Black-box Optimization with a Politician

Sébastien Bubeck SEBUBECK@MICROSOFT.COM
Microsoft Research

Yin-Tat Lee YINTAT@MIT.EDU
MIT

Abstract

We propose a new framework for black-box convex optimization which is well-suited for situations where gradient computations are expensive. We derive a new method for this framework which leverages several concepts from convex optimization, from standard first-order methods (e.g. gradient descent or quasi-Newton methods) to analytical centers (i.e. minimizers of self-concordant barriers). We demonstrate empirically that our new technique compares favorably with state of the art algorithms (such as BFGS).

1. Introduction

In standard black-box convex optimization (Nemirovski and Yudin, 1983; Nesterov, 2004; Bubeck, 2015) first-order methods interact with an *oracle*: given a query point x, the oracle reports the value and gradient of the underlying objective function f at x. In this paper we propose to replace the oracle by a *politician*. Instead of answering the original query x the politician changes the question and answers a new query y which is guaranteed to be better than the original query x in the sense that $f(y) \leq f(x)$. The newly selected query y also depends on the history of queries that were made to the politician. Formally we introduce the following definition (for sake of simplicty we write $\nabla f(x)$ for either a gradient or a subgradient of f at x).

Definition 1 *Let $\mathcal{X} \subset \mathbb{R}^n$ and $f : \mathcal{X} \to \mathbb{R}$. A politician Φ for f is a mapping from $\mathcal{X} \times \cup_{k=0}^{\infty}(\mathcal{X} \times \mathbb{R} \times \mathbb{R}^n)^k$ to $\mathcal{X}$ such that for any $k \geq 0, x \in \mathcal{X}, h \in (\mathcal{X} \times \mathbb{R} \times \mathbb{R}^n)^k$ one has $f(\Phi(x,h)) \leq f(x)$. Furthermore when queried at x with history h a politician for f also output $f(\Phi(x,h))$ and $\nabla f(\Phi(x,h))$ (in order to not overload notation we do not include these outputs in the range of Φ).*

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

Let us clarify the interaction of a first-order method with a politician. Note that we refer to the couple (first-order method, politician) as the *algorithm*. Let $M : \cup_{k=0}^{\infty}(\mathcal{X} \times \mathbb{R} \times \mathbb{R}^n)^k \to \mathcal{X}$ be a first-order method and Φ a politician for some function $f : \mathcal{X} \to \mathbb{R}$. The course of the algorithm (M, Φ) then goes as follows: at iteration $k+1$ one first calculates the method's query point $x_{k+1} = M(h_k)$ (with $h_0 = \emptyset$), then one calculates the politician's new query point $y_{k+1} = \Phi(x_{k+1}, h_k)$ and the first order information at this point $(f(y_{k+1}), \nabla f(y_{k+1}))$, and finally one updates the history with this new information $h_{k+1} = (h_k, (y_{k+1}, f(y_{k+1}), \nabla f(y_{k+1})))$. Note that a standard oracle simply corresponds to a politician $\mathcal{O}$ for f such that $\mathcal{O}(x,h) = (x, f(x), \nabla f(x))$ (in particular the algorithm $(M, \mathcal{O})$ is the usual algorithm corresponding to the first-order method M).

The philosophy of the above definition is that it gives in some sense an automatic way to combine different optimization algorithms. Say for example that we wish to combine the ellipsoid method with gradient descent. One way to do so is to design an "ellipsoidal politician": the politician keeps track of a feasible ellipsoidal region based on the previously computed gradients, and when asked with the query x the politician chooses as a new query y the result of a line-search on the line between x and the center of current ellipsoid. Gradient descent with this ellipsoidal politician would then replace the step $x \leftarrow x - \eta \nabla f(x)$ by $x \leftarrow y - \eta \nabla f(y)$. The hope is that in practice such a combination would integrate the fast incremental progress of gradient descent with the geometrical progress of the ellipsoid method.

In this paper we focus on unconstrained convex optimization. We are particularly interested in situations where calculating a (sub)gradient has superlinear complexity (i.e., $\gg n$) such as in logistic regression and semidefinite programming. In such cases it is natural to try to make the most out of the computed gradients by incorporating geometric reasoning (such as in the ellipsoid method). We do so by introducing the *geometric politician* (Section 3), which is based on a combination of the recent ideas of

(Bubeck et al., 2015) with standard cutting plane/interior point methods machinery (through the notion of a "center" of a set, see Section 4). For a given first order method M, we denote by $M+$ the algorithm obtained by running M with the geometric politician. We demonstrate empirically (Section 5) the effectiveness of the geometric politician on various standard first-order methods for convex optimization (gradient descent, Nesterov's accelerated gradient descent, non-linear conjugate gradient, BFGS). In particular we show that BFGS+ is a surprisingly robust and parameter-free algorithm with state of the art performance across a wide range of problems (both smooth and non-smooth).

2. Affine invariant politician

As mentioned above we assume that the complexity of computing the map $x \mapsto \nabla f(x)$ is superlinear. This implies that we can afford to have a politician such that the complexity of computing the map $(x, h) \mapsto \Phi(x, h)$ is $O(n \times \text{poly}(k))$ (we think of the number of iterations k as typically much smaller than the dimension n). We show in this section that this condition is (essentially) automatically satisfied as long as the politician is affine invariant in the following sense (we use a slight abuse of language and refer to a map $f \mapsto \Phi_f$, where Φ_f is a politician for f, as a politician):

Definition 2 *A politician $f \mapsto \Phi_f$ is called affine invariant if for any function f and any affine map $T : \mathbb{R}^m \to \mathbb{R}^n$ such that $T(x) = z + Lx$ for some matrix L, $k \geq 0, x \in \mathbb{R}^m, (y_i, v_i, g_i) \in \mathbb{R}^m \times \mathbb{R} \times \mathbb{R}^n$, one has*

$$T(\Phi_{f\circ T}(x, (y_i, v_i, L^\top g_i)_{i\in[k]})) = \Phi_f(T(x), (T(y_i), v_i, g_i)_{i\in[k]}).$$

We say that an affine invariant politician has cost $\psi : \mathbb{N} \to \mathbb{N}$ if for any $f : \mathbb{R}^k \to \mathbb{R}$ the map $(x, h) \in \mathbb{R}^k \times (\mathbb{R}^k \times \mathbb{R} \times \mathbb{R}^k)^k \mapsto \Phi_f(x, h)$ can be computed in time $\psi(k)$.

Proposition 1 *Let Φ be an affine invariant politician with cost ψ. Then for any $f : \mathbb{R}^n \to \mathbb{R}$, $(y_i, v_i, g_i) \in \mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n, i \in [k]$ and $x, y_i \in y_1 + \text{Span}(g_1, \ldots, g_k)$ one can compute $\Phi_f(x, (y_i, v_i, g_i)_{i\in[k]}) \in \mathbb{R}^n$ in time $\psi(k) + O(nk^2)$.*

Proof Let G be the $n \times k$ matrix with i^{th} column given by g_i. We consider the QR decomposition of G which can be computed in time $O(nk^2)$, that is Q is an $n \times k$ matrix and R a $k \times k$ matrix such that $G = QR$ and $Q^\top Q = \text{I}_k$. Let T be the affine map defined by $T = y_1 + Q$. Note that since $x \in y_1 + \text{Span}(g_1, \ldots, g_k)$ one has $x = T(Q^\top(x - y_1))$ (and similarly for y_i). Thus by affine invariance one has

$$\begin{aligned}&\Phi_f(x, (y_i, v_i, g_i))\\ &= \Phi_f(T(Q^\top(x - y_1)), (T(Q^\top(y_i - y_1)), v_i, g_i))\\ &= y_1 + Q\Phi_{f\circ T}(Q^\top(x - y_1), (Q^\top(y_i - y_1), v_i, R_i)),\end{aligned}$$

where R_i is the i^{th} column of R. Furthermore by definition of the cost ψ and since $f \circ T$ is defined on $\mathbb{R}^k$ we see that this last quantity can be computed in time $\psi(k) + O(nk^2)$, thus concluding the proof. ■

The above proposition shows that with an affine invariant politician and a first order method M verifying for any $(y_i, v_i, g_i)_{i\in[k]} \in (\mathbb{R}^n \times \mathbb{R} \times \mathbb{R}^n)^k$,

$$M((y_i, v_i, g_i)_{i\in[k]}) \in y_1 + \text{Span}(y_1, \ldots, y_k, g_1, \ldots, g_k),$$

one can run k steps of the corresponding algorithm in time $O(nk^2 + k\psi(k))$ plus the time to compute the k function values and gradients of the underlying function f to be optimized. Note that one gets a time of $O(nk^2)$ instead of $O(nk^3)$ as one can store the QR decomposition from one step to the next, and updating the decomposition only cost $O(nk)$.

3. Geometric politician

We describe in this section the *geometric politician* which is based on ideas developed in (Bubeck et al., 2015). A key observation in the latter paper is that if f is a α-strongly convex function minimized at x^* then one has for any x,

$$\left\|x^* - x - \frac{1}{\alpha}\nabla f(x)\right\|^2 \leq \frac{\|\nabla f(x)\|^2}{\alpha^2} - \frac{2}{\alpha}\left(f(x) - f(x^*)\right).$$

This motivates the following definition:

$$\text{B}(x, \alpha, \mathbf{fval}) := \left\{z \in \mathbb{R}^n : \left\|z - x - \frac{1}{\alpha}\nabla f(x)\right\|^2 \leq \frac{\|\nabla f(x)\|^2}{\alpha^2} - \frac{2}{\alpha}\left(f(x) - \mathbf{fval}\right)\right\}.$$

In particular given the first order information at $y_1, \ldots, y_k$ one knows that the optimum x^* lies in the region $R_k \subset \mathbb{R}^n$ defined by

$$R_k = \bigcap_{i\in[k]} B(y_i, \alpha, \mathbf{fval}) \text{ where } \mathbf{fval} = \min_{i\in[k]} f(y_i). \quad (1)$$

Now suppose that given this first order information at $y_1, \ldots, y_k$ the first order method asks to query x. How should we modify this query in order to take into account the geometric information that $x^* \in R_k$? First observe

that for any z, $\mathrm{B}(z, \alpha, \mathbf{fval})$ is contained in a halfspace that has z on its boundary (in the limiting case $\alpha \to 0$ the set $\mathrm{B}(z, \alpha, f(z))$ is exactly a halfspace). In particular if the next query point y_{k+1} is the center of gravity of R_k then we have that the volume of R_{k+1} is at most $1 - 1/e$ times the volume of R_k (see (Grünbaum, 1960)), thus leading to an exponential convergence rate. However the region R_k can be very large initially, and the center of gravity might have a large function value and gradient, which means that R_k would be intersected with a large sphere (possibly so large that it is close to a halfspace). On the other hand the first order method recommends to query x, which we can think of as a local improvement of y_k, which should lead to a much smaller sphere. The issue is that the position of this sphere might be such that the intersection with R_k is almost as large as the sphere itself. In order to balance between the geometric and function value/gradient considerations we propose for the new query to do a line search between the center of R_k and the recommended query x. The geometric politician follows this recipe with two important modifications: (i) there are many choices of centers that would guarantee an exponential convergence rate while being much easier to compute than the center of gravity, and we choose here to consider the *volumetric center*, see Section 4 for the definition and more details about this notion; (ii) we use a simple heuristic to adapt online the strong convexity parameter α, namely we start with some large value for α and if it happens that the feasible region R_k is empty then we know that α was too large, in which case we reduce it. We can now describe formally the geometric politician, see Algorithm 1. Importantly one can verify that the geometric politician is affine invariant and thus can be implemented efficiently (see the proof of Proposition 1).

Algorithm 1: Geometric Politician

Parameter: An upper bound on the strong convexity parameter α. (Can be $+\infty$.)
Input: Query x, past queries and the corresponding first order information $(y_i, f(y_i), \nabla f(y_i))_{i \in [k]}$.
Let $\mathbf{fval} = \min_{i \in [k]} f(y_i)$ and the feasible region $R_k(\alpha) = \bigcap_{i \in [k]} \mathrm{B}(y_i, \alpha, \mathbf{fval})$.
if $R_k(\alpha) = \emptyset$ **then**
  Let α be the largest number such that $R_k(\alpha) \neq \emptyset$.
  $\alpha \leftarrow \alpha / 4$.
end
Let $y_{k+1} = \mathrm{argmin}_{y \in \{tx + (1-t)c(R_k(\alpha)), t \in \mathbb{R}\}} f(y)$ where $c(R_k(\alpha))$ is the volumetric center of $R_k(\alpha)$ (see Section 4).
Output: y_{k+1}, $f(y_{k+1})$ and $\nabla f(y_{k+1})$.

4. Volumetric center

The volumetric barrier for a polytope was introduced in (Vaidya, 1996) to construct an algorithm with both the oracle complexity of the center of gravity method and the computational complexity of the ellipsoid method (see [Section 2.3, (Bubeck, 2015)] for more details and (Lee et al., 2015) for recent advances on this construction). Recalling that the standard logarithmic barrier F_P for the polytope $P = \{x \in \mathbb{R}^n : a_i^\top x < b_i, i \in [m]\}$ is defined by

$$F_P(x) = -\sum_{i=1}^m \log(b_i - a_i^\top x),$$

one defines the volumetric barrier v_P for P by

$$v_P(x) = \mathrm{logdet}(\nabla^2 F_P(x)).$$

The volumetric center $c(P)$ is then defined as the minimizer of v_P. In the context of the geometric politician (see Algorithm 1) we are dealing with an intersection of balls rather than an intersection of halfspaces. More precisely the region of interest is of the form:

$$R = \bigcap_{i=1}^k \left\{x \in \mathbb{R}^n : \|x - c_i\| \leq r_i\right\}.$$

For such a domain the natural self-concordant barrier to consider is:

$$F_R(x) = -\frac{1}{2} \sum_{i=1}^k \log\left(r_i^2 - \|x - c_i\|^2\right).$$

The volumetric barrier is defined as before by

$$v_R(x) = \mathrm{logdet}(\nabla^2 F_R(x)),$$

and the volumetric center of R is the minimizer of v_R. It is shown in (Anstreicher, 2004) that v_R is a self-concordant barrier which means that the center can be updated (when a new ball is added to R) via few iterations of Newton's method. Often in practice, it takes less than 5 iterations to update the minimizer of a self-concordant barrier (Goffin and Vial, 1999; Bahn et al., 1995) when we add a new constraint. Hence, the complexity merely depends on how fast we can compute the gradient and Hessian of F_R and v_R.

Proposition 2 *For the analytic barrier F_R, we have that*

$$\nabla F_R(x) = A^\top 1_{k \times 1},$$
$$\nabla^2 F_R(x) = 2A^\top A + \lambda^{(1)} I$$

where d is a vector defined by $\left(r_i^2 - \|x - c_i\|^2\right)^{-1}$, A is a $k \times n$ matrix with i^{th} row given by $d_i(x)(x - c_i)$, $\lambda^{(p)} = \sum_{i \in [k]} d_i^p(x)$ and $1_{k \times 1}$ is a $k \times 1$ matrix with all entries being 1.

For the volumetric center, we have that

$$\nabla v_R(x) = \left(\left(2\mathtt{tr}H^{-1}\right)\mathrm{I} + 4H^{-1}\right)A^\top d + 8A^\top\sigma,$$

$$\begin{aligned}\nabla^2 v_R(x) = {} & 48A^\top\Sigma A - 64A^\top\left(AH^{-1}A^\top\right)^{(2)}A \\ & + \left(8\mathtt{tr}(D\Sigma) + 2\lambda^{(2)}\mathtt{tr}(H^{-1})\right)I + 4\lambda^{(2)}H^{-1} \\ & + 8\mathtt{tr}(H^{-1})A^\top DA + 16\mathtt{sym}\left(A^\top DAH^{-1}\right) \\ & - 4\mathtt{tr}(H^{-2})A^\top DJDA - 8H^{-1}A^\top DJDAH^{-1} \\ & - 8\mathtt{sym}(A^\top DJDAH^{-2}) - 8\left(d^\top AH^{-1}A^\top d\right)H^{-1} \\ & - 16\mathtt{sym}\left(A^\top\mathtt{diag}\left(AH^{-2}A^\top\right)JDA\right) \\ & - 32\mathtt{sym}\left(A^\top\mathtt{diag}(AH^{-1}A^\top d)AH^{-1}\right)\end{aligned}$$

where $H = \nabla^2 F_R(x)$, $\sigma_i = e_i^\top AH^{-1}A^\top e_i$, e_i is the indicator vector with i^{th} coordinate, J is a $k\times k$ matrix with all entries being 1, $\mathtt{sym}(B) = B + B^\top$, $\mathtt{diag}(v)$ is a diagonal matrix with $\mathtt{diag}(v)_{ii} = v_i$, $D = \mathtt{diag}(d)$, $\Sigma = \mathtt{diag}(\sigma)$, and $B^{(2)}$ is the Schur square of B defined by $B^{(2)}_{ij} = B^2_{ij}$.

The above proposition shows that one step of Newton method for analytic center requires 1 dense matrix multiplication and solving 1 linear system; and for volumetric center, it requires 5 dense matrix multiplications, 1 matrix inversion and solving 1 linear system if implemented correctly. Although the analytic center is a more popular choice for "geometrical" algorithms, we choose volumetric center here because it gives a better convergence rate (Vaidya, 1996; Atkinson and Vaidya, 1995) and the extra cost $\psi(k)$ is negligible to the cost of updating QR decomposition nk.

5. Experiments

In this section, we compare the geometric politician against two libraries for first order methods, minFunc (Schmidt, 2012) and TFOCS (Becker et al., 2011). Both are popular MATLAB libraries for minimizing general smooth convex functions. Since the focus of this paper is all about how to find a good step direction using a politician, we use the exact line search (up to machine accuracy) whenever possible. This eliminates the effect of different line searches and reduces the number of algorithms we need to test. TFOCS is the only algorithm we use which does not use line search because they do not provide such option. To compensate on the unfairness to TFOCS, we note that the algorithm TFOCS uses is accelerated gradient descent and hence we implement the Gonzaga-Karas's accelerated gradient descent (Gonzaga and Karas, 2013), which is specifically designed to be used with exact line search. Another reason we pick this variant of accelerated gradient descent is because we found it to be the fastest variant of accelerated gradient descent (excluding the geometric descent of (Bubeck et al., 2015)) for our tested data (Gonzaga and Karas also observed that on their own dataset).

The algorithms to be tested are the following:

- [SD] Steepest descent algorithm in minFunc.
- [Nes] Accelerated gradient descent, General Scheme 2.2.6 in (Nesterov, 2004).
- [TFOCS] Accelerated gradient descent in TFOCS.
- [GK] Gonzaga-Karas's of Accelerated Gradient Descent (Sec 5.1).
- [Geo] Geometric Descent (Bubeck et al., 2015).
- [CG] Non-Linear Conjugate Gradient in minFunc.
- [BFGS] Broyden–Fletcher–Goldfarb–Shanno algorithm in minFunc.
- [PCG] Preconditioned Non-Linear Conjugate Gradient in minFunc.
- [$\emptyset$+] Geometric Politician itself (Sec 5.1).
- [GK+] Using GK with Geometric Oracle (Sec 5.1).
- [BFGS+] Using BFGS with Geometric Oracle (Sec 5.1).

We only tested the geometric oracle on GK and BFGS because they are respectively the best algorithms in theory and practice on our tested data. The $\emptyset+$ algorithm is used as the control group to test if the geometric politician by itself is sufficient to achieve good convergence rate. We note that all algorithms except Nes are parameter free; each step of SD, Nes, TFOCS, GK, Geo, CG takes $O(n)$ time and each step of BFGS, PCG, $\emptyset+$, GK+ and BFGS+ takes roughly $O(nk)$ time for k^{th} iteration.

5.1. Details of Implementations

The first algorithm we implement is the $\emptyset+$ algorithm which simply repeatedly call the politician. As we will see, this algorithm is great for non smooth problems but not competitive for smooth problems.

Algorithm 2: $\emptyset+$

Input: x_0.
for $k \leftarrow 1, 2, \cdots$ **do**
  Set $x_{k+1} \leftarrow \Phi_f(x_k, (x_i, f(x_i), \nabla f(x_i))_{i\in[k]})$.
end

The second algorithm we implement is the accelerated gradient descent proposed by Gonzaga and Karas (Gonzaga and Karas, 2013). This algorithm uses line search to learn

Algorithm 3: Gonzaga-Karas's variant of Accelerated Gradient Descent

Input: x_1.
$\gamma = 2\alpha$, $v_0 = x_0$ and $y_0 = x_0$.
for $k \leftarrow 1, 2, \cdots$ **do**
  $y_k \leftarrow \Phi_f(y_{k-1})$.
  $x_{k+1} = \texttt{line\_search}(y_k, -\nabla f(y_k))$.
  if $\alpha \geq \gamma/1.02$ *and we are using first order oracle* **then** $\alpha = \gamma/2$. (*)
  if $\alpha \geq \frac{\|\nabla f(y_k)\|^2}{2(f(y_k)-f(x_{k+1}))}$ **then**
    $\alpha = \frac{\|\nabla f(y_k)\|^2}{20(f(y_k)-f(x_{k+1}))}$.
  $G = \gamma\left(\frac{\alpha}{2}\|v_k - y_k\|^2 + \langle \nabla f(y_k), v_k - y_k\rangle\right)$.
  $A = G + \frac{1}{2}\|\nabla f(y_k)\|^2 + (\alpha-\gamma)(f(x_k) - f(y_k))$.
  $B = (\alpha-\gamma)(f(x_{k+1}) - f(x_k)) - \gamma(f(y_k) - f(x_k)) - G$.
  $C = \gamma(f(x_{k+1}) - f(x_k))$.
  $\beta = \frac{-B+\sqrt{B^2-4AC}}{2A}$, $\gamma = (1-\beta)\gamma + \beta\alpha$.
  $v_{k+1} = \frac{1}{\gamma}((1-\beta)\gamma v_k + \beta(\alpha y_k - \nabla f(y_k))$.
end

the the smoothness parameter and strong convexity parameter, see Algorithm 3. We disable the line (*) in the algorithm if Φ_f is a politician instead of an oracle because $\gamma \geq \alpha$ does not hold for the strong convexity parameter α if Φ_f is not an oracle.

The third algorithm we implemented is the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm. This algorithm uses the gradients to reconstruct the Hessian and use it to approximate Newton's method, see Algorithm 4. We note that another natural way to employ the politician with BFGS is to set $x_{k+1} = \texttt{line\_search}(\Phi_f(x_k), p)$ and this runs faster in practice; however, this algorithm computes two gradients per iteration (namely $\nabla f(x_k)$ and $\nabla f(\Phi_f(x_k))$) while we restrict ourselves to algorithms which compute one gradient per iteration.

5.2. Quadratic function

We consider the function

$$f(x) = (x-c)^\top D(x-c), \tag{2}$$

where D is a diagonal matrix with entries uniformly sampled from $[0, 1]$ and c is a random vector with entries uniformly sampled from the normal distribution $N(0, 1)$. Since this is a quadratic function, CG, BFGS and BFGS+ are equivalent and optimal, namely, they output the minimum point in the span of all previous gradients.

Algorithm 4: BFGS

Input: x_1.
for $k \leftarrow 1, 2, \cdots$ **do**
  $p = -\nabla f(x_k)$.
  for $i \leftarrow k-1, \cdots, 1$ **do**
    $\alpha_i \leftarrow \langle s_i, p\rangle / \langle s_i, y_i\rangle$.
    $p = p - \alpha_i y_i$.
  end
  $p = \langle s_{k-1}, y_{k-1}\rangle / \langle y_{k-1}, y_{k-1}\rangle\, p$.
  for $i \leftarrow 1, \cdots, k-1$ **do**
    $\beta_i \leftarrow \langle y_i, p\rangle / \langle s_i, y_i\rangle$.
    $p = p + (\alpha_i - \beta_i) y_i$.
  end
  $x_{k+1} = \Phi_f(\texttt{line\_search}(x_k, p))$.
  $s_k = x_{k+1} - x_k$, $y_k = \nabla f(x_{k+1}) - \nabla f(x_k)$.
end

5.3. Variant of Nesterov's Worst Function

(Nesterov, 2004) introduced the function

$$f(x) = (1-x[1])^2 + \sum_{k=1}^{n-1}(x[k] - x[k+1])^2$$

and used it to give a lower bound for all first-order methods. To distinguish the performance between CG, BFGS and BFGS+, we consider the following non-quadratic variant

$$f(x) = g(1-x[1]) + \sum_{k=1}^{n-1} g(x[k] - x[k+1]) \tag{3}$$

for some function g to be defined. If we pick $g(x) = |x|$ then all first order methods takes at least n iterations to minimize f exactly. On the other hand with $g(x) = \max(|x| - 0.1, 0)$ one of the minimizer of f is $(1, \frac{9}{10}, \frac{8}{10}, \cdots, \frac{1}{10}, 0, 0, \cdots, 0)$, and thus it takes at least 11 iterations for first order methods to minimize f in this case. We "regularize" the situation a bit and consider the function

$$g(x) = \begin{cases} \sqrt{(x-0.1)^2 + 0.001^2} - 0.001 & \text{if } x \geq 0.1 \\ \sqrt{(x+0.1)^2 + 0.001^2} - 0.001 & \text{if } x \leq -0.1 \\ 0 & \text{otherwise} \end{cases}.$$

Since this function is far from quadratic, our algorithms ($\emptyset$+, GK+, BFGS+) converge much faster. This is thus a nice example where the geometric politician helps a lot because the underlying dimension of the problem is small.

5.4. Binary regression with smoothed hinge loss

We consider the binary classification problem on the datasets from (Chang and Lin, 2011). The problem is to

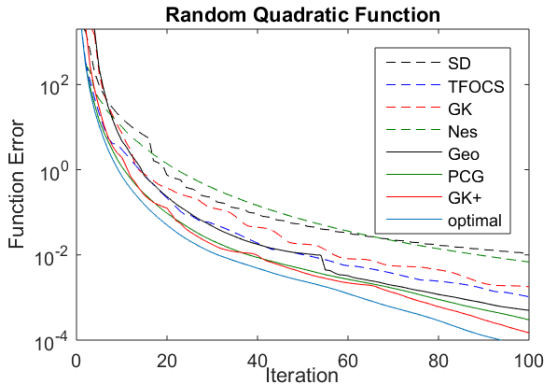

Figure 1. Comparison of first-order methods for the function (2) with $n = 10000$.

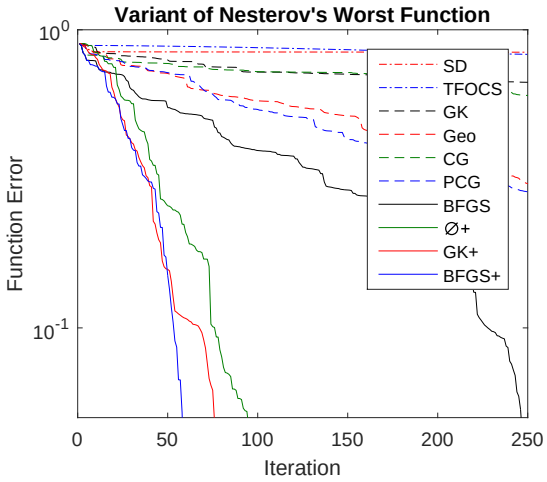

Figure 2. Comparison of first-order methods for the function (3) with $n = 10000$.

minimize the regularized empirical risk:

$$f_t(x) = \frac{1}{n}\sum_{i=1}^{n} \varphi_t(b_i a_i^T x) + \frac{\lambda}{2}|x|^2 \qquad (4)$$

where $a_i \in \mathbb{R}^d$, $b_i \in \mathbb{R}$ are given by the datasets, λ is the regularization coefficient, φ_t is the smoothed hinge loss defined by

$$\varphi_t(z) = \begin{cases} 0 & \text{if } z \le -1 \\ z + 1 - \frac{t}{2} & \text{if } z \ge -1 + t \\ \frac{1}{2t}(z+1)^2 & \text{otherwise} \end{cases}$$

and t is the smoothness parameter. The usual choice for t is 1, here we test both $t = 1$ and $t = 10^{-4}$. The latter case is to test how well the algorithms perform when the function is non-smooth.

We note that for this problem it would be natural to compare ourselves with SGD (stochastic gradient descent) or more refined stochastic algorithms such as SAG (Le Roux et al., 2012) or SVRG (Johnson and Zhang, 2013). However since the focus of this paper is on general black-box optimization we stick to comparing only to general methods. It is an interesting open problem to extend our algorithms to the stochastic setting, see Section 6.

In figures 3 and 4, we show the performance profile for problems in the LIBSVM datasets (and with different values for the regularization parameter λ). More precisely for a given algorithm we plot $x \in [1, 10]$ versus the fraction of datasets that the algorithm can solve (up to a certain prespecified accuracy) in a number of iterations which is at most x times the number of iterations of the best algorithm for this dataset. Figure 3 shows the case $t = 1$ with the targeted accuracy 10^{-6}; Figure 4 shows the case $t = 10^{-4}$ with the targeted accuracy 10^{-3}. We see that TFOCS is slower than SD for many problems, this is simply because SD uses the line search while TFOCS does not, and this makes a huge difference for simple problems. Among algorithms taking $O(n)$ time per iteration, CG and Geo perform the best, while for the $O(nk)$ algorithms we see that BFGS, BFGS+ and GK+ perform the best. The gap in performance is particularly striking in the non-smooth case where BFGS+ is the fastest algorithm on almost all problems and all other methods (except GK+) are lagging far behind (for 20% of the problems all other methods take 10 times more iterations than BFGS+ and GK+).

Finally in figures 5 and 6 we test five algorithms on three specific datasets (respectively in the smooth and non-smooth case). In both figures we see that BFGS+ performs the best for all three datasets. BFGS performs second for smooth problems while GK+ performs second for nonsmooth problems.

5.5. Summary

The experiments show that BFGS+ and BFGS perform the best among all methods for smooth test problems while BFGS+ and GK+ perform the best for nonsmooth test problems. The first phenomenon is due to the optimality of these algorithm for quadratic problems. We leave the explanation for the second phenomenon as an open problem. At least, the experiments show that this is not due to the geometric oracle itself since $\emptyset+$ is much slower, and this is not due to the original algorithm since GK performs much worse than GK+ for those problems. Overall these experiments are very promising for the geometric oracle as a replacement of quasi Newton method for non-smooth problems and as a general purpose solver due to its robustness.

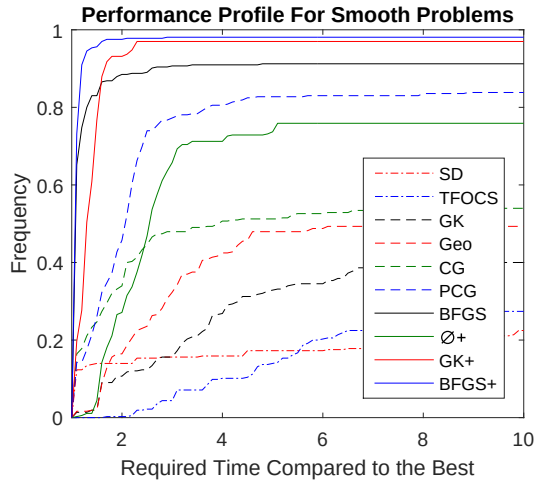

Figure 3. Performance profile on problem (4) with $t = 1$ and $\lambda = 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}$.

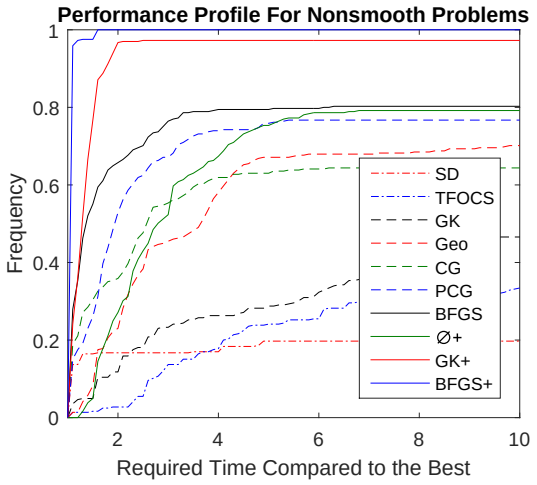

Figure 4. Performance profile on problem (4) with $t = 10^{-4}$ and $\lambda = 10^{-4}, 10^{-5}, 10^{-6}, 10^{-7}, 10^{-8}$.

6. Discussion

First order methods generally involve only very basic operations at each step (addition, scalar multiplication). In this paper we formalize each step's operations (besides the gradient calculation) as the work of the politician. We showed that the cost per step of an affine invariant politician $\psi(k)$ is negligible compared to the gradient calculation (which is $\Omega(n)$). This opens up a lot of possibilities: instead of basic addition or scalar multiplication one can imagine computing a center of gravity, solving a linear program, or even searching over an exponential space (indeed, say $k < 30$ and $n > 10^{10}$, then $2^k < n$). Our experiments demonstrate the effectiveness of this strategy. On the other hand from a theoretical point of view a lot remains to be done. For example, one can prove results of the following flavor:

Theorem 1 *Let f such that $\alpha \mathrm{I} \preceq \nabla^2 f(x) \preceq \beta \mathrm{I}, \forall x \in \mathbb{R}^n$ and let $\kappa = \beta/\alpha$. Suppose that in the Geometric Politician we replace the volumetric center by the center of gravity or the center of the John ellipsoid. Let y_k be the output of the k^{th} step of SD+ with some initial point x_0. Then, we have that*

$$f(y_k) - f(x^*) \leq \kappa \left(1 - \frac{1}{\Theta(\min(n \log(\kappa), \kappa))}\right)^k (f(x_0) - f(x^*)) .$$

and

$$f(y_k) - f(x^*) \leq \frac{2\beta R^2}{k+4}$$

where $R = \max_{f(x) \leq f(x_0)} \|x - x^*\|$.

This claim says that, up to a logarithmic factor, SD+ enjoys simultaneously the incremental progress of gradient descent and the geometrical progress of cutting plane methods. There are three caveats in this claim:

- We use the center of gravity or the center of the John ellipsoid instead of the volumetric center. Note however that it is well-known that the volumetric center is usually more difficult to analyze, (Vaidya, 1996; Atkinson and Vaidya, 1995).
- The extraneous $\log(\kappa)$ comes from the number of potential restart when we decrease α. Is there a better way to learn α that would not incur this additional logarithmic term?
- (Bubeck et al., 2015) shows essentially that one can combine the ellipsoid method with gradient descent to achieve the optimal $1 - \sqrt{1/\kappa}$ rate. Can we prove such a result for SD+?

The geometric politician could be refined in many ways. Here are two simple questions that we leave for future work:

- One can think that gradient descent stores 1 gradient information, accelerated gradient descent stores 2 gradient information, and our method stores all past gradient information. We believe that neither 1, 2 nor all is the correct answer. Instead, the algorithm should dynamically decide the number of gradients to store based on the size of its memory, the cost of computing gradients, and the information each gradient reveals.
- Is there a stochastic version of our algorithm? How well would such a method compare with state of the art stochastic algorithms such as SAG (Le Roux et al., 2012) and SVRG (Johnson and Zhang, 2013)?

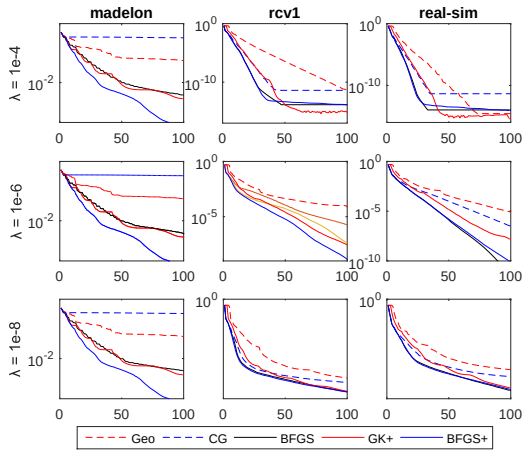

Figure 5. Comparison between Geo, CG, BFGS, GK+, BFGS+ on problem (4) with $t = 1$ and $\lambda = 10^{-4}, 10^{-6}, 10^{-8}$.

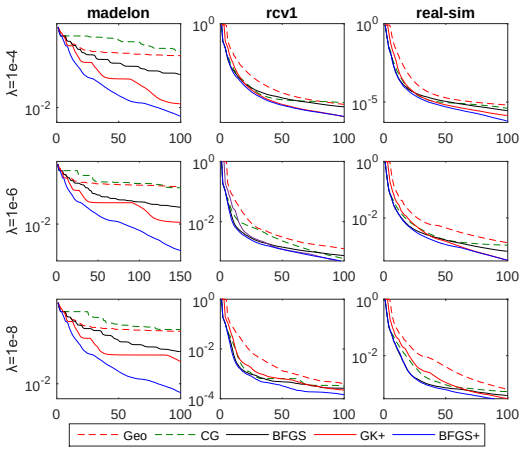

Figure 6. Comparison between Geo, CG, BFGS, GK+, BFGS+ on problem (4) with $t = 10^{-4}$ and $\lambda = 10^{-4}, 10^{-6}, 10^{-8}$.

References

M. K. Anstreicher. The volumetric barrier for convex quadratic constraints. *Mathematical Programming*, 100 (3):613–662, 2004.

David S Atkinson and Pravin M Vaidya. A cutting plane algorithm for convex programming that uses analytic centers. *Mathematical Programming*, 69(1-3):1–43, 1995.

Olivier Bahn, O Du Merle, J-L Goffin, and J-P Vial. A cutting plane method from analytic centers for stochastic programming. *Mathematical Programming*, 69(1-3):45–73, 1995.

Stephen R Becker, Emmanuel J Candès, and Michael C Grant. Templates for convex cone problems with applications to sparse signal recovery. *Mathematical programming computation*, 3(3):165–218, 2011.

S. Bubeck. Convex optimization: Algorithms and complexity. *Foundations and Trends in Machine Learning*, 8 (3-4):231–357, 2015.

S. Bubeck, Y.-T. Lee, and M. Singh. A geometric alternative to nesterov's accelerated gradient descent. *Arxiv preprint arXiv:1506.08187*, 2015.

Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST).*, 2(3):27, 2011. Software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Jean-Louis Goffin and Jean-Philippe Vial. Shallow, deep and very deep cuts in the analytic center cutting plane method. *Mathematical Programming*, 84(1):89–103, 1999.

Clóvis C Gonzaga and Elizabeth W Karas. Fine tuning nesterovs steepest descent algorithm for differentiable convex programming. *Mathematical Programming*, 138(1-2):141–166, 2013.

B. Grünbaum. Partitions of mass-distributions and of convex bodies by hyperplanes. *Pacific J. Math*, 10(4):1257–1261, 1960.

R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in Neural Information Processing Systems (NIPS)*, 2013.

N. Le Roux, M. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for strongly-convex optimization with finite training sets. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Y.-T. Lee, A. Sidford, and S. C.-W Wong. A faster cutting plane method and its implications for combinatorial and convex optimization. *Arxiv preprint arXiv:1508.04874*, 2015.

A. Nemirovski and D. Yudin. *Problem Complexity and Method Efficiency in Optimization*. Wiley Interscience, 1983.

Y. Nesterov. *Introductory lectures on convex optimization: A basic course*. Kluwer Academic Publishers, 2004.

M Schmidt. minfunc: unconstrained differentiable multivariate optimization in matlab. *URL http://www. di. ens. fr/mschmidt/Software/minFunc. html*, 2012.

P. M. Vaidya. A new algorithm for minimizing convex functions over convex sets. *Mathematical programming*, 73(3):291–341, 1996.