_k at each iteration can of course lead to an increased number of backtracks which we try to minimize by picking an appropriate value of α. For these kinds of solvers, generally $\alpha = 0.9$ is used which is what we have set as the default method.

Intertwined with updating L_k is the problem of updating θ_k (*line 5*). First we establish bounds on the prediction error at the $(k+1)$st iteration as follows:

$$\phi(z_{k+1}) - \phi(z^\star) \leq \frac{1}{2} L {\theta_k}^2 \|z_0 - z^\star\|^2 \leq 2\frac{L}{k^2}\|z_0 - z^\star\|^2 \tag{11}$$

This shows that the error bound is directly proportional to $L_k {\theta_k}^2$. In a simple backtracking step, as we increase L_k (by at least a factor of $1/\beta$), the error bound increases too. This can be avoided by updating θ_k along with L_k in each iteration by using the following inequality which ensures that convergence is preserved:

$$\frac{L_{k+1}{\theta_{k+1}}^2}{1-\theta_{k+1}} \geq L_k {\theta_k}^2 \tag{12}$$

Solving for θ_{k+1} gives the update as in *line 5* of algorithm 1.

The algorithm states in *line 7* that to update $\bar{z}_k$ we must find the arg min of some function of the gradient of g and nonsmooth function h. This can be simply reduced to the proximity function for h with step size $\frac{1}{L\theta_k}$ evaluated at $\bar{z}_k - \frac{\nabla g(y_k)}{L\theta_k}$ (as stated before, we are assuming that the user is providing this prox function). The proximity operator *prox* of a convex function h at x is defined as the unique solution to the following:

$$\text{prox}_{t,h}(x) = \arg\min_z \; h(y) + \frac{1}{2t}\|x - z\|^2 \tag{13}$$

Here t is the step size. The update in *line 7* is then equivalent to a proximity function as follows:

$$
\begin{aligned}
&\arg\min_z \ \langle \nabla g(y_k), z\rangle + \frac{L\theta_k}{2}\|z-\bar{z}_k\|^2 + h(z) \\
&= \arg\min_z \ h(z) + \frac{L\theta_k}{2}\left(\frac{2}{L\theta_k}\langle \nabla g(y_k), z\rangle + \|z\|^2 - 2\langle \bar{z}_k, z\rangle + \|\bar{z}_k\|^2\right) \\
&= \arg\min_z \ h(z) + \frac{L\theta_k}{2}\left(2\langle \frac{\nabla g(y_k)}{L\theta_k} - \bar{z}_k, z\rangle + \|z\|^2 + \|\bar{z}_k\|^2\right) \\
&= \arg\min_z \ h(z) + \frac{L\theta_k}{2}\left(2\langle \frac{\nabla g(y_k)}{L\theta_k} - y_k, z\rangle + \|z\|^2\right) + \frac{L\theta_k}{2}\|y_k\|^2 \\
&= \arg\min_z \ h(z) + \frac{L\theta_k}{2}\left(\|\frac{\nabla g(y_k)}{L\theta_k} - \bar{z}_k + z\|^2 - \|\frac{\nabla g(y_k)}{L\theta_k} - y_k\|^2\right) + \frac{L\theta_k}{2}\|\bar{z}_k\|^2 \\
&= \arg\min_z \ h(z) + \frac{L\theta_k}{2}\|\bar{z}_k - \frac{\nabla g(y_k)}{L\theta_k} - z\|^2 \\
&= \text{prox}_{\frac{1}{L\theta_k},h}\left(\bar{z}_k - \frac{\nabla g(y_k)}{L\theta_k}\right)
\end{aligned}
$$

This clearly holds follows because the term $\frac{L\theta_k}{2}\|\bar{z}_k\|^2 - \frac{L\theta_k}{2}\|\frac{\nabla g(y_k)}{L\theta_k} - y_k\|^2$ being independent of z does not affect the minimization and can be dropped to obtained the proximity function as in the last line.

For the Lan, Lu, and Monteiro method, the update for z_k in *line 8* can be computed similarly using the proximity function as follows:

$$
\begin{aligned}
&\arg\min_z \ \langle \nabla g(y_k), z\rangle + \frac{L}{2}\|z-y_k\|^2 + h(z) \\
&= \arg\min_z \ h(z) + \frac{L}{2}\left(\frac{2}{L}\langle \nabla g(y_k), z\rangle + \|z\|^2 - 2\langle y_k, z\rangle + \|y_k\|^2\right) \\
&= \arg\min_z \ h(z) + \frac{L}{2}\left(2\langle \frac{\nabla g(y_k)}{L} - y_k, z\rangle + \|z\|^2 + \|y_k\|^2\right) \\
&= \arg\min_z \ h(z) + \frac{L}{2}\left(2\langle \frac{\nabla g(y_k)}{L} - y_k, z\rangle + \|z\|^2\right) + \frac{L}{2}\|y_k\|^2 \\
&= \arg\min_z \ h(z) + \frac{L}{2}\left(\|\frac{g(y_k)}{L} - y_k + z\|^2 - \|\frac{\nabla g(y_k)}{L} - y_k\|^2\right) + \frac{L}{2}\|y_k\|^2 \\
&= \arg\min_z \ h(z) + \frac{L}{2}\|y_k - \frac{\nabla g(y_k)}{L} - z\|^2 \\
&= \text{prox}_{\frac{1}{L},h}\left(y_k - \frac{\nabla g(y_k)}{L}\right)
\end{aligned}
$$

Similar to earlier, the last line clearly holds since $\frac{L}{2}\|y_k\|^2 - \frac{L}{2}\|\frac{\nabla g(y_k)}{L} - y_k\|^2$ is independent of z (i.e., we are able to treat it has a constant), and thus it does not affect the optimization. Therefore, we can drop this term to get the proximity function in the last line.

Line 17 of the algorithm states that if the distance between sequential z_k's becomes sufficiently small (i.e., less than the given tolerance) then the algorithm has converged and we have found

our optimal z. Usually, tolerance is taken to be 10^{-8}.

Future Work

We have yet to complete the source code for our 2 first-order solvers. For the final submission, we will of course finish our implementation and include a couple of examples using our implementation. We plan to replicate some of the results found the paper accompanying TFOCS namely the Dantzig Selector problem.

References