Minimizing Oracle-Structured Composite Functions

Xinyue Shen Alnur Ali Stephen Boyd

April 27, 2021

Abstract

We consider the problem of minimizing a composite convex function with two different access methods: an *oracle*, for which we can evaluate the value and gradient, and a *structured function*, which we access only by solving a convex optimization problem. We are motivated by two associated technological developments. For the oracle, systems like PyTorch or TensorFlow can automatically and efficiently compute gradients, given a computation graph description. For the structured function, systems like CVXPY accept a high level domain specific language description of the problem, and automatically translate it to a standard form for efficient solution. We develop a method that makes minimal assumptions about the two functions, does not require the tuning of algorithm parameters, and works well in practice across a variety of problems. Our algorithm combines a number of well-known ideas, including a low-rank quasi-Newton approximation of curvature, piecewise affine lower bounds from bundle-type methods, and two types of damping to ensure stability. We illustrate the method on stochastic optimization, utility maximization, and risk-averse programming problems.

1 Introduction

Our story starts with a well studied problem, minimizing a convex function that is the sum of two convex functions with different access methods, referred to as a *composite function* [52]. The first function is smooth, and we can access it only by a few methods, such as evaluating its value and gradient at a given point. The other function is not necessarily smooth, but is structured, and we can access it only by solving a convex optimization problem that involves it. In the typical setting, the second function is one for which we can efficiently compute its proximal operator, usually analytically. Here we assume a bit more for the second function, specifically, that we can minimize it plus a structured function of modest complexity.

Our goal is to minimize such functions automatically, with a method that works well across a large variety of problem instances using its default parameters. We leverage new technological developments: systems for automatic differentiation that automatically and effficiently compute gradients given a computation graph description, and systems for solving convex optimization problems described in a domain specific language for multiple parameter values.

1.1 Oracle-structured composite function

We seek to minimize $h(x) = f(x) + g(x)$ over $x \in \mathbf{R}^n$. We assume that

- $f : \Omega \to \mathbf{R}$ is convex and differentiable, where $\Omega \subseteq \mathbf{R}^n$ is convex and open. We assume that a point $x^0 \in \Omega$ is known.
- $g : \mathbf{R}^n \to \mathbf{R} \cup \{+\infty\}$ is convex, with closed sublevel sets, and not necessarily differentiable. Infinite values of g encode constraints on x.

Our method will access these two functions in very specific ways.

- We can evaluate $f(x)$ and $\nabla f(x)$ at any x. For $x \notin \Omega$, our oracle returns the value $+\infty$ for $f(x)$. (We will discuss a few other possible access methods for f in the sequel.)
- We can minimize $g(x)$ plus another structured function. Here we use the term structured loosely, to mean in the sense of Nesterov, or as a practical matter, in a disciplined convex programming (DCP) description. This is an extension of the usual assumption in composite function minimization, where the usual assumption is that the proximal operator of g can be evaluated analytically.

We refer to f as the oracle part of the objective, g as the structured part of the objective, and $h = f + g$ as an oracle-structured composite objective. We denote the optimal value of the problem as

$$h^\star = \inf_x \left(f(x) + g(x)\right).$$

Assumptions. We assume that the sublevel sets of h are compact, and $h^\star < \infty$ (so at least some sublevel sets are nonempty). These assumptions imply that $f(x) + g(x)$ has a minimizer, *i.e.*, a point $x^\star$ with $h(x^\star) = h^\star$. Our convergence proofs make the typical assumption that ∇f is Lipschitz continuous with constant L, but we stress that this is not used in the algorithm itself.

Optimality condition. The optimality condition is

$$\nabla f(x) + q = 0, \qquad q \in \partial g(x), \tag{1}$$

where $\partial g(x)$ denotes the subdifferential of g at x. For $x \in \Omega$ and $q \in \partial g(x)$, we can interpret $\nabla f(x)+q$ as a residual in the optimality condition. (We will use this in the stopping criterion of our algorithm.) Our access to f directly gives us $\nabla f(x)$; we will see that our access method to g indirectly produces a subgradient $q \in \partial g(x)$.

1.2 Practical considerations

We are motivated by two technological considerations related to our access methods to f and g, which we mention briefly here.

Handling f. To handle f we can rely on automatic differentiation systems that have been developed in recent years, such as PyTorch [57], TensorFlow [1], and Zygote/Flux/Autograd [33, 34, 46]. Automatic differentiation is an old topic [2, 53] (see [10] for a recent review), but these recent systems go way beyond the basic algorithms for automatic differentiation, in terms of ease of use and run-time efficiency, across multiple computation platforms. We describe f (but not its gradient) using existing libraries and languages; thereafter, $f(x)$ and $\nabla f(x)$ can be evaluated very efficiently, on many computation platforms, ranging from single CPU to multiple GPUs. These systems are widely used throughout machine learning, mostly for fitting deep neural networks.

Handling g. To handle g we make use of domain specific languages (DSLs) for convex optimization, such as CVX [29], CVXPY [3, 20], and Convex.jl [64]. These systems take a description of g in a special language based on disciplined convex programming (DCP) [30]. This description of g is then automatically transformed into a standard form, such as a cone program, and then solved. Recently, such systems have been enhanced to include parameters, which are constants in the problem each time it is solved, but can be changed and the problem re-solved efficiently, skipping the compilation process. These systems are reasonably good at preserving structure in the problem during the compilation, so the solve times can be quite small when exploitable structure is present, which we will see is the case in our method.

We mention that g can contain hidden additional variables. By this we mean that g has the form $g(x) = \inf_z G(x, z)$, where G is convex in (x, z). Roughly speaking, z contains hidden variables that do not appear in f. Such functions are immediately handled by structured systems, without any additional effort. In particular, we do not need to work out an analytical form for $g(x)$ in this case. In this case just evaluating g requires solving an optimization problem (over z). Our method will avoid any evaluations of g.

When to just use a structured solver. Finally, we mention that if f is simple enough to be handled by a DCP-based system, then simply minimizing $f + g$ using such a system is the preferred method of solution. We are interested here in problems where this is not the case. Typically this means that f is complex in the sense of involving substantial data, for example, a sample average of some function with 10^6 or more samples. (We will see this phenomenon in the numerical examples given in §5.)

Contribution. The method we propose in the next section is in the family of variable metric bundle methods, and closely related to a number of other methods found in the literature (we review related work in §2.9). As an algorithm, our method is not particularly novel; we consider our contribution to be its careful design to be compatible with our access methods.

2 Oracle-structured minimization method

In this section we propose a generic method for solving the oracle-structured minimization problem, which we call *oracle-structured minimization method* (OSMM). OSMM combines several well known methods from optimization, including variable metric or quasi-Newton curvature estimates to accelerate convergence, bundle methods that build up a piecewise affine model, and two types of damping, based on a trust penalty and a line search. These are chosen to be compatible with our access methods.

We will denote the iterates with a superscript, so x^k denotes the kth iterate of the algorithm. We will let $x^{k+1/2}$ denote the tentative iterate at the $(k+1)$st iteration, before the line search. We will assume that $x^0 \in \Omega$, *i.e.*, $f(x^0) < \infty$. Our algorithm will guarantee that $x^k \in \Omega$ for all k. It is a descent method, *i.e.*, $h(x^{k+1}) < h(x^k)$. While $h(x^0) = \infty$ is possible, we will see that $h(x^k) < \infty$ for $k \geq 1$.

As with many other optimization algorithms, OSMM is based on forming an approximation of the function f in each iteration.

2.1 Approximation of the oracle function

In iteration k, we form a convex approximation of f, given by $\hat{f}_k : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$, based on information obtained from previous iterations and possibly prior knowledge of f. The approximation has the specific form

$$\hat{f}_k(x) = l_k(x) + (1/2)(x - x^k)^T H_k (x - x^k). \tag{2}$$

Here H_k is positive semidefinite, and $l_k : \mathbf{R}^n \to \mathbf{R} \cup \{\infty\}$ is a convex minorant of f, *i.e.*, $l_k(x) \leq f(x)$ for all x.

Assumptions on H_k. The only assumption we make about H_k is that it is positive semidefinite and bounded, *i.e.*, there exists a C such that $\|H_k\|_2 \leq C$ for all k. In practice, H_k accelerates convergence by serving as an estimate of the curvature of f. The simplest choice, $H_k = 0$, results in algorithm that converges but does not offer the practical benefit of convergence acceleration.

Choice of H_k. There are many ways of choosing a positive semi-definite H_k to approximate the curvature of f at x^k. One obvious choice is the Hessian $H_k = \nabla^2 f(x^k)$, but this requires that f be twice differentiable, and also violates our assumption about how we access f. A lesser violation of the access method might use an approximation of the Hessian based on evaluations of the mapping $z \mapsto \nabla^2 f(x^k) z$ (*i.e.*, Hessian-vector multiplication), which can be practical in many cases [22]. A simple and effective choice is $H_k = (a_k/n)I$, where a_k is an approximation of $\mathbf{Tr}\, \nabla^2 f(x^k)$, obtained for example by the Hutchinson method [32, 48].

Quasi-Newton methods are a general class of curvature approximations that are compatible with our assumptions on access method for f. These methods, which have a very long history, build up an approximation of H_k using only the current and previously evaluated

gradients [14, 16, 17, 19, 24]. When H_k is low rank, or diagonal plus low rank, the method is practical even for large values of n. (Such methods are often called limited memory, since they do not require the storage of an $n \times n$ matrix.) For OSMM we propose to use the low-rank quasi-Newton choice given in [23], described in detail in §A.1. We can express H_k as

$$H_k = G_k G_k^T, \tag{3}$$

where $G_k \in \mathbf{R}^{n\times r}$, and r is a chosen (maximum) rank for H_k.

As many others have observed, limited memory quasi-Newton methods deliver most of their benefit for relatively small values of r, like $r = 10$ or $r = 20$. These values allow the methods to be used even when n is very large (say, 10^5), since the storage requirement (specifically, of G_k) grows linearly with r, and the computational cost of evaluating $x^{k+1/2}$ grows quadratically in r, and only linearly in n.

Assumptions on l_k. We make the usual assumption on the minorant l_k that it is tight at x^k, *i.e.*, $l_k(x^k) = f(x^k)$. It follows that $\hat{f}_k(x^k) = f(x^k)$. It also follows that l_k is differentiable at x^k, and $\nabla l_k(x^k) = \nabla f(x^k)$. To see this, we note that since l_k is a minorant of f, tight at x^k, we have

$$\partial l_k(x^k) \subseteq \partial f(x^k) = \{\nabla f(x^k)\}.$$

The first inclusion can be seen since any affine lower bound on l_k, tight at x^k, is also an affine lower bound on f, tight at x^k, so its linear part is a subgradient of f at x^k. The right-hand equality holds since f is differentiable, so its subdifferential contains only one element, its gradient. Finally, since $\partial l_k(x^k)$ contains only $\nabla f(x^k)$, we conclude it is differentiable at x^k, with gradient $\nabla f(x^k)$.

In addition to the mathematical assumptions about l_k described above, we will assume that l_k has a structured description. This implies that $\hat{f}_k$ has a structured description.

Minorants. The simplest minorant is the first order Taylor approximation

$$l_k(x) = f(x^k) + \nabla f(x^k)^T(x - x^k).$$

A more complex minorant is the piecewise affine minorant

$$l_k(x) = \max_{i=1,\ldots,k} \left(f(x^i) + \nabla f(x^i)^T(x - x^i)\right),$$

which uses all previously evaluated gradients of f.

For OSMM we propose the piecewise affine minorant

$$l_k(x) = \max_{i=\max\{0,k-M+1\},\ldots,k} \left(f(x^i) + \nabla f(x^i)^T(x - x^i)\right), \tag{4}$$

the pointwise maximum of the affine minorants from the previous M gradient evaluations, where M is the memory. With memory $M = 1$, this reduces to the Taylor approximation.

The problem of choosing the memory M is very similar to the problem of choosing r, the rank of the curvature approximation. The storage requirements grows linearly with M, and

the computational cost of evaluating $x^{k+1/2}$ grows quadratically with M. As with the choice of r, small values such as $M = 10$ or $M = 20$ seem to work well in practice.

We mention a few additional useful minorants that use additional prior information about f or its domain Ω. First, we can add to l_k constraints that contain Ω. Suppose we know that $\tilde{\Omega} \supset \Omega$, where $\tilde{\Omega}$ has a structured description, *e.g.*, a box. We can then use the minorant

$$l_k(x) = \max_{i=\max\{0,k-M+1\},\ldots,k} \left(f(x^i) + \nabla f(x^i)^T(x - x^i)\right) + I_{\tilde{\Omega}}(x),$$

where $I_{\tilde{\Omega}}$ is the indicator function of $\tilde{\Omega}$. In a similar way, if a (constant) lower bound ℓ on f is known, we can replace any minorant l_k with $\max\{l_k(x), \ell\}$.

If it is known that f is ρ-strongly convex, we can strengthen the piecewise affine minorant (4) to the piecewise quadratic minorant

$$l_k(x) = \max_{i=\max\{0,k-M+1\},\ldots,k} \left(f(x^i) + \nabla f(x^i)^T(x - x^i) + (\rho/2)\|x - x^i\|_2^2\right).$$

(Each term in the maximum has the same quadratic part $(\rho/2)\|x\|_2^2$, so l_k can be expressed as a piecewise affine function plus $(\rho/2)\|x\|_2^2$.)

Lower bound. We observe that

$$\ell_k = \inf_x \left(l_k(x) + g(x)\right)$$

is a lower bound on the optimal value $h^\star$. It can be computed using a system for structured optimization. At iteration k we let L_k denote the best (largest) lower bound found so far,

$$L_k = \max\{\ell_1, \ldots, \ell_k\}. \tag{5}$$

2.2 Tentative update

At iteration k, our tentative next iterate $x^{k+1/2}$ is obtained by minimizing our approximation of f, plus g and a trust penalty term:

$$x^{k+1/2} = \operatorname*{argmin}_x \left(\hat{f}_k(x) + g(x) + \frac{\lambda_k}{2}\|x - x^k\|_2^2\right). \tag{6}$$

The last term is a (Levenberg-Marquardt or proximal) trust penalty, which penalizes deviation from x^k; the positive parameter λ_k scales the trust penalty. We assume that $x^{k+1/2}$ in (6) can be computed using a system for structured optimization. (The minimizer in (6) exists and is unique, so $x^{k+1/2}$ is well defined. To see this we observe that the objective is finite for $x = x^k$, and the function being minimized is strictly convex.) We note that while $\hat{f}_k(x^{k+1/2})$ and $g(x^{k+1/2})$ are finite, $f(x^{k+1/2}) = \infty$ (and therefore also $h(x^{k+1/2}) = \infty$) is possible.

The two quadratic terms in the objective in (6) can be combined to express the tentative update as

$$x^{k+1/2} = \operatorname*{argmin}_x \left(l_k(x) + g(x) + \frac{1}{2}(x - x^k)^T (H_k + \lambda_k I)(x - x^k) \right), \tag{7}$$

which shows that the trust penalty term can be interpreted as a regularizer for the curvature estimate H_k.

In the DCP description of the problem (7), using (3) we express the last term as

$$\frac{1}{2}(x - x^k)^T (H_k + \lambda_k I)(x - x^k) = \frac{1}{2}\|G_k^T(x - x^k)\|_2^2 + \frac{\lambda_k}{2}\|x - x^k\|_2^2.$$

This keeps the problem (7) tractable when $r \ll n$ and n is large. In particular, there is no need to form the $n \times n$ matrix H_k.

Tentative update optimality condition. For future reference, we note that the optimality condition for the minimization in (7) that defines $x^{k+1/2}$ is

$$0 \in \partial l_k(x^{k+1/2}) + \partial g(x^{k+1/2}) + (H_k + \lambda_k I)(x^{k+1/2} - x^k). \tag{8}$$

When $x^{k+1/2}$ is computed using a structured solver, we can recover specific subgradients in the subdifferentials $\partial l_k(x^{k+1/2})$ and $\partial g(x^{k+1/2})$ that satisfy (8). (How to do this is explained in §A.2. We only need to find one of the two subgradients, since the three terms in (8) sum to zero, and we know the third term.) We will denote the specific subgradient in $\partial g(x^{k+1/2})$ in (8) as q^{k+1}.

When l_k is the piecewise affine minorant (4), its subdifferential $\partial l_k(x^{k+1/2})$ has the form

$$\partial l_k(x^{k+1/2}) = \mathbf{Co}\{\nabla f(x^i) \mid l_k(x^{k+1/2}) = f(x^i) + \nabla f(x^i)^T(x^{k+1/2} - x^i)\}, \tag{9}$$

the convex hull of the gradients associated with the active terms in maximum defining l_k. In the simplest case when l_k is differentiable at $x^{k+1/2}$, *i.e.*, only one term is active, this reduces to $\{\nabla l_k(x^{k+1/2})\} = \{\nabla f(x^i)\}$, where i is the (unique) index for which $l_k(x^{k+1/2}) = f(x^i) + \nabla f(x^i)^T(x^{k-1/2} - x^i)$.

As explained in §A.2, we can compute a specific subgradient $\sum_i \gamma_i \nabla f(x^i) \in \partial l_k(x^{k+1/2})$ for which (8) holds, where γ_i are nonnegative and sum to one, and positive only for i associated with active terms in the maximum that defines $l_k(x^{k+1/2})$. We then have

$$q^{k+1} = -\sum_i \gamma_i \nabla f(x^i) - (H_k + \lambda_k I)(x^{k+1/2} - x^k) \in \partial g(x^{k+1/2}). \tag{10}$$

2.3 Descent direction

If x^k is a fixed point of the tentative update, *i.e.*, $x^{k+1/2} = x^k$, then x^k is optimal. To see this, if $x^{k+1/2} = x^k$, from (8) we have

$$0 \in \partial l_k(x^k) + \partial g(x^k) = \nabla f(x^k) + \partial g(x^k), \tag{11}$$

so x^k is optimal. From the first inclusion in (11), we can also conclude that x^k minimizes $l_k(x) + g(x)$, so $L_k = l_k(x^k) + g(x^k) = f(x^k) + g(x^k)$, *i.e.*, the lower bound in (5) is tight when $x^{k+1/2} = x^k$.

Otherwise, the tentative step

$$v^k = x^{k+1/2} - x^k \tag{12}$$

is a descent direction for h at x^k, *i.e.*, for small enough $t > 0$ we have $h(x^k + tv^k) < h(x^k)$. That is, the directional derivative $h'(x^k; v^k)$ is negative.

To see this, we first observe that by (7),

$$l_k(x^{k+1/2}) + g(x^{k+1/2}) + \frac{1}{2}(v^k)^T(H_k + \lambda_k I)v^k < l_k(x^k) + g(x^k), \tag{13}$$

since $x^{k+1/2}$ minimizes the left-hand side, and the right-hand side is the same expression, evaluated at x^k. We also have

$$l_k(x^{k+1/2}) + g(x^{k+1/2}) \geq l_k(x^k) + g(x^k) + (l_k + g)'(x^k; v^k).$$

Combining these two inequalities we get

$$(l_k + g)'(x^k; v^k) < -\frac{1}{2}(v^k)^T(H_k + \lambda_k I)v^k.$$

Finally, we observe that

$$(l_k + g)'(x^k; v^k) = (f + g)'(x^k; v^k) = h'(x^k; v^k),$$

since l_k is differentiable at x^k, with $\nabla l_k(x^k) = \nabla f(x^k)$. So we have

$$h'(x^k; v^k) < -\frac{1}{2}(v^k)^T(H_k + \lambda_k I)v^k, \tag{14}$$

which shows that v^k is a descent direction for h at x^k.

2.4 Line search

The next iterate x^{k+1} is found as

$$x^{k+1} = x^k + t_k v^k = x^k + t_k(x^{k+1/2} - x^k), \tag{15}$$

where $t_k \in (0, 1]$ is the step size. When $t_k = 1$, we say the step is un-damped. We will choose t_k using a variation on a traditional Armijo-type line search [7] that avoids additional evaluations of g.

For $t \in [0, 1]$ we define

$$\phi_k(t) = f(x^k + tv^k) + tg(x^{k+1/2}) + (1 - t)g(x^k)). \tag{16}$$

Since the second and third terms are the chord above g, we have, for $t \in [0,1]$,

$$\phi_k(t) \geq h(x^k + tv^k). \tag{17}$$

Evidently $\phi_k(0) = h(x^k)$, and ϕ_k is differentiable, with

$$\phi_k'(0) = \nabla f(x^k)^T v^k + g(x^{k+1/2}) - g(x^k).$$

Since $\nabla f(x^k) = \nabla l_k(x^k)$ and l_k is convex, we get

$$\nabla f(x^k)^T v^k = \nabla l_k(x^k)^T v^k \leq l_k(x^{k+1/2}) - l_k(x^k),$$

so we have

$$\phi_k'(0) \leq l_k(x^{k+1/2}) - l_k(x^k) + g(x^{k+1/2}) - g(x^k).$$

Combining this with (13), we obtain

$$\phi_k'(0) < -\frac{1}{2}(v^k)^T(H_k + \lambda_k I)v^k. \tag{18}$$

Thus for $t > 0$ small,

$$\phi_k(t) = \phi_k(0) + \phi_k(0)'t + o(t^2) < h(x^k) - \frac{t}{2}(v^k)^T(H_k + \lambda_k I)v^k + o(t^2). \tag{19}$$

Step length. Let $\alpha, \beta \in (0,1)$. We take $t_k = \beta^j$, where j is the smallest nonnegative integer for which

$$\phi_k(t_k) \leq h(x^k) - \frac{\alpha t_k}{2}(v^k)^T(H_k + \lambda_k I)v^k \tag{20}$$

holds. (The condition (20) holds for some j by (19).) A nice feature of this line search is that it does not require any additional evaluations of the function g (which can be expensive), since we already know $g(x^k)$ and $g(x^{k+1/2})$. As has been noted by many authors, the choice of the line search parameters α and β is not critical. Traditional default values such as

$$\alpha = 0.05, \qquad \beta = 0.5 \tag{21}$$

work well in practice.

2.5 Adjusting the trust parameter

We have already observed that λ_k is a regularizer for H_k. A natural choice is to choose the regularizer parameter roughly proportional to $\tau_k = \mathbf{Tr}\, H_k/n$, since $\tau_k I$ is the minimum Frobenius norm approximation of H_k by a multiple of the identity. Thus we take

$$\lambda_k = \mu_k \left(\tau_k + \tau_{\min}\right), \tag{22}$$

where μ_k gives the trust parameter relative to τ_k, and $\tau_{\min}$ is a positive lower limit.

We update μ_k by decreasing it when the line search is undamped, *i.e.*, $t_k = 1$, and increasing it when the line search is damped, *i.e.*, $t_k < 1$. We do this with

$$\mu_{k+1} = \begin{cases} \max\{\gamma_{\rm dec}\mu_k, \mu_{\min}\} & t_k = 1 \\ \min\{\gamma_{\rm inc}\mu_k, \mu_{\max}\} & t_k < 1, \end{cases} \tag{23}$$

where $\mu_{\min}$ and $\mu_{\max}$ are positive lower and upper limits for μ_k, $\gamma_{\rm dec} \in (0, 1)$ is the factor by which we decrease μ_k, and $\gamma_{\rm inc} \in (1, \infty)$ is the factor by which we increase μ_k. The values

$$\tau_{\min} = 10^{-3}, \quad \gamma_{\rm dec} = 0.8, \quad \gamma_{\rm inc} = 1.1, \quad \mu_{\min} = 10^{-4}, \quad \mu_{\max} = 10^5 \tag{24}$$

give good results for a wide range of problems. We can take $\mu_0 = 1$.

We mention one initialization that is useful when f is twice differentiable and we have the ability to evaluate $z \mapsto \nabla^2 f(x^k)z$ (*i.e.*, Hessian-vector multiplication). In this case we can replace τ_0 with an estimate of $\mathbf{Tr}\,\nabla^2 f(x^0)/n$ obtained using the Hutchinson method [32].

2.6 Stopping criteria

We use two stopping criteria, one based on a gap between upper and lower bounds on the optimal value, and the other based on an optimality condition residual. The gap condition is simple:

$$h(x^k) - L_k \leq \epsilon_{\rm abs}^{\rm gap} + \epsilon_{\rm rel}^{\rm gap}|h(x^k)|, \tag{25}$$

where $\epsilon_{\rm abs}^{\rm gap}$ and $\epsilon_{\rm rel}^{\rm gap}$ are positive absolute and relative gap tolerances, respectively. Evaluating L_k can be almost as expensive as evaluating $x^{k+1/2}$, but it is used only in the stopping criterion. To reduce this overhead, we evaluate L_k only every ten iterations. Reasonable values for the gap tolerances are $\epsilon_{\rm abs}^{\rm gap} = 10^{-4}$ and $\epsilon_{\rm rel}^{\rm gap} = 10^{-3}$.

The residual based stopping criterion is tested whenever we take an undamped step, *i.e.*, $t_k = 1$. In this case $x^{k+1} = x^{k+1/2}$, and we obtain $q^{k+1} \in \partial g(x^{k+1})$ in (10), so $\nabla f(x^{k+1}) + q^{k+1}$ is a residual for the optimality condition (1). The stopping criterion is

$$\frac{1}{\sqrt{n}}\|\nabla f(x^{k+1}) + q^{k+1}\|_2 \leq \epsilon_{\rm abs}^{\rm res} + \epsilon_{\rm rel}^{\rm res}\left(\frac{1}{\sqrt{n}}\|\nabla f(x^{k+1})\|_2 + \frac{1}{\sqrt{n}}\|q^{k+1}\|_2\right), \tag{26}$$

where $\epsilon_{\rm abs}^{\rm res}$ and $\epsilon_{\rm rel}^{\rm res}$ are relative and absolute residual tolerances. (Dividing the norm expressions above by $\sqrt{n}$ gives the root mean square or RMS values of the argument.) Reasonable values for these parameters are $\epsilon_{\rm abs}^{\rm res} = 10^{-4}$ and $\epsilon_{\rm rel}^{\rm res} = 10^{-3}$.

2.7 Algorithm summary

We summarize OSMM in algorithm 2.1.

Algorithm 2.1 *Oracle-structured minimization method.*

given an initial point $x^0 \in \Omega$.
for $k = 0, 1, \ldots, k_{\max}$

1. *Form a surrogate objective.* Form l_k and H_k.
2. *Tentative step.* Compute $x^{k+1/2}$ by (6).
3. *Line search and update.* Set line search step size t_k by (20) and x^{k+1} by (15).
4. *Compute lower bound.* If k is a multiple of 10, evaluate L_k.
5. *Check stopping criterion.* Quit if (25) or (26) holds.
6. *Update trust penalty parameter.* Update λ_{k+1} by (22) and (23).

The algorithm parameters in OSMM are the memory of the minorant M, the rank of the curvature estimate r, the line search parameters given in (21), the λ_k update parameters given on (24), and the relative and absolute gap and residual tolerances, given in §2.6. The practical performance of OSMM is not particularly sensitive to the choice of these parameters; our implementation uses as default values the ones described above, with memory $M = 20$ and rank $r = 20$.

2.8 Implementation

We have implemented OSMM in an open-source Python package, available at

`https://github.com/cvxgrp/osmm`.

The user supplies a PyTorch description of the oracle function f, a CVXPY description of the structured function g, and an initial point $x^0 \in \Omega$. The package invokes PyTorch to evaluate f and its gradient ∇f. The convex model $\hat{f}_k$ is then formed and handed off to CVXPY to efficiently compute the next tentative iterate.

2.9 Related work

There is a lot of prior work related to the method proposed in this paper. The work on variable metric bundle methods [9,18,25,36,37,40–42,45,49,50,54,61,63,66,70] and (inexact) proximal Newton-type methods [8, 11, 12, 27, 38, 39, 43, 44, 51, 59, 60, 62, 71] are probably the most closely related, though our method differs in key ways arising from our assumed access methods. Proximal Newton methods are similar in philosophy to our approach, as both types of methods really shine when the structured part of the objective can be minimized efficiently. However, on a purely technical level, proximal Newton methods generally form the (usual) first-order model of the oracle part of the objective, without a bundle term, which is of course different than the model we use, and can affect the convergence speed.

A few variable metric bundle methods have been proposed over the years, but these methods also differ from ours in subtle (but important) ways. In general, these methods are not geared towards solving composite minimization problems. Additionally, our line search method is different, as ours is carefully designed to satisfy the access conditions.

Finally, there has been a surge of interest in scalable quasi-Newton methods [22,28,31,69] recently. The focus has been on developing low-rank, regularized, and sub-sampled Newton-type methods. Of these, the family of low-rank approximations to the Hessian are most related to the curvature estimate used in our method.

3 Convergence

In this section we show the convergence of OSMM, and give a numerical example to illustrate its typical performance, as the two most important algorithm parameters, memory M and rank r, vary. (More examples will be given in §5.)

3.1 Convergence

We demonstrate two types of convergence. First, we show the iterates x^k converge asymptotically in a certain sense, *i.e.*, the sequence of tentative steps $v^k \to 0$ as $k \to \infty$. (From (8), this implies that when the iterates x^k do converge, the convergence point must be optimal.) Second, we show the objective values generated by OSMM converge to the optimal value. The results additionally require the (standard) assumption that ∇f be Lipschitz continuous.

Convergence of iterates. To see that $v^k \to 0$, we observe from (17) and (20) that

$$h(x^{k+1}) - h(x^k) \leq -\frac{\alpha t_k}{2}(v^k)^T(H_k + \lambda_k I)v^k \leq 0. \tag{27}$$

Since $h(x^k)$ is decreasing and bounded below by $h^\star$, it converges, which implies that the left-hand side of (27) converges to zero as $k \to \infty$. This in turn implies that

$$t_k(v^k)^T(H_k + \lambda_k I)v^k \to 0 \quad \text{as} \quad k \to \infty.$$

By construction λ_k is lower bounded by $\tau_{\min}\mu_{\min}$, and t_k is also bounded away from 0, as will be shown later in (48), so we get that $v^k \to 0$, as claimed.

Convergence of objective values. We require the basic fact that there exists a subsequence of iterates (k_ℓ) accepting unit step length, *i.e.*, that $t_{k_\ell} = 1$ for all ℓ. We give a proof of this fact in §A.3, assuming that ∇f is Lipschitz continuous.

Now, by convexity,

$$h(x^{k_\ell+1}) - h^\star \leq (\nabla f(x^{k_\ell+1}) + q^{k_\ell+1})^T(x^{k_\ell+1} - x^\star),$$

for some $q^{k_\ell} \in \partial g(x^{k_\ell})$. Adding and subtracting any $l_{k_\ell}^{k_\ell+1} \in \partial l_k(x^{k+1})$, we get

$$h(x^{k_\ell+1}) - h^\star \leq \big((\nabla f(x^{k_\ell+1}) - l_{k_\ell}^{k_\ell+1}) + (l_{k_\ell}^{k_\ell+1} + q^{k_\ell+1})\big)^T (x^{k_\ell+1} - x^\star). \tag{28}$$

From our assumptions on h, and because OSMM is a descent method, we see that $\|x^{k_\ell+1} - x^\star\|_2$ is bounded. Moreover, because H_k and λ_k are uniformly bounded, we get from (8) that

$$\|l_{k_\ell}^{k_\ell+1} + q^{k_\ell+1}\|_2 \leq \|H_{k_\ell} + \lambda_{k_\ell} I\|_2 \|v^{k_\ell}\|_2 \lesssim \|v^{k_\ell}\|_2,$$

where we write $a \lesssim b$ to mean that a and b satisfy $a \leq Cb$, for some $C > 0$. Finally, it can be shown (see §A.4 for details) that the limited memory piecewise affine minorant satisfies

$$\|\nabla f(x^{k_\ell+1}) - l_{k_\ell}^{k_\ell+1}\|_2 \lesssim \max_{j=\max\{0,\ldots,k_\ell-M+1\},\ldots,k_\ell} \|v^j\|_2. \tag{29}$$

Putting these together with (28), we obtain that

$$h(x^{k_\ell+1}) - h^\star \lesssim \max_{j=\max\{0,\ldots,k_\ell-M+1\},\ldots,k} \|v^j\|_2.$$

Earlier we showed that $v^k \to 0$, so $h(x^{k_\ell+1}) \to h^\star$ as $\ell \to \infty$. Because the sequence of objective values $h(x^k)$ is convergent, we get $h(x^k) \to h^\star$, as claimed.

3.2 A numerical example

Next we investigate the convergence of OSMM numerically. Here and throughout this paper, we use a Tesla V100-SXM2-32GB-LS GPU with 32 gigabytes of memory to evaluate f and ∇f via PyTorch, and an Intel Xeon E5-2698 v4 2.20 GHz CPU to compute the tentative iterates via CVXPY and for any baselines.

Problem formulation. We consider an instance of the Kelly gambling problem [13, §4] [15, 35],

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^N \pi_i \log(r_i^T x) \\ \text{subject to} & x \geq 0, \quad \mathbf{1}^T x = 1, \end{array} \tag{30}$$

where $x \in \mathbf{R}^n$ is the variable, and $r_i \in \mathbf{R}_+^n$, $i = 1, \ldots, N$, $\pi \in \mathbf{R}_+^N$ are the problem data, with $\sum_{i=1}^N \pi_i = 1$. Here x_j is the fraction of our wealth we put on bet j, $(r_i)_j$ gives the return of bet j with outcome i, and π_i is the probability of outcome i. To put the Kelly gambling problem into our oracle-structured form, we take f to be the objective in (30), and g to be the indicator function of the constraints, in this case the probability simplex.

Problem instance. Our problem instance has $n = 1{,}000$ bets and $N = 1{,}000{,}000$ possible outcomes. Note that simply storing the returns r_i, which are dense in this instance, requires roughly four gigabytes, causing most existing solvers to struggle, as we will soon see. More detail about the Kelly gambling problem, and how the data were chosen, can be found in §5.1. This problem instance requires 4,800 seconds to solve using MOSEK [6], a high performance commercial solver, with accuracy set to its lower value.

Rank and memory		Compute times (sec.)		Total solve time and iterations		
r	M	$x^{k+1/2}$	L_k	Time (sec.)	Iterations	f evals./iter.
0	1	0.029	0.012	5.1	60	5.6
0	20	0.062	0.077	8.6	52	4.9
0	50	0.14	0.39	16	44	4.8
20	1	0.058	0.013	0.81	11	1.9
20	20	0.11	0.070	1.7	11	1.9
20	50	0.20	0.34	3.8	11	1.9
50	1	0.061	0.014	0.85	11	1.9
50	20	0.11	0.093	1.7	11	1.9
50	50	0.22	0.40	3.9	11	1.9

Table 1: Times to evaluate tentative update $x^{k+1/2}$ and lower bound L_k, total solve time and iterations to reach 10^{-6} accuracy, and average number of evaluations of f per iteration, for OSMM with the nine combinations of rank r and memory M.

Results. We solve (30) by OSMM with rank $r \in \{0, 20, 50\}$ and memory $M \in \{1, 20, 50\}$, a total of nine choices of algorithm parameters. The choices $r = 0$ and $M = 1$ correspond to using no estimate of curvature, and no memory, respectively. OSMM is run for 100 iterations for each combination of r and M; $x^{k+1/2}$ and L_k are computed using ECOS. The results are shown in figure 1 and table 1. The optimal objective value is -0.079. Figure 1 shows the convergence of OSMM, in terms of iterations (the left column) and elapsed time (the right column). The top row shows the true suboptimality $h(x^k) - h^\star$, which of course we do not know when the algorithm is running, since we do not know $h^\star$. The middle row shows the gap $h(x^k) - L_k$ (which we do know as the algorithm runs). The bottom row shows the RMS value of the optimality condition residual, *i.e.*, the left-hand side of (26).

We see that rank values $r = 20$ and $r = 50$, and memory values $M = 1$ or $M = 20$ yield the fastest convergence. We have observed similar results in many other problems. The default choices in OSMM are $r = 20$, $M = 20$.

More detail is given in table 1. The total solve time and iterations are based on achieving a high accuracy of 10^{-6}, which is far more accurate than would be needed in practice. For our nine choices of algorithm parameters, the total OSMM time ranges from around 0.8 to 16 seconds, substantially faster than using MOSEK to solve the problem.

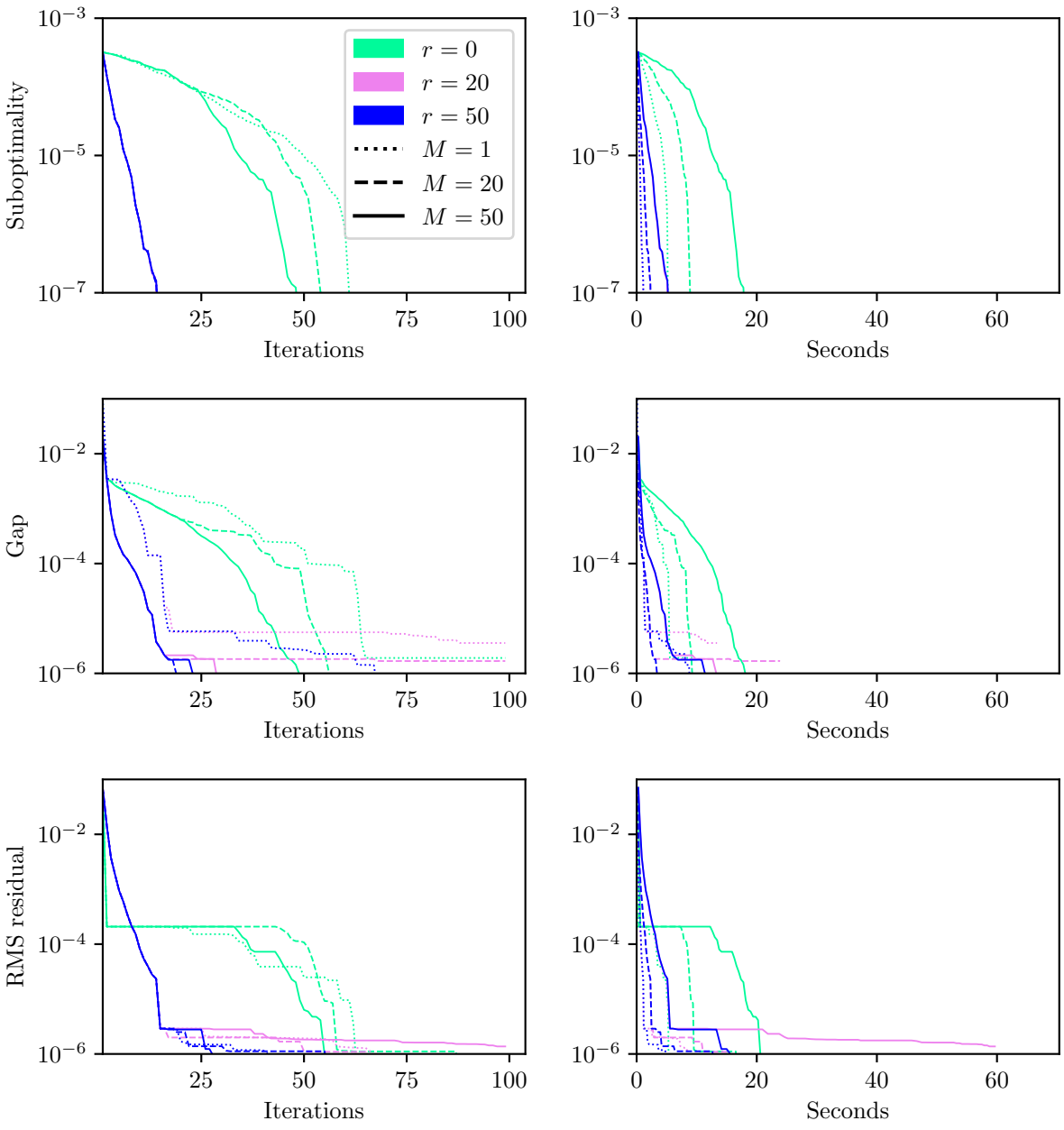

Figure 1: Suboptimality (top row), and RMS residual (bottom row), versus iterations (left column) and run-time in seconds (right column) for OSMM on the Kelly gambling problem for the nine combinations of rank r and memory M.

4 Generic applications

In this section we describe some generic applications that reduce to our specific oracle-structured composite function minimization problem. We focus on the function f, which should be differentiable, but not so simple that it can be handled directly in a structured optimization system. We will see that this generally occurs when there is a lot of data required to specify f.

4.1 Stochastic programming

Sample average approximation for stochastic programming. We start with the problem of minimizing $\mathbf{E}\, F(x,\omega)+g(x)$, where F is convex in x for each value of the random variable ω. We will approximate the first objective term using a sample average. We generate N independent samples $\omega_1,\ldots,\omega_N$, and take

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} F(x,\omega_i). \tag{31}$$

As a variation we can use importance sampling to get a lower variance estimate of $\mathbf{E}\, F(x,\omega)$. To do this we generate the samples from a proposal distribution with density q, and form the sample average estimate

$$f(x) = \frac{1}{N}\sum_{i=1}^{N} \frac{p(\omega_i)}{q(\omega_i)} F(x,\omega_i), \tag{32}$$

where p is the density of ω.

In both cases we can take N to be quite large, since we will only need to evaluate f and its gradient, and current systems for this are very efficient. For example, the gradients $\nabla F(x,\omega_i)$ can be computed in parallel. As a practical matter, this happens automatically, with no or very little directive from the user who specifies f.

Validation and stopping criterion tolerance. The functions f given in (31) and (32) are only approximations of the true objective $\mathbf{E}\, F(x,\omega)$, though we hope they are good approximations when we take N large, as we can with OSMM. To understand how accurate the sample average (31) is, we generate another set of independent samples $\tilde{\omega}_1,\ldots,\tilde{\omega}_N$ and define the validation function as

$$f^{\mathrm{val}}(x) = \frac{1}{N}\sum_{i=1}^{N} F(x,\tilde{\omega}_i)$$

(and similarly if we use importance sampling). The magnitude of the difference $|f^{\mathrm{val}}(x) - f(x)|$ gives us a rough idea of the accuracy in approximating $\mathbf{E}\, F(x,\omega)$. (Better estimates

of the accuracy can be obtained by repeating this multiple times, but we are interested in only a crude estimate.)

Solving the oracle-structured problem to an accuracy substantially better than the Monte Carlo sampling accuracy does not make sense in practice. This justifies replacing the absolute gap tolerance $\epsilon_{\text{abs}}^{\text{gap}}$ with the maximum of a fixed absolute tolerance and the sampling error $|f^{\text{val}}(x^k) - f(x^k)|$. (We can evaluate the sampling error whenever we evaluate the gap, *i.e.*, every 10 iterations.) Roughly speaking, we stop when we know we have solved the problem to an accuracy that is better than our approximation.

4.2 Utility maximization

An important special case of stochastic optimization is utility maximization, where we seek to maximize

$$\mathbf{E}\, U(H(x,\omega)) - g(x), \tag{33}$$

where $U : \mathbf{R} \to \mathbf{R}$ is a concave increasing utility function, and H is concave in x for each ω. The first term, the expected utility, is concave. This is equivalent to the stochastic programming problem of minimizing

$$\mathbf{E}(-U(H(x,\omega)) + g(x),$$

which is stochastic programming with $F = -U \circ H$, which is convex in x. We can replace the expectation with a sample average using (31), or an importance sampling sample average using (32). Utility maximization is a common method for handling the variance or uncertainty in a stochastic objective; it introduces risk aversion.

4.3 Conditional and entropic value-at-risk programming

Another method to introduce risk aversion into a stochastic optimization problem is to mimimize value-at-risk (VaR), or a specific quantile of $F(x,\omega)$, where F is convex in x for each value of the random variable ω. The value-at-risk is defined as

$$\mathbf{VaR}(F(x,\omega);\eta) = \inf\{\gamma \mid \mathbf{Prob}(F(x,\omega) \leq \gamma) \geq \eta\},$$

where $\eta \in (0,1)$ is a given quantile. With an additional structured convex function g in the objective, we obtain the VaR problem

$$\text{minimize} \quad \mathbf{VaR}(F(x,\omega);\eta)) + g(x), \tag{34}$$

which, roughly speaking, is the problem of minimizing the η-quantile of the random variable $F(x,\omega) + g(x)$. Aside from a few special cases, this problem is not convex. We proceed by replacing the nonconvex VaR term with a convex upper bound on VaR, such as conditional value-at-risk (CVaR) [58,65] or entropic value-at-risk (EVaR) [4]. Beyond resulting in tractable convex problems, these upper bounds possess a number of nice properties, such as being coherent risk measures, which VaR is not; see [4, 58] for a discussion.

CVaR is given by

$$\mathbf{CVaR}(F(x,\omega);\eta) = \inf_{\alpha\in\mathbf{R}} \left\{ \frac{\mathbf{E}(F(x,\omega)-\alpha)_+}{1-\eta} + \alpha \right\}, \tag{35}$$

where $(z)_+ = \max\{z, 0\}$, and EVaR is given by

$$\mathbf{EVaR}(F(x,\omega);\eta) = \inf_{\alpha>0} \left\{ \alpha \log \left(\frac{\mathbf{E}\exp(F(x,\omega)/\alpha)}{1-\eta} \right) \right\}. \tag{36}$$

Both of these are convex functions of x. We have, for any x,

$$\mathbf{VaR}(F(x,\omega);\eta) \leq \mathbf{CVaR}(F(x,\omega);\eta) \leq \mathbf{EVaR}(F(x,\omega);\eta)$$

(see, *e.g.*, [4]). With CVaR, we obtain the convex problem

$$\text{minimize} \quad \mathbf{CVaR}(F(x,\omega);\eta)) + g(x),$$

and similarly for EVaR.

We now show how the CVaR and EVaR problems can be approximated as oracle-structured problems. We start with CVaR. We generate independent samples ω_i, $i = 1,\ldots,N$, and replace the expectation with the empirical mean,

$$f(x,\alpha) = \frac{1}{N}\sum_{i=1}^{N} \frac{(F(x,\omega_i)-\alpha)_+}{1-\eta} + \alpha,$$

with variables x and α. This function is jointly convex in x and α; minimizing over α gives $\mathbf{CVaR}(F(x,\omega);\eta)$ for the empirical distribution. We adjoin α to x to obtain a problem in oracle-structured form, *i.e.*, minimizing $f(x,\alpha) + g(x)$, over x and α. (That is, we take (x,α) as what we call x in our general form.) While f is not differentiable in (x,α), we have observed that our method still works very well.

For EVaR, we obtain

$$f(x,\alpha) = \alpha \log \left(\frac{1}{N}\sum_{i=1}^{N} \frac{\exp(F(x,\omega_i)/\alpha)}{1-\eta} \right),$$

which is jointly convex in x and α. (To see convexity, we observe that f is the perspective function of the log-sum-exp function; see [13, §3.2.6].) As with CVaR, minimizing over α yields $\mathbf{EVaR}(F(x,\omega);\eta)$ for the empirical distribution. Unlike our approximation with CVaR, this function is differentiable.

4.4 Generic exponential family density fitting

We consider fitting an exponential family of densities, given by

$$p_\theta(z) = e^{-\left(\phi(z)^T\theta + A(\theta)\right)}, \tag{37}$$

to samples $z_1, \ldots, z_m \in \mathcal{S}$. Here, $\mathcal{S}$ is the support of the density, $\phi : \mathcal{S} \to \mathbf{R}^n$ is the sufficient statistic, and $\theta \in \Theta \subseteq \mathbf{R}^n$ is the canonical parameter (to be fitted). The density normalizes via the log-partition (or cumulant generating) function

$$A(\theta) = \log \int_{\mathcal{S}} e^{-\phi(z)^T\theta}\, dz.$$

We assume Θ is convex, and additionally that it only contains parameters for which the log-partition function is finite.

The negative log-likelihood, given samples $z_1, \ldots, z_m$, is

$$\sum_{i=1}^m \log p_\theta(z_i) = c^T\theta + mA(\theta),$$

where $c = \sum_{i=1}^m \phi(z_i)$. So maximum likelihood estimation of θ corresponds to solving the density fitting problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{m}c^T\theta + A(\theta) + \lambda r(\theta) \\ \text{subject to} & \theta \in \Theta, \end{array} \tag{38}$$

with variable θ. We can include a (potentially nonsmooth) convex regularization term $\lambda r(\theta)$ in the objective, where $\lambda \geq 0$ is the regularization strength, and $r : \Theta \to \mathbf{R}$ is the regularizer. Since the log-partition function is convex [67], the density fitting problem (38) is also convex.

The log-partition function is generally intractable except for a few special cases, so we replace the integral in $A(\theta)$ with a finite sum using importance sampling, *i.e.*,

$$\int_{\mathcal{S}} e^{-\phi(z)^T\theta}\, dz \approx \frac{1}{N}\sum_{i=1}^N \frac{1(\omega_i \in \mathcal{S})}{q(\omega_i)} e^{-\phi(\omega_i)^T\theta},$$

where $\omega_i,\ i = 1, \ldots, N$, are independent draws from the proposal distribution q. The number of samples N can be very large, especially when the number of dimensions n is moderate. When $\mathcal{S}$ is bounded and its dimension is small, we can simply use a Riemann sum, so that the samples ω_i are lattice points in $\mathcal{S}$ and we have $q(\omega_i) = 1/|\mathcal{S}|$. The problem (38) is clearly in oracle-structured form, once we take $A(\theta)$ (with its Monte Carlo approximation) to be f, and the rest of the objective plus the indicator of Θ to be g.

A number of interesting regularizers are possible. The squared ℓ_2 norm, *i.e.*, $r(\theta) = \|\theta\|_2^2$, is of course a natural choice. When $\mathcal{S}$ is bounded, a different option is to use the squared L_2 norm of the gradient of the log-density

$$r(\theta) = \int_{\mathcal{S}} \|\nabla \log p_\theta(z)\|_2^2\, dz.$$

This regularizer enforces a certain kind of smoothness: the regularized density p_θ tends to the uniform distribution on $\mathcal{S}$, as the regularization strength λ grows. Finally, observe that we can write

$$\int_{\mathcal{S}} \|\nabla \log p_\theta(z)\|_2^2 \, dz = \int_{\mathcal{S}} \|D\phi(z)^T\theta\|_2^2 \, dz = \theta^T \left(\int_{\mathcal{S}} D\phi(z) D\phi(z)^T \, dz \right) \theta,$$

where $D\phi$ is the Jacobian of sufficient statistic ϕ. We can replace the integral with a finite sum again, and obtain the regularizer

$$r(\theta) \approx \theta^T \left(\frac{1}{N} \sum_{i=1}^{N} \frac{1(\omega_i \in \mathcal{S})}{q(\omega_i)} D\phi(\omega_i) D\phi(\omega_i)^T \right) \theta.$$

5 Numerical examples

In this section we demonstrate the performance of OSMM through several numerical examples, all taken from the generic applications described in the previous section. We start by showing results for two different portfolio selection problems, the Kelly gambling example (shown earlier in §3.2), and minimizing CVaR. We then show a density estimation example. After that we present results for a supply chain optimization problem with entropic value-at-risk minimization. All of these examples are structured stochastic optimization problems, and we use simple sample averages to approximate expectations. OSMM is designed to handle the case when f is complex, which in the case of sample averages means N is large. We will see that when N is small, OSMM is actually slower than just solving the problem directly using a structured solver; when N is large, it is much faster (and in many cases, directly using a structured solver fails).

We report the time needed for OSMM to reach high accuracy, *i.e.*, $h(x^k) - h^\star \leq 10^{-6}$. (This makes a fairer comparison with MOSEK, SCS [55, 56], and ECOS [21].) We also indicate when practical accuracy is reached, using our default parameters, and the sampling accuracy. We use the default parameters in OSMM, and use ECOS as the solver in CVXPY to compute the tentative iterate $x^{k+1/2}$. We do not perform any parameter tuning for our method.

5.1 Kelly gambling

Problem formulation. In the Kelly gambling problem there are n bets we can wager on, and N possible outcomes, with probabilities π_i, $i = 1, \ldots, N$. The bet returns are given by $r_i \in \mathbf{R}^n_+$, where $(r_i)_j$ is the return, *i.e.*, the amount you win for each dollar you put on bet j when outcome i occurs. We are to choose $x \in \mathbf{R}^n$, with $\mathbf{1}^T x = 1$, where x_j is the fraction of our wealth we place on bet j. We seek to maximize the average log return, which maximizes long-term wealth growth if we repeatedly bet. This leads to the (convex) optimization problem

$$\begin{array}{ll} \text{maximize} & \sum_{i=1}^N \pi_i \log(r_i^T x) \\ \text{subject to} & x \geq 0, \quad \mathbf{1}^T x = 1, \end{array}$$

N	OSMM	MOSEK	SCS	ECOS
1,000	0.76	0.58	6.4	2.2
10,000	0.64	4.5	2,100	50
100,000	0.62	62	—	910
1,000,000	0.64	890	—	20,000

Table 2: Solve times in seconds on the Kelly gambling problem. A dash ("—") means the solver failed, either for numerical reasons, or because it did not reach the required 10^{-6} suboptimality in 24 hours.

with variable x.

Problem instance. We consider $n = 200$ bets. The probabilities of the outcomes π_i are independent draws from a uniform distribution on $[0, 1]$, normalized to sum to one. The returns of the bets in each outcome are independently drawn from a log normal distribution, *i.e.*, $\exp(z)$, $z \sim \mathcal{N}(0, 1)$, and then scaled so the expected return of bet j is $\bar{r}_j$, *i.e.*, $\sum_{i=1}^{N} \pi_i (r_i)_j = \bar{r}_j$, where $\bar{r}_j$ is drawn from a uniform distribution on $[0.9, 1.1]$. For our problem instance, the solution has 57 nonzero entries ranging from 0.001 to 0.04, and the optimal mean log return is 0.057.

Results. The run-times for OSMM, MOSEK, SCS, and ECOS are shown in table 2. When the number of Monte Carlo samples is small, *e.g.*, $N = 1{,}000$, MOSEK performs the fastest and takes less than a second to attain high accuracy. OSMM also takes less than one second. ECOS takes about two seconds, and SCS takes roughly six seconds. However, when N is larger (*i.e.*, $N = 10{,}000$, 100,000, or 1,000,000), OSMM is the fastest method, always taking less than a second to attain the required 10^{-6} optimality gap. When $N = 10{,}000$, MOSEK is still competitive, but SCS and ECOS are two to four orders of magnitude slower than OSMM. When $N = 100{,}000$, MOSEK and ECOS are two to three orders of magnitude slower, and when $N = 1{,}000{,}000$, MOSEK and ECOS are three to five orders of magnitude slower. SCS fails for both the two larger values of N. These findings suggest that OSMM is useful when the number of samples is large, as it exhibits good scaling with N.

OSMM spends 0.0013 and 0.0024 seconds to evaluate f and its gradient ∇f, respectively, when $N = 1{,}000{,}000$. Computing the tentative iterate $x^{k+1/2}$ and the lower bound L_k from (5) takes 0.033 and 0.014 seconds, respectively. The line search also turns out to be quite efficient here, as f is evaluated twice during the line search on average.

Figure 2 shows the convergence of OSMM with $N = 1{,}000{,}000$. In the top panel, we can see that practical accuracy is reached after 14 iterations, and high accuracy is reached after 16 iterations, as shown by the dotted black and green lines, respectively.

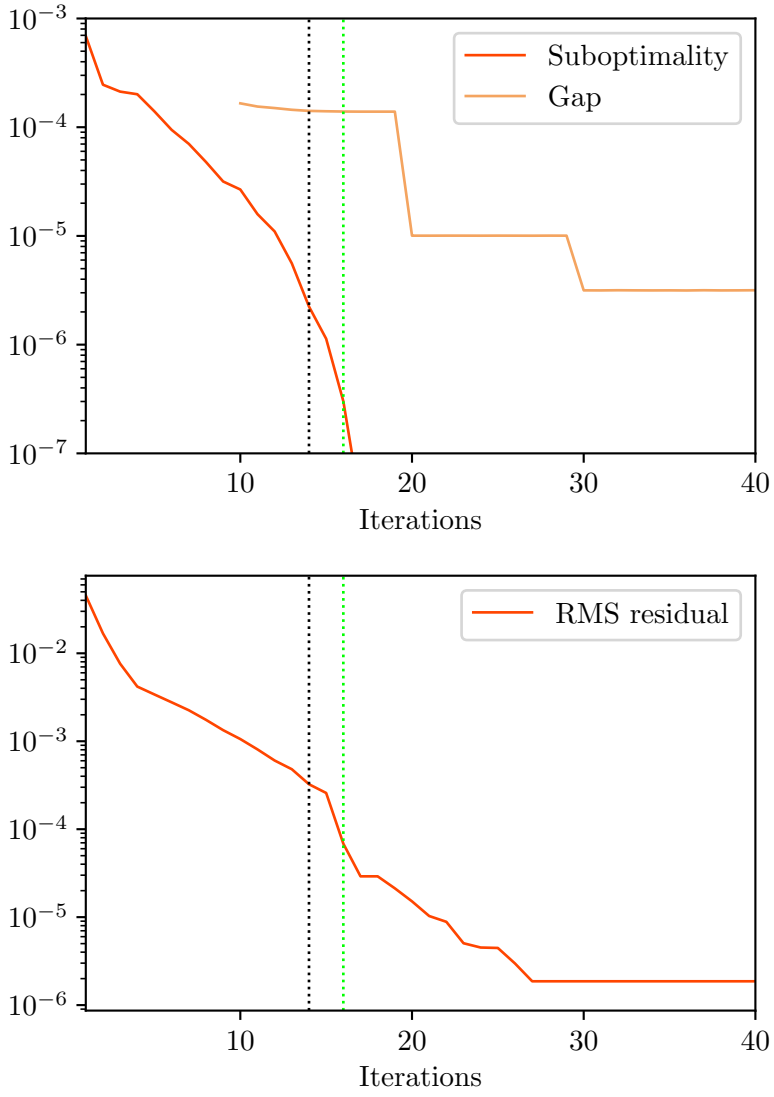

Figure 2: Suboptimality, gap (top row), and RMS residual (bottom row) on the Kelly gambling problem with $N = 1{,}000{,}000$. The dotted green and black lines show when high accuracy and practical accuracy are reached, respectively.

5.2 CVaR portfolio optimization with derivatives

Problem formulation. We consider making investments in m stocks, and call and put derivatives on them, with various strike prices. Our investment will be for one period (of, say, one month). We let $\omega \in \mathbf{R}^m_{++}$ denote the (fractional) change in prices of the m underlying stocks, which we model as random with a log-normal distribution, *i.e.*, $\log \omega \sim \mathcal{N}(\mu, \Sigma)$, where the log is elementwise. For simplicity we will assume there is one call and one put option available for each stock. Let $p_{\mathrm{c}} \in \mathbf{R}^m_{++}$ and $s_{\mathrm{c}} \in \mathbf{R}^m_{++}$ be the call option prices (premia) and strike prices, normalized by the current stock price, so, for example, $(s_{\mathrm{c}})_i = 1.15$ means the strike price is 1.15 times the current stock price. Let $p_{\mathrm{p}} \in \mathbf{R}^m_{++}$ and $s_{\mathrm{p}} \in \mathbf{R}^m_{++}$ denote the corresponding quantities for the m put options.

The amount we receive per dollar of investment in the underlying stocks is ω, the ratio of the current to final stock price. For every dollar invested in the call options we receive $(\omega - s_{\mathrm{c}})_+/p_{\mathrm{c}}$, where the division is elementwise, and $(a)_+ = \max\{a, 0\}$. For the put options, the total we receive per dollar of investment is $(s_{\mathrm{p}} - \omega)_+/p_{\mathrm{p}}$.

We make investments in $n = 3m$ different assets, the m underlying stocks and m associated call and put options. We let $x \in \mathbf{R}^n$ denote the fractions of our wealth that we invest in the assets, so $\mathbf{1}^T x = 1$. We consider long and short positions, with $x_i < 0$ denoting a short position. We consider a simple set of portfolio constraints, $x \geq x_{\min}$ (*e.g.*, $x_{\min} = -0.1$ limits the maximum short position for any asset to not exceed 10% of the total portfolio value), and $\|x\|_1 \leq L$, where L is a leverage limit. (Since $\mathbf{1}^T x = 1$, this means that the total long position cannot exceed a multiple $(L+1)/2$ of total wealth, and the total short position cannot exceed a multiple $(L-1)/2$ of the total wealth.) We partition x as $x = (x_{\mathrm{u}}, x_{\mathrm{c}}, x_{\mathrm{p}})$, with each subvector in $\mathbf{R}^m$. Our portfolio has total return

$$x_{\mathrm{u}}^T \omega + x_{\mathrm{c}}^T((s_{\mathrm{c}} - \omega)_+/p_{\mathrm{c}}) + x_{\mathrm{p}}^T((s_{\mathrm{p}} - \omega)_+/p_{\mathrm{p}}) = r(\omega)^T x,$$

where $r(\omega) \in \mathbf{R}^n$ is the total return of the $n = 3m$ assets, *i.e.*, stocks and options.

Our problem is to choose the portfolio so as to minimize the conditional value at risk (described in (35)) of the negative total return, *i.e.*,

$$\begin{array}{ll} \text{minimize} & \mathbf{CVaR}(-r(\omega)^T x; \eta) \\ \text{subject to} & x \geq x_{\min},\ \mathbf{1}^T x = 1,\ \|x\|_1 \leq L, \end{array}$$

where $\eta \in (0, 1)$ sets the risk aversion.

We use a sample average approximation of the expectation in CVaR to obtain the problem

$$\begin{array}{ll} \text{minimize} & \frac{1}{N}\sum_{i=1}^N \frac{(-r(\omega_i)^T x - \alpha)_+}{1-\eta} + \alpha \\ \text{subject to} & x \geq x_{\min},\ \mathbf{1}^T x = 1,\ \|x\|_1 \leq L \end{array}$$

with variables $x = (x_{\mathrm{u}}, x_{\mathrm{c}}, x_{\mathrm{p}}) \in \mathbf{R}^n$ and $\alpha \in \mathbf{R}$. The vectors $\omega_i \in \mathbf{R}^n_{++}$, $i = 1, \ldots, N$, are independent samples of the price change ω.

N	OSMM	MOSEK	SCS	ECOS
10,000	11	6.0	250	18
100,000	6.5	64	2,900	310
1,000,000	6.3	1,900	30,000	5,200

Table 3: Solve times in seconds on the conditional value-at-risk problem. A dash ("—") means the solver failed, either for numerical reasons, or because it did not reach the required 10^{-6} suboptimality in 24 hours.

Problem instance. We take the number of stocks as $m = 100$, so $n = 300$. We take the minimum position limit $x_{\min} = -0.1$ and leverage limit $L = 1.6$. We use risk aversion parameter $\eta = 0.8$, so we are attempting to minimize the 20th percentile of the portfolio loss. The price change covariance Σ is generated according to $\Sigma = \sigma^2(I + 0.2FF^T)$, where $\sigma = 1/\sqrt{2}$, and the entries of $F \in \mathbf{R}^{m\times 5}$ are independent draws from a standard normal distribution. We set the mean price change according to $\mu_i = 0.03\sqrt{\Sigma_{ii}} - 0.5\Sigma_{ii}$, $i = 1, \ldots, m$. For each call option, the strike price is set as the 80th percentile of ω, and for each put option it is the 20th percentile. The option prices are determined by the Black-Scholes formula with zero discount. (These data are approximately consistent with an investment period of one month for U.S. equities.)

When we solve this problem instance, the optimal portfolio return has mean 1.3%, standard deviation 5%, and loss probability 39%; annualized, these correspond to a return of 16%, standard deviation 18%, and loss probability 20%. The optimal portfolio contains as assets the underlying stocks as well as call and put options.

Results. Table 3 shows the run-times for N ranging from 10,000 to 1,000,000. We see again that for small values of N, it is more efficient to solve the problem directly using a structured solver, whereas for large values, OSMM is far more efficient. When $N = 1{,}000{,}000$, OSMM (using PyTorch) takes 0.0021 seconds to evaluate f and 0.0053 seconds to evaluate ∇f; OSMM takes 0.050 seconds to compute the tentative iterate $x^{k+1/2}$, and 0.021 seconds to evaluate the lower bound L_k (using CVXPY). Figure 3 shows the convergence of OSMM with $N = 1{,}000{,}000$. Practical accuracy are high accuracy are reached at the same time after 100 iterations.

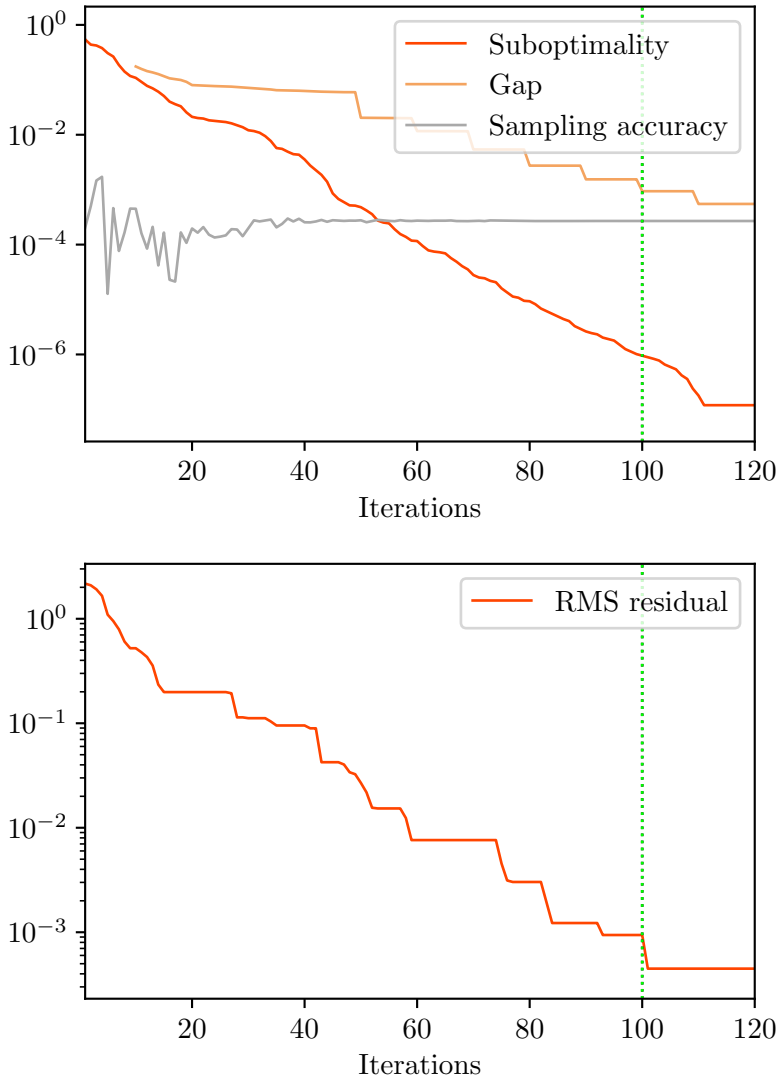

Figure 3: Suboptimality, gap, sampling accuracy (top row), and RMS residual (bottom row) on the conditional value-at-risk problem with $N = 1{,}000{,}000$. The dotted green and black lines coincide, indicating that high accuracy and practical accuracy are reached at the same time.

N	OSMM	MOSEK	SCS	ECOS
10,000	0.72	1.2	1,100	0.92
100,000	1.2	11	—	14
1,000,000	0.84	120	—	190

Table 4: Solve times in seconds on the density estimation problem. A dash ("—") means the solver failed, either for numerical reasons, or because it did not reach the required 10^{-6} suboptimality in 24 hours.

5.3 Exponential series density estimation

Problem formulation. We consider an instance of the generic exponential family density fitting problem described in §4.4. We consider data $z_1, \ldots, z_m \in \mathbf{R}^2$, and wish to fit a density p supported on $\mathcal{S} = [-1, 1]^2$. We let the sufficient statistics ϕ_i, $i = 1, \ldots, n$, be the Legendre polynomials up to degree four, so $n = 14$. (This is known as an exponential series density estimator [26, 47, 68].) We solve the density estimation problem (38).

Problem instance. We take m data points sampled from a mixture of three Gaussian densities, restricted to $\mathcal{S}$, with means $(1/3, 1/3)$, $(1/3, -1/3)$, and $(-1/3, -1/3)$, weights 0.4, 0.3, and 0.3, and common covariance $(1/36)I$. (So the data do not come from the family of density we use to fit.) We form a Riemann sum using points in $\mathcal{S}$ lying on a grid with side lengths $\sqrt{N}$. Recall that N, the number of samples, here refers to our approximate evaluation of the integral that arises in the log-partition function, and not the number of data samples, which is fixed at $m = 2{,}000$.

Results. Table 4 shows the run-times for the various methods. We see the usual pattern where directly solving the problem can be efficient for small N, but OSMM is much faster for large N. When $N = 1{,}000{,}000$, it takes OSMM 0.001 and 0.0014 seconds to evaluate f and ∇f, respectively, and 0.015 and 0.0097 seconds to compute $x^{k+1/2}$ and L_k, respectively.

Figure 4 shows the convergence of OSMM for $N = 1{,}000{,}000$. Practical accuracy is reached after 39 iterations with suboptimality 10^{-5}, and after 42 iterations OSMM reaches high accuracy. In this instance, our lower bound $L_k = -\infty$, so neither it nor the gap are plotted in the figure.

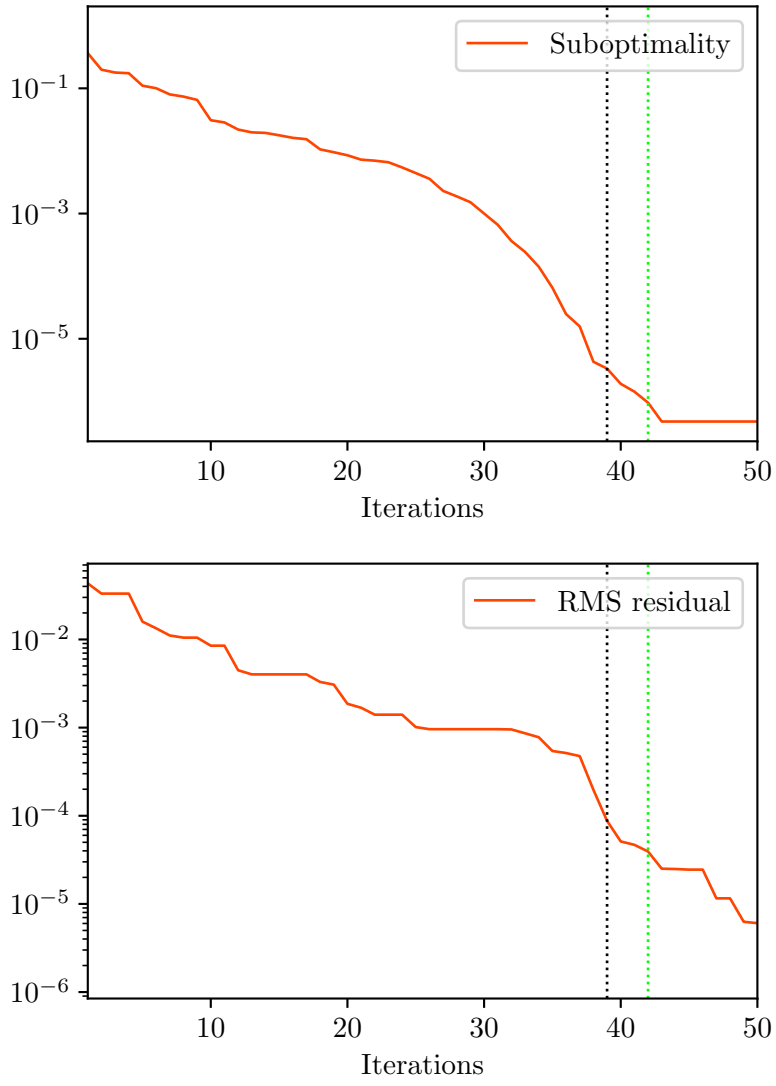

Figure 4: Suboptimality (top row) and RMS residual (bottom row) on the exponential family density fitting problem with $N = 1{,}000{,}000$. The dotted green and black lines show when high accuracy and practical accuracy are reached, respectively.

5.4 Vector news vendor with entropic value-at-risk

We consider a variant of the classic news vendor problem that involves entropic value-at-risk programming [5].

Problem formulation. We choose quantities $q \in \mathbf{R}^n_+$ of n products to produce, at total cost $\phi(q)$, where $\phi : \mathbf{R}^n_+ \to \mathbf{R}_+$. We have constraints on the quantities we can produce, $q \leq q_{\max}$, and on the total production cost, $\phi(q) \leq \phi_{\max}$. We sell the amount $\min(q, d)$, where $d \in \mathbf{R}^n_+$ is the demand, and the min is taken elementwise, at market prices $p \in \mathbf{R}^n_+$, so the total revenue is $p^T \min(q, d)$, and the profit is

$$P = p^T \min\{q, d\} - \phi(q).$$

We assume that $\omega = (d, p) \in \mathbf{R}^{2n}_+$ is a random variable with known distribution, so $P(q, \omega)$ is a random variable that depends on q.

We choose the quantities q to minimize the EVaR of the negative profit,

$$\begin{array}{ll} \text{minimize} & \mathbf{EVaR}(-P(q,\omega);\eta) \\ \text{subject to} & \phi(q) \leq \phi_{\max},\ q \geq 0,\ q \leq q_{\max}, \end{array}$$

where η is a specified quantile.

As described in §4.3, we approximate this with Monte Carlo samples $(d_1, p_1), \ldots, (d_N, p_N)$ to obtain the problem

$$\begin{array}{ll} \text{minimize} & \alpha \log\left(\frac{1}{(1-\eta)N}\sum_{i=1}^N \exp\left(\frac{-p_i^T \min(q,d_i)+\phi(q)}{\alpha}\right)\right) \\ \text{subject to} & \phi(q) \leq \phi_{\max},\ q \geq 0,\ q \leq q_{\max},\ \alpha \geq 0, \end{array}$$

with variables q and $\alpha \in \mathbf{R}_+$. We denote the realizations of the demand d and prices p on the ith Monte Carlo simulation by $d_i, p_i \in \mathbf{R}^n_+,\ i = 1, \ldots, N$, respectively.

Problem instance. We take $n = 500$ and risk aversion parameter $\eta = 0.9$. We assume the demand and market prices follow a joint log-normal distribution, *i.e.*, $(d, p) = \exp z$, $z \sim \mathcal{N}(\mu, \Sigma)$, and the exponential is elementwise. We draw the entries of $\mu \in \mathbf{R}^{2n}$ independently from a uniform distribution on $[-0.2, 0]$, and set $\Sigma = 0.1FF^T$, where the entries of $F \in \mathbf{R}^{2n\times 5}$ are independently drawn from a standard normal distribution. We use a production cost which is separable and piecewise affine with one kink point for each entry of q,

$$\phi(q) = a^T q + 0.5a^T(q - b)_+,$$

where the elements in a and b are both drawn uniformly at random from $[0.2, 0.9]$ and $[0.01, 0.03]$, respectively. The maximum production quantities $q_{\max}$ is set as $5b$, and the maximum cost is $\phi_{\max} = 1$. With these parameter values, the optimal profit has mean 3.2 and standard deviation 0.98.

N	OSMM	MOSEK	SCS	ECOS
1,000	6.7	120	—	—
10,000	11	7,400	—	—
100,000	10	—	—	—
1,000,000	53	—	—	—

Table 5: Solve times in seconds on the vector news vendor problem. A dash ("—") means the solver failed, either for numerical reasons, or because it did not reach the required 10^{-6} suboptimality in 24 hours.

Results. The run-times for the various methods are in table 5. In this instance, OSMM is the fastest for all values of N ranging from 1,000 to 1,000,000, and the other solvers fail for nearly all values of N. When $N = 1{,}000{,}000$, OSMM takes 0.16 and 0.27 seconds to evaluate f and ∇f, respectively; it takes 0.076 seconds to compute the tentative iterate, and 0.036 seconds to compute the lower bound.

Figure 5 shows the convergence of OSMM with $N = 1{,}000{,}000$. We can see that practical accuracy is reached after 50 iterations with suboptimality on the order of 10^{-5}, while it takes 58 iterations to reach high accuracy.

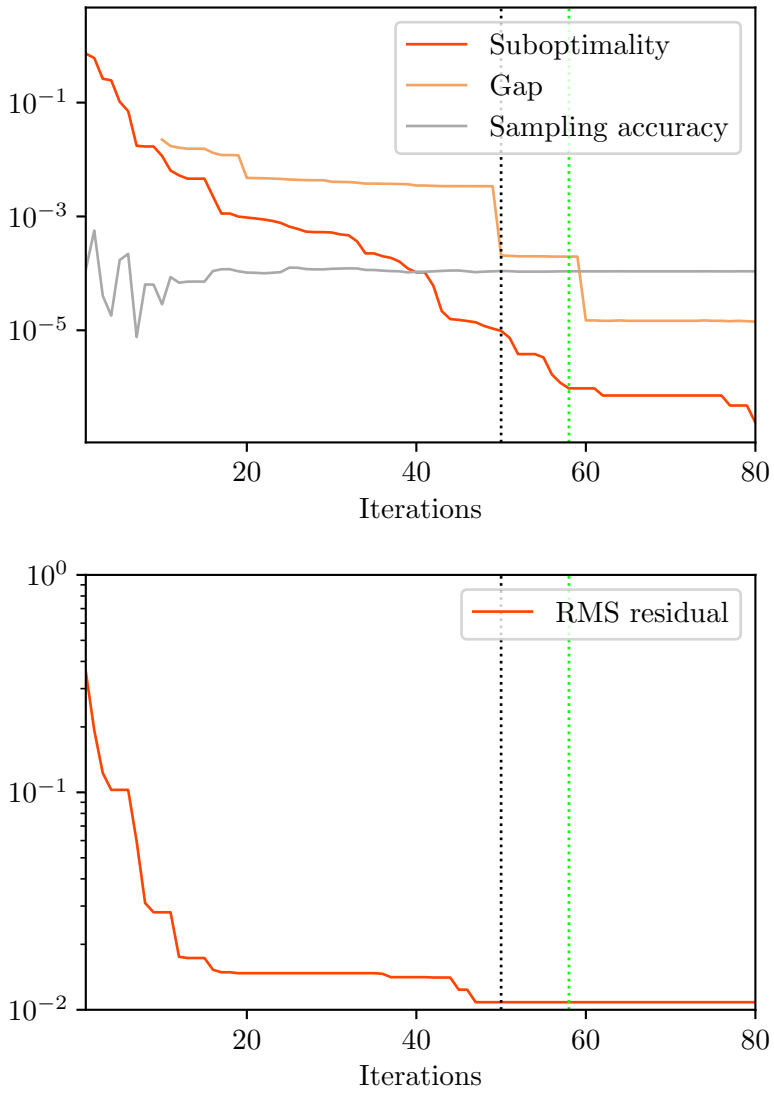

Figure 5: Suboptimality, gap, sampling accuracy (top row), and RMS residual (bottom row) on the vector news vendor problem with $N = 1{,}000{,}000$. The dotted green and black lines show when high accuracy and practical accuracy are reached, respectively.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] D. S. Adamson and C. W. Winant. A SLANG simulation of an initially strong shock wave downstream of an infinite area change. In *Proceedings of the Conference on Applications of Continuous-System Simulation Languages*, pages 231–240, 1969.

[3] A. Agrawal, R. Verschueren, S. Diamond, and S. Boyd. A rewriting system for convex optimization problems. *Journal of Control and Decision*, 5(1):42–60, 2018.

[4] A. Ahmadi-Javid. An information-theoretic approach to constructing coherent risk measures. In *2011 IEEE International Symposium on Information Theory Proceedings*, pages 2125–2127. IEEE, 2011.

[5] S. Ahmed, U. Çakmak, and A. Shapiro. Coherent risk measures in inventory problems. *European Journal of Operational Research*, 182(1):226–238, 2007.

[6] MOSEK ApS. *MOSEK Optimizer API for Python 9.2.40*, 2019.

[7] L. Armijo. Minimization of functions having Lipschitz continuous first partial derivatives. *Pacific Journal of mathematics*, 16(1):1–3, 1966.

[8] H. Asi and J. Duchi. Modeling simple structures and geometry for better stochastic optimization algorithms. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 2425–2434, 2019.

[9] A. Bagirov, N. Karmitsa, and M. M. Mäkelä. *Bundle Methods. Introduction to Nonsmooth Optimization: Theory, Practice and Software*, pages 305–310. Springer International Publishing, Cham, 2014.

[10] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of machine learning research*, 18, 2018.

[11] S. Becker, J. Fadili, and P. Ochs. On quasi-Newton forward-backward splitting: Proximal calculus and convergence. *SIAM Journal on Optimization*, 29(4):2445–2481, 2019.

[12] S. Becker and M. J. Fadili. A quasi-Newton proximal splitting method. *arXiv preprint arXiv:1206.1156*, 2012.

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

[14] C. G. Broyden. A class of methods for solving nonlinear simultaneous equations. *Mathematics of computation*, 19(92):577–593, 1965.

[15] E. Busseti, E. K. Ryu, and S. Boyd. Risk-constrained Kelly gambling. *The Journal of Investing*, 25(3):118–134, 2016.

[16] R. H. Byrd, J. Nocedal, and R. B. Schnabel. Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1):129–156, 1994.

[17] W. C. Davidon. Variable metric method for minimization. Technical report, Argonne National Lab., Lemont, Ill., 1959.

[18] W. De Oliveira and M. Solodov. A doubly stabilized bundle method for nonsmooth convex optimization. *Mathematical programming*, 156(1-2):125–159, 2016.

[19] J. E. Dennis, Jr and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM review*, 19(1):46–89, 1977.

[20] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 2016.

[21] A. Domahidi, E. Chu, and S. Boyd. ECOS: An SOCP solver for embedded systems. In *European Control Conference (ECC)*, pages 3071–3076, 2013.

[22] M. A. Erdogdu and A. Montanari. Convergence rates of sub-sampled Newton methods. *Advances in Neural Information Processing Systems*, 28:3052–3060, 2015.

[23] R. Fletcher. A new low rank quasi-Newton update scheme for nonlinear programming. In *IFIP Conference on System Modeling and Optimization*, pages 275–293. Springer, 2005.

[24] R. Fletcher and M. Powell. A rapidly convergent descent method for minimization. *The computer journal*, 6(2):163–168, 1963.

[25] M. Fukushima. A descent algorithm for nonsmooth convex optimization. *Mathematical Programming*, 30(2):163–175, 1984.

[26] Y. Gao, Y. Y. Zhang, and X. Wu. Penalized exponential series estimation of copula densities with an application to intergenerational dependence of body mass index. *Empirical Economics*, 48(1):61–81, 2015.

[27] H. Ghanbari and K. Scheinberg. Proximal quasi-Newton methods for regularized convex optimization with linear and accelerated sublinear convergence rates. *Computational Optimization and Applications*, 69(3):597–627, 2018.

[28] R. Gower, N. Le Roux, and F. Bach. Tracking the gradients using the hessian: A new look at variance reducing stochastic methods. In *International Conference on Artificial Intelligence and Statistics*, pages 707–715. PMLR, 2018.

[29] M. Grant and S. Boyd. CVX: Matlab software for disciplined convex programming, version 2.1, 2014.

[30] M. Grant, S. Boyd, and Y. Ye. Disciplined convex programming. In L. Liberti and N. Maculan, editors, *Global Optimization: From Theory to Implementation*, Nonconvex Optimization and its Applications, pages 155–210. Springer, 2006.

[31] X. Huang, X. Liang, Z. Liu, L. Li, Y. Yu, and Y. Li. Span: A stochastic projected approximate newton method. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(02):1520–1527, 2020.

[32] M. F. Hutchinson. A stochastic estimator of the trace of the influence matrix for Laplacian smoothing splines. *Communications in Statistics-Simulation and Computation*, 18(3):1059–1076, 1989.

[33] M. Innes, A. Edelman, K. Fischer, C. Rackauckas, E. Saba, V. B. Shah, and W. Tebbutt. A differentiable programming system to bridge machine learning and scientific computing. *arXiv preprint arXiv:1907.07587*, 2019.

[34] M. Innes, E. Saba, K. Fischer, D. Gandhi, M. C. Rudilosso, N. M. Joy, T. Karmali, A. Pal, and V. Shah. Fashionable modelling with Flux. *CoRR*, abs/1811.01457, 2018.

[35] J. L. Kelly Jr. A new interpretation of information rate. In *The Kelly capital growth investment criterion: theory and practice*, pages 25–34. World Scientific, 2011.

[36] K. C. Kiwiel. Proximity control in bundle methods for convex nondifferentiable minimization. *Mathematical programming*, 46(1):105–122, 1990.

[37] K. C. Kiwiel. Efficiency of proximal bundle methods. *Journal of Optimization Theory and Applications*, 104(3):589–603, 2000.

[38] C.-P. Lee and S. J. Wright. Inexact successive quadratic approximation for regularized optimization. *Computational Optimization and Applications*, 72(3):641–674, 2019.

[39] J. D. Lee, Y. Sun, and M. A. Saunders. Proximal Newton-type methods for minimizing composite functions. *SIAM Journal on Optimization*, 24(3):1420–1443, 2014.

[40] C. Lemaréchal. *Nonsmooth optimization and descent methods*. RR-78-004, 1978.

[41] C. Lemaréchal, A. Nemirovskii, and Y. Nesterov. New variants of bundle methods. *Mathematical programming*, 69(1):111–147, 1995.

[42] C. Lemaréchal and C. Sagastizábal. Variable metric bundle methods: from conceptual to implementable forms. *Mathematical Programming*, 76(3):393–410, 1997.

[43] E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Computational mathematics and mathematical physics*, 6(5):1–50, 1966.

[44] J. Li, M. S. Andersen, and L. Vandenberghe. Inexact proximal Newton methods for self-concordant functions. *Mathematical Methods of Operations Research*, 85(1):19–41, 2017.

[45] L. Lukšan and J. Vlček. A bundle-Newton method for nonsmooth unconstrained minimization. *Mathematical Programming*, 83(1):373–391, 1998.

[46] D. Maclaurin, D. Duvenaud, and R. P. Adams. Autograd: Effortless gradients in numpy. In *ICML 2015 AutoML Workshop*, volume 238, page 5, 2015.

[47] P. Marsh. Goodness of fit tests via exponential series density estimation. *Computational statistics & data analysis*, 51(5):2428–2441, 2007.

[48] R. A. Meyer, C. Musco, C. Musco, and D. P. Woodruff. Hutch++: Optimal stochastic trace estimation. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 142–155. SIAM, 2021.

[49] R. Mifflin. A quasi-second-order proximal bundle algorithm. *Mathematical Programming*, 73(1):51–72, 1996.

[50] R. Mifflin, D. Sun, and L. Qi. Quasi-Newton bundle-type methods for nondifferentiable convex optimization. *SIAM Journal on Optimization*, 8(2):583–603, 1998.

[51] B. S. Mordukhovich, X. Yuan, S. Zeng, and J. Zhang. A globally convergent proximal Newton-type method in nonsmooth convex optimization. *arXiv preprint arXiv:2011.08166*, 2020.

[52] Y. Nesterov. Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161, 2013.

[53] J. F. Nolan. *Analytical differentiation on a digital computer*. PhD thesis, Massachusetts Institute of Technology, 1953.

[54] D. Noll. Bundle method for non-convex minimization with inexact subgradients and function values. In *Computational and analytical mathematics*, pages 555–592. Springer, 2013.

[55] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. Conic optimization via operator splitting and homogeneous self-dual embedding. *Journal of Optimization Theory and Applications*, 169(3):1042–1068, June 2016.

[56] B. O'Donoghue, E. Chu, N. Parikh, and S. Boyd. SCS: Splitting conic solver, version 2.1.2. `https://github.com/cvxgrp/scs`, November 2019.

[57] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems 32*, pages 8024–8035, 2019.

[58] R. T. Rockafellar and S. Uryasev. Conditional value-at-risk for general loss distributions. *Journal of banking & finance*, 26(7):1443–1471, 2002.

[59] K. Scheinberg and X. Tang. Practical inexact proximal quasi-Newton method with global complexity analysis. *Mathematical Programming*, 160(1):495–529, 2016.

[60] M. Schmidt, E. Berg, M. Friedlander, and K. Murphy. Optimizing costly functions with simple constraints: A limited-memory projected quasi-Newton algorithm. In *Artificial Intelligence and Statistics*, pages 456–463. PMLR, 2009.

[61] H. Schramm and J. Zowe. A version of the bundle idea for minimizing a nonsmooth function: Conceptual idea, convergence analysis, numerical results. *SIAM journal on optimization*, 2(1):121–152, 1992.

[62] S. Sra, S. Nowozin, and S. J. Wright. *Optimization for machine learning*. MIT Press, 2012.

[63] C. H. Teo, S. V. N. Vishwanathan, A. Smola, and Q. V. Le. Bundle methods for regularized risk minimization. *Journal of Machine Learning Research*, 11(1), 2010.

[64] M. Udell, K. Mohan, D. Zeng, J. Hong, S. Diamond, and S. Boyd. Convex optimization in Julia. In *Proceedings of the Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28, 2014.

[65] S. Uryasev and R. T. Rockafellar. *Conditional Value-at-Risk: Optimization Approach*, pages 411–435. Springer US, Boston, MA, 2001.

[66] W. van Ackooij, J. Y. Bello Cruz, and W. de Oliveira. A strongly convergent proximal bundle method for convex minimization in Hilbert spaces. *Optimization*, 65(1):145–167, 2016.

[67] M. Wainwright and M. Jordan. *Graphical models, exponential families, and variational inference*. Now Publishers Inc, 2008.

[68] X. Wu. Exponential series estimator of multivariate densities. *Journal of Econometrics*, 156(2):354–366, 2010.

[69] H. Ye, L. Luo, and Z. Zhang. Approximate Newton methods and their local convergence. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 3931–3939, 2017.

[70] J. Yu, S. V. N. Vishwanathan, S. Günter, and N. N. Schraudolph. A quasi-Newton approach to nonsmooth convex optimization problems in machine learning. *The Journal of Machine Learning Research*, 11:1145–1200, 2010.

[71] M.-C. Yue, Z. Zhou, and A. M.-C. So. A family of inexact SQA methods for non-smooth convex minimization with provable convergence guarantees based on the Luo–Tseng error bound property. *Mathematical Programming*, 174(1):327–358, 2019.

A Appendix

A.1 Details of forming G_k

We form G_k in (3) by adopting a quasi-Newton update given in [23]. The main idea is to divide G_k into two matrices $G_k^{(1)}$ and $G_k^{(2)}$, which are the first r_1 and the last $r_2 = r - r_1$ columns in G_k, respectively, and update them separately into $G_{k+1}^{(1)}$ and $G_{k+1}^{(2)}$. Then G_{k+1} is updated by

$$G_{k+1} = \begin{bmatrix} G_{k+1}^{(1)} & G_{k+1}^{(2)} \end{bmatrix}.$$

The detail is as follows. Let $s_k = x^{k+1} - x^k$ and $y_k = \nabla f(x^{k+1}) - \nabla f(x^k)$. From the convexity of f, $s_k^T y_k \geq 0$. Suppose that $s_k^T y_k$ is not too small such that

$$s_k^T y_k > \max(\varepsilon_{\text{abs}}, \varepsilon_{\text{rel}} \|s_k\|_2 \|y_k\|_2), \tag{39}$$

where constants $\varepsilon_{\text{abs}}, \varepsilon_{\text{rel}} > 0$ are given. Then r_1 is chosen as the largest integer in $[0, r]$ such that $G_k^{(1)}$ satisfies

$$s_k^T y_k - \left\| (G_k^{(1)})^T s_k \right\|_2^2 > \varepsilon_{\text{rel}} \|s_k\|_2 \left\| y_k - G_k^{(1)} (G_k^{(1)})^T s_k \right\|_2.$$

According to (39) the above holds at least for $r_1 = 0$, in which case $G_k^{(1)}$ degenerates to 0.

Once r_1 is obtained, we define $w_k^{(1)} = (G_k^{(1)})^T s_k$ and $w_k^{(2)} = (G_k^{(2)})^T s_k$. Then $R_k^{(1)} \in \mathbf{R}^{(r_1+1)\times(r_1+1)}$ is the upper triangular factor in the following R-Q decomposition

$$R_k^{(1)} Q_k^{(1)} = \begin{bmatrix} \frac{1}{\sqrt{s_k^T y_k - \|w_k^{(1)}\|_2^2}} & 0_{r_1}^T \\ \frac{-1}{\sqrt{s_k^T y_k - \|w_k^{(1)}\|_2^2}} w_k^{(1)} & I_{r_1} \end{bmatrix} \in \mathbf{R}^{(r_1+1)\times(r_1+1)},$$

and $Q_k^{(2)} \in \mathbf{R}^{r_2 \times (r_2-1)}$ is a set of orthonormal basis orthogonal to $w_k^{(2)}$. The update is

$$G_{k+1}^{(1)} = \begin{bmatrix} y_k & G_k^{(1)} \end{bmatrix} R_k^{(1)}, \quad G_{k+1}^{(2)} = G_k^{(2)} Q_k^{(2)}.$$

There are some corner cases. If (39) holds and $r_1 = 0$, then $G_{k+1}^{(1)} = y_k / \sqrt{s_k^T y_k}$. If $r_1 = r - 1$ or $r_1 = r$, then $G_{k+1}^{(2)}$ vanishes, and G_{k+1} takes the first r columns of $G_{k+1}^{(1)}$.

In cases where (39) does not hold, if $\|w_k^{(2)}\|_2 > \varepsilon_{\text{abs}}$, then we can still define $G_{k+1}^{(2)}$ in the same way, and $G_{k+1} = \begin{bmatrix} G_{k+1}^{(2)} & 0_n \end{bmatrix}$. Otherwise, $G_{k+1} = G_k$.

It can be easily checked that by the G_k defined as above, the trace of H_k is uniformly upper bounded in k.

The default values for the parameters are $\varepsilon_{\text{abs}} = 10^{-8}$ and $\varepsilon_{\text{rel}} = 10^{-3}$.

A.2 Details of computing an optimal subgradient of g

Here we show how to compute a subgradient $q^{k+1} \in \partial g(x^{k+1/2})$ satisfying the optimality conditions in (8), which implies (10). This, in turn, is important because it allows us to compute the stopping criteria described in §2.6.

Since we know the third term on the right-hand side of (8), it suffices to find an optimal subgradient $l_k^{k+1/2} \in \partial l_k(x^{k+1/2})$, which is easier. We start by rewriting the defining problem for $x^{k+1/2}$ in a more useful form. Putting (4) and (7) together, we see that we can alternatively express $x^{k+1/2}$ as the solution to the convex problem

$$\begin{array}{ll} \text{minimize} & z + g(x) + \frac{1}{2}(x - x^k)^T(H_k + \lambda_k I)(x - x^k) \\ \text{subject to} & z \geq f(x^i) + \nabla f(x^i)^T(x - x^i), \quad \text{for } i = \max\{0, k - M + 1\}, \ldots, k, \end{array} \tag{40}$$

with variables x and $z \in \mathbf{R}$.

The KKT conditions for problem (40), which are necessary and sufficient for the points $(x^{k+1/2}, z^{k+1/2})$ and $\gamma_i,\ i = \max\{0, k - M + 1\}, \ldots, k$, to be primal and dual optimal, are

$$z^{k+1/2} \geq f(x^i) + \nabla f(x^i)^T(x^{k+1/2} - x^i), \quad i = \max\{0, k - M + 1\}, \ldots, k; \tag{41}$$

$$\sum_{i=\max\{0,k-M+1\}}^{k} \gamma_i = 1, \quad \gamma_i \geq 0, \quad i = \max\{0, k - M + 1\}, \ldots, k; \tag{42}$$

$$z^{k+1/2} > f(x^i) + \nabla f(x^i)^T(x^{k+1/2} - x^i) \implies \gamma_i = 0, \quad i \in \max\{0, k - M + 1\}, \ldots, k; \tag{43}$$

$$\partial g(x^{k+1/2}) + (H_k + \lambda_k I)v^k + \sum_{i=\max\{0,k-M+1\}}^{k} \gamma_i \nabla f(x^i) \ni 0. \tag{44}$$

Here we used the definition of v^k to simplify the stationarity condition (44).

Now we claim that

$$l_k^{k+1/2} = \sum_{i=\max\{0,k-M+1\}}^{k} \gamma_i \nabla f(x^i) \in \partial l_k(x^{k+1/2}).$$

To see this, note that (42) says the γ_i are nonnegative and sum to one, while (41) and (43) together imply γ_i is positive as long as $\nabla f(x^i)$ is active; this satisfies (9), which says the subdifferential $\partial l_k(x^{k+1/2})$ is the convex hull of the active gradients. Therefore, re-arranging (44) gives

$$q^{k+1} = -l_k^{k+1/2} - (H_k + \lambda_k I)v^k \in \partial g(x^{k+1/2}),$$

which shows (10).

A.3 Proof that undamped steps occur infinitely often

Assume that ∇f is Lipschitz continuous with constant L. Also, assume $\mu_{\max}\tau_{\min} > 2L/(1 - \alpha)$. We show that for any number of iterations k_0, there is some $k \geq k_0$ such that $t_k = 1$. This means that there exists a subsequence $(k_\ell)_{\ell=1}^{\infty}$ such that $t_{k_\ell} = 1$.

To show the result, we first claim that the line search condition (20) is satisfied as soon as

$$t_k \leq \frac{1-\alpha}{2L}\lambda_k. \tag{45}$$

We prove the claim later. Taking the claim as a given for now, the main result follows by deriving a contradiction. To get a contradiction, suppose there exists some number of iterations k_0 such that $t_k < 1$ for each $k \geq k_0$. Then, re-arranging the claim, we get that $\lambda_k < 2L/(1-\alpha)$, for each $k \geq k_0$. But from (23), we have $\mu_k = \min\left\{\gamma_{\text{inc}}^{k-k_0}\mu_{k_0}, \mu_{\max}\right\}$, since we assumed $t_k < 1$ for every $k \geq k_0$. Additionally, from (22), we get $\lambda_k \geq \mu_k\tau_{\min}$. So, we now have two cases. Either we have $\lambda_k \geq \mu_{\max}\tau_{\min} > 2L/(1-\alpha)$, which is a contradiction. Or we have $\lambda_k \geq \gamma_{\text{inc}}^{k-k_0}\mu_{k_0}\tau_{\min}$, in which case $\gamma_{\text{inc}}^{k-k_0}\mu_{k_0}\tau_{\min}$ grows exponentially in k; this means we must have $\lambda_k \geq 2L/(1-\alpha)$, for k sufficiently large, which is again a contradiction. This finishes the proof of the main result.

Now we prove the claim. Observe that

$$Lt_k^2\|v^k\|^2 \leq \frac{1-\alpha}{2}\lambda_k t_k\|v^k\|^2 \leq \frac{1-\alpha}{2}t_k(v^k)^T(H_k + \lambda_k I)v^k, \tag{46}$$

where we used (45) to get the first inequality. We now use the following two facts: ∇f being L-Lipschitz continuous implies that (i) ϕ_k is convex in t, and (ii) ϕ_k' is $L\|v^k\|^2$-Lipschitz in t. By the first fact (convexity), we have

$$\phi_k(t_k) \leq \phi_k(0) + \phi_k'(t_k)t_k.$$

Adding and subtracting $\phi_k'(0)$ gives

$$\phi_k(t_k) \leq \phi_k(0) + \phi_k'(0)t_k + (\phi_k'(t_k) - \phi_k'(0))t_k.$$

The second fact (Lipschitz continuity) yields a bound on the third term on the right-hand side,

$$\phi_k(t_k) \leq \phi_k(0) + \phi_k'(0)t_k + Lt_k^2\|v^k\|^2. \tag{47}$$

Finally, using (18) and (46) to bound $\phi_k'(0)$ and $Lt_k^2\|v^k\|_2^2$ in (47) yields (20), which implies the line search condition (20) is satisfied, as claimed.

We note that the claim above also implies the following lower bound, which is used in §3.1. Observe that if t_k indeed satisfies the line search condition, then we can express $t_k = \beta^j$, where j is the smallest integer such that (45) holds (see §2.4). Now consider two cases. If $1 \leq (1-\alpha)\lambda_k/(2L)$, then we can simply take $j = 0$, so that $t_k = 1$. On the other hand, if $1 > (1-\alpha)\lambda_k/(2L)$, then a short calculation shows that $j = \lceil \log_\beta((1-\alpha)\lambda_k/(2L))\rceil$, and so $j < 1 + \log_\beta((1-\alpha)\lambda_k/(2L))$, giving $t_k = \beta^j > \beta(1-\alpha)\lambda_k/(2L)$. Therefore, to sum up, we have (45) implies that t_k satisfies the inequalities

$$t_k > \beta\min\left\{1, \frac{1-\alpha}{2L}\lambda_k\right\} \geq \beta\min\left\{1, \frac{1-\alpha}{2L}\mu_{\min}\tau_{\min}\right\}, \tag{48}$$

where we also used the fact that $\lambda_k \geq \mu_{\min}\tau_{\min}$.

A.4 Proof that the limited memory piecewise affine minorant is accurate enough

Assume that ∇f is Lipschitz continuous with constant L. For the rest of the proof, fix any k such that $t_k = 1$. (From §A.3, it is always possible to find at least one such k.) We will show that for any such k, the limited memory piecewise affine minorant (4) satisfies the bound (29); because our choice of k was arbitrary, the required result will then follow.

For any $l_k^{k+1} \in \partial l_k(x^{k+1})$, note that adding and subtracting $\nabla f(x^k)$ in the left-hand side of (29) easily gives

$$\|\nabla f(x^{k+1}) - l_k^{k+1}\|_2 \leq \|\nabla f(x^{k+1}) - \nabla f(x^k)\|_2 + \|\nabla f(x^k) - l_k^{k+1}\|_2. \tag{49}$$

The Lipschitz continuity of ∇f, in turn, immediately gives a bound for the first term on the right-hand side of (49), *i.e.*, we get

$$\begin{aligned}\|\nabla f(x^{k+1}) - l_k^{k+1}\|_2 &\leq L\|x^{k+1} - x^k\|_2 + \|\nabla f(x^k) - l_k^{k+1}\|_2 \\ &\lesssim \|v^k\|_2 + \|\nabla f(x^k) - l_k^{k+1}\|_2,\end{aligned} \tag{50}$$

using the definition of v^k.

Therefore, we focus on the second term on the right-hand side of (49). For this term, (9) tells us that for any $l_k^{k+1} \in \partial l_k(x^{k+1})$,

$$\|\nabla f(x^k) - l_k^{k+1}\|_2 \leq \max_{j=\max\{0,k-M+1\},\ldots,k} \|\nabla f(x^k) - \nabla f(x^j)\|_2, \tag{51}$$

because the maximum of a convex function over a convex polytope is attained at one of its vertices. The Lipschitz continuity of ∇f again shows that, for any $j \in \{\max\{0, k-M+1\}, \ldots, k\}$,

$$\begin{aligned}\|\nabla f(x^k) - \nabla f(x^j)\|_2 &\leq L\|x^k - x^j\|_2 \\ &= L\|(x^k - x^{k-1}) + (x^{k-1} - x^{k-2}) + \cdots + (x^{j+2} - x^{j+1}) + (x^{j+1} - x^j)\|_2 \\ &\leq L\sum_{\ell=j}^{k-1} \|v^\ell\|_2 \\ &\lesssim \max_{\ell=j,\ldots,k-1} \|v^\ell\|_2.\end{aligned}$$

To get the third line, we used the fact that the sum on the second line telescopes, and applied the triangle inequality. To get the fourth line, we used the fact that the average is less than the max. Putting this last inequality together with (51), we see that

$$\|\nabla f(x^k) - l_k^{k+1}\|_2 \lesssim \max_{j=\max\{0,k-M+1\},\ldots,k} \|v^j\|_2, \tag{52}$$

for any $l_k^{k+1} \in \partial l_k(x^{k+1})$. Combining (52) and (50) gives (29). This completes the proof of the result.