Multidimensional Stochastic Approximation: Adaptive Algorithms and Applications*

Mark Broadie† Columbia University

Deniz M. Cicek‡ Columbia University

Assaf Zeevi§ Columbia University

October 27, 2011

Abstract

We consider prototypical sequential stochastic optimization methods of Robbins-Monro (RM), Kiefer-Wolfowitz (KW) and simultaneous perturbations stochastic approximation (SPSA) varieties and propose adaptive modifications for multidimensional applications. These adaptive versions dynamically scale and shift the tuning sequences to better match the characteristics of the unknown underlying function, as well as the noise level. We test our algorithms on a variety of representative applications in inventory management, health care, revenue management, supply chain management, financial engineering and queueing theory.

1 Introduction

1.1 Background and overview of main contributions

Stochastic approximation is an iterative procedure which can be used to estimate the root of a function or the point of optimum, where the function values cannot be calculated exactly and typically need to be estimated in the presence of noise. A common application is Monte Carlo simulation where the noise is due to the sample generation scheme.

The first type of stochastic approximation algorithm dates back to the work of Robbins and Monro (RM) [22] and focuses on estimating the root of an unknown function. Following this, Kiefer and Wolfowitz [15] introduced an algorithm to find the point of maximum of a function observed with noise, i.e., to solve $\max_{x\in\mathbb{R}^d} f(x) = \mathbb{E}[\widetilde{f}(x)]$ where $\widetilde{f}$ is a noisy observation of $f : \mathbb{R}^d \to \mathbb{R}$. Kiefer and Wolfowitz [15] considered the one-dimensional case ($d = 1$), but shortly thereafter, a multidimensional version of the KW-algorithm was introduced by Blum [6]. His multidimensional

*This work was supported by NSF grants DMII-0447652 and DMS-0914539.

†Graduate School of Business, e-mail: mnb2@columbia.edu

‡Graduate School of Business, email: dcicek05@gsb.columbia.edu (contact author)

§Graduate School of Business, e-mail: assaf@gsb.columbia.edu

KW-algorithm uses a one-sided finite-difference approximation of the gradient in each direction and can be described by the following recursion:

$$X^{(n+1)} = X^{(n)} + a^{(n)} \frac{\widetilde{g}(X^{(n)})}{c^{(n)}}, \quad n = 1, 2, \dots \tag{1}$$

Here $\widetilde{g}(X^{(n)}) = (\widetilde{f}(X^{(n)} + c^{(n)}e_1) - \widetilde{f}(X^{(n)}), \dots, \widetilde{f}(X^{(n)} + c^{(n)}e_d) - \widetilde{f}(X^{(n)}))$, where $\{e_1, \dots, e_d\}$ is the standard basis in $\mathbb{R}^d$, and the "tuning sequences" $\{a^{(n)}\}$ and $\{c^{(n)}\}$ are real-valued and deterministic. If the gradient of the underlying function, $f'(x)$, satisfies $(x - x^*)^T f'(x) \leq -K_0 \|x - x^*\|^2$ and $\|f'(x)\| \leq K_1 \|x - x^*\|$ for all $x \in \mathbb{R}^d$, and for some $0 < K_0 < K_1$, then under some further assumptions on the tuning sequences, it is proven that the mean-squared-error (MSE), $\mathbb{E}(X^{(n)} - x^*)^2$, converges to zero as $n \to \infty$; see [9] for a simple proof, (the original proof in [6] establishes almost sure convergence).

Most of the literature on stochastic approximation algorithms focus on their asymptotic performance. To that end, in the case of KW-type algorithms, the choice $a^{(n)} = \alpha/(n + \beta)$ and $c^{(n)} = \gamma/n^{1/4}$, for some $\alpha, \gamma \in \mathbb{R}_+$ and $\beta \in \mathbb{Z}_+$, is proven to be optimal in the sense that this choice optimizes the rate of convergence of the MSE; see, [11]. With this choice of sequences, the MSE converges to zero at rate $O(1/\sqrt{n})$ provided that the constant α in the specification of the $\{a^{(n)}\}$ sequence is chosen large enough relative to the unknown constant K_0 which describes the "curvature" of the underlying function. It is well documented, see, e.g., [16], that the finite-time behavior of these algorithms is quite sensitive to the specification of these tuning sequences through the parameters α, β and γ, and to the best of our knowledge there is no clear guidance in the literature on how to choose these parameters.

Roughly speaking, there are three main finite-time performance issues stemming from a "poor" choice of the constants in the tuning sequences. First, if the $\{a^{(n)}\}$ sequence is "too small" relative to the gradient (since the constant K_0 is not known in advance), then as shown in [19] the theoretical convergence rate guarantees may no longer hold even asymptotically, i.e., the convergence rate degrades. On the other hand, if the $\{a^{(n)}\}$ sequence is chosen "too large" relative to the magnitude of the gradient, the algorithm may take steps which are "too large" in search for the optimum value of the function, and if the domain is constrained to, say, a hypercube or hyper-rectangle, the algorithm may oscillate between boundaries of the domain for a "large" number of iterations. Finally, gradient estimates may be "too noisy" as a consequence of a small $\{c^{(n)}\}$ sequence. See [9] for a detailed description and analysis of each one of these issues for one-dimensional problems, and further discussion in Section 2.

Main Contributions. This paper introduces a class of the multidimensional stochastic approximation algorithms which adaptively adjust the constants α, β and γ in the tuning sequences to match underlying function characteristics (e.g., magnitude of the gradient) and the noise level af-

fecting observations. The multidimensional adaptation ideas are extensions of the one-dimensional KW-algorithm proposed in [9]. The present paper also describes versions of the RM root-finding algorithm and the simultaneous perturbations stochastic approximation (SPSA) algorithm introduced by Spall [23], and focuses on the efficiency of these algorithms on a range of realistic stylized applications. In particular, the test problems are drawn from in the following application domains: inventory management (a multidimensional newsvendor problem); health care (determining ambulance base locations); revenue management (determining airline seat protection levels for different fare classes); supply chain management (cost minimization in a multiechelon production-inventory system), financial engineering (calibration of Merton model parameters to put option prices); and service systems/queueing (a call center staffing problem). In almost all of these test cases, we observe that the finite-time performance is significantly improved by the proposed adaptive adjustments in the tuning sequences.

1.2 Related literature

The Robbins-Monro algorithm. The first stochastic approximation algorithm introduced by Robbins and Monro [22], and then extended to multiple dimensions by Blum [6], uses the following recursion to estimate the zero crossing of a function:

$$X^{(n+1)} = X^{(n)} - a^{(n)}\widetilde{g}(X^{(n)}), \tag{2}$$

where $\widetilde{g}(\cdot)$ is a noisy observation of the function $g(\cdot) : \mathbb{R}^d \to \mathbb{R}^d$, and $\{a^{(n)}\}$ is a positive real number sequence. Under some assumptions on the function and the $\{a^{(n)}\}$ sequence, Blum [6] proves that $X^{(n)} \to x^*$ almost surely as $n \to \infty$ where $g(x^*) = 0$; see [5] and [17] for more recent convergence results for this algorithm.

The SPSA algorithm. The multidimensional KW-algorithm (1) uses one-sided finite-difference approximation of the gradient to avoid having a large number of function evaluations at every gradient estimation point. Note that for a d-dimensional problem, we need $d+1$ function evaluations to estimate the gradient at each iteration. In order to decrease the number of function evaluations per iteration and increase efficiency, Spall [23] introduced the simultaneous perturbations stochastic approximation (SPSA) algorithm which uses a randomization idea that relies on only two function evaluations at every iteration (independent of the dimension of the problem d). More specifically, the algorithm can be described using the following recursion:

$$X_k^{(n+1)} = X_k^{(n)} + a^{(n)} \left(\frac{\widetilde{f}(X^{(n)} + \Delta^{(n)} c^{(n)}) - \widetilde{f}(X^{(n)} - \Delta^{(n)} c^{(n)})}{\Delta_k^{(n)} c^{(n)}} \right), \tag{3}$$

for $k = 1, \ldots, d$, where $\Delta^{(n)} \in \mathbb{R}^d$ is a vector of d mutually independent mean-zero random

variables $\Delta^{(n)} = (\Delta_1^{(n)}, \Delta_2^{(n)}, \ldots, \Delta_d^{(n)})$ satisfying some moment conditions and independent of $X^{(1)}, X^{(2)}, \ldots, X^{(n)}$. For the numerical studies in [23], $\Delta_k^{(n)}$ are taken to be Rademacher random variables, i.e., $\Delta_k^{(n)} = \pm 1$ with probability 1/2 each. Spall [23] proves that the iterates converge to x^* almost surely, and based on some numerical examples concludes that the asymptotic mean-squared-error (MSE), $\mathbb{E}[X^{(n)} - x^*]^2$ tends to be smaller in the SPSA algorithm than the KW-algorithm, with the ratio of the two becoming smaller as the dimension of the problem (d) gets larger.

Adaptive stochastic approximation algorithms. The sensitivity of stochastic approximation algorithms to the specification of the tuning sequences is well documented; cf. [2], [3], [4, §VIII.5a], [19], [24]. But, to the best of our knowledge, Kesten [14] seems to be the only paper to suggest a procedure to adapt the tuning sequence. In particular, he proposes using the same value of $a^{(n)}$ until the sign of $X^{(n)} - X^{(n-1)}$ differs from the sign of $X^{(n-1)} - X^{(n-2)}$. This fix attempts to address the rate degradation problem, but at the same time hinges on the value of $a^{(1)}$ chosen sufficiently "large" relative to the magnitude of the gradient. Abdelhamid [1] gives formulas for the optimal values of the constants in the sequences but these formulas require prior knowledge of the value of the first, second and third derivatives of the unknown function which are not generally available; he does not prescribe adaptive adjustments. In another variation of stochastic approximation algorithms, Polyak [21] presents an RM-type algorithm and solves the rate degradation problem asymptotically; by using a "larger" $\{a^{(n)}\}$ sequence that is fixed at the beginning of the algorithm and averaging the iterates. Even though the new $\{a^{(n)}\}$ sequence converges to zero slower, and there are asymptotic convergence guarantees, if the constant in this sequence is not chosen appropriately, the iterates do not show convergent behavior for a very large number of iterations (see [9]). Similarly, the algorithm introduced by Andradóttir [3] does not adaptively adjust the constants in the sequences. The RM-type algorithm she introduces in [3] uses two function evaluations at each iteration, and calculates a "scale-free" gradient estimate but does not address the long oscillatory period problem stemming from an $\{a^{(n)}\}$ sequence that is "too large."

Remainder of the paper. Section 2 introduces the scaling and shifting ideas to adaptively adjust the tuning sequences in multiple dimensions; the full algorithm description is given in Appendix A. The MATLAB implementation can be downloaded from `<www.columbia.edu/∼mnb2/broadie/research.html>`. Section 3 includes the test problems and finite-time performance comparisons of the scaled-and-shifted algorithms with their non-adaptive counterparts. Concluding remarks are given in Section 4.

2 Scaled-and-Shifted Stochastic Approximation Algorithms

One of the main factors influencing the performance of the Robbins-Monro algorithm is the choice of the parameters $\alpha \in \mathbb{R}$ and $\beta \in \mathbb{Z}$ in the specification of the step-size sequence $a^{(n)} = \alpha/(n + \beta)$.

The performance of the Kiefer-Wolfowitz and SPSA algorithms that use finite-difference estimates of the gradient hinges on the choice of the parameter $\gamma \in \mathbb{R}$ in the specification of the sequence $c^{(n)} = \gamma/n^{1/4}$, in addition to the two parameters α and β. In this section, we first describe in more detail performance issues related to the choice of these constants, and then explain how to adaptively adjust these parameters in multidimensional settings.

In their original paper, Kiefer and Wolfowitz [15] argue that it may be too restrictive to impose global assumptions on the unknown function $f(\cdot)$ and it suffices to impose these functional assumptions on a compact set $I_0 \subset \mathbb{R}^d$ which is known to contain x^*, the point of optimum. In this paper, iterates of RM, KW and SPSA algorithms are truncated in a similar manner.

2.1 Performance issues and general preliminaries

Performance issues Three possible problems can arise due to "poor" specification of the parameters in the tuning sequences (see also [9] for further discussion). The first issue arises from the step-size sequence, $\{a^{(n)}\}$ being "too small" relative to the gradient. This results in degradation of the convergence rate of the algorithm (see [19] for a simple illustration). The second issue is a long "oscillatory period" which is due to the $\{a^{(n)}\}$ sequence being "too large" relative to the gradient. That is, one may observe a large number of iterates that "bounce" back and forth on the boundaries of the truncation region. Finally, if the $\{c^{(n)}\}$ sequence is "too small" relative to the ambient noise level affecting function evaluations, the gradient estimates become very noisy. This causes the iterates to move in random directions governed purely by noise, resulting in poor finite-time performance.

Preliminaries In this section, we extend the core ideas presented in [9] to multidimensional setting. Before providing details, we note that if the objective function has different characteristics along different dimensions, the finite-time algorithm performance can be improved by using different tuning sequences along each dimension (see Section 2.4 in [8]). We assume the knowledge of an initial multidimensional interval $I^{(0)} = [l_1, u_1] \times [l_2, u_2] \times \ldots \times [l_d, u_d]$, which is known to contain x^*. (Other choices of this domain are possible but for simplicity and concreteness we restrict attention to a hyper-rectangle.) At each step n, the iterates are projected onto the *truncation region* $I^{(n)}$. Since the KW-algorithm requires an evaluation of the function at $X^{(n)}$ and $X^{(n)} + c^{(n)}$, we set the truncation region to $I^{(n)} := [l_1, u_1 - c_1^{(n)}] \times [l_2, u_2 - c_2^{(n)}] \times \ldots \times [l_d, u_d - c_d^{(n)}]$ and this guarantees all function evaluations are within the initial region $I^{(0)}$. The truncation region for algorithms that use two-sided finite-difference approximation of the gradient (such as SPSA) will be taken to be $I^{(n)} = [l_1 + c_1^{(n)}, u_1 - c_1^{(n)}] \times [l_2 + c_2^{(n)}, u_2 - c_2^{(n)}] \times \ldots \times [l_d + c_d^{(n)}, u_d - c_d^{(n)}]$. For RM-type algorithms $I^{(n)} = I^{(0)}$ for all $n = 1, 2, \ldots$ since the function is evaluated only at $X^{(n)}$ at every iteration.

For concreteness, we explain how to scale and shift the tuning sequences using the scaled-and-

shifted KW-algorithm (SS-KW for short). With the same notation as in (1), the SS-KW algorithm in multiple dimensions uses the recursion:

$$X_k^{(n+1)} = \Pi_{I^{(n+1)}} \left(X_k^{(n)} + a_k^{(n)} \frac{\widetilde{f}(X^{(n)} + c_k^{(n)} e_k) - \widetilde{f}(X^{(n)})}{c_k^{(n)}} \right), \quad \text{for all } k = 1, \ldots, d \text{ and } n = 1, 2, \ldots \tag{4}$$

Here $\Pi_{I^{(n+1)}}$ is the Euclidean projection operator with respect to the truncation region $I^{(n+1)}$. The scaled-and-shifted Robbins-Monro (SS-RM) algorithm uses only the adaptations of the $\{a^{(n)}\}$ sequence. The scaled-and-shifted SPSA algorithm (SS-SPSA) adapts the tuning sequences exactly as in the SS-KW algorithm, the only difference being the gradient estimate is two-sided as discussed above.

In what follows we describe the key building blocks of our proposed class of adaptive algorithms deferring full details of the algorithms to the Appendix.

2.2 Scaling up the $\{a^{(n)}\}$ sequence

For flat functions, i.e., functions where the magnitude of the gradient is "small," one needs to choose a "big enough" constant α_k in the $a_k^{(n)} = \alpha_k/(n + \beta_k)$ sequence in order to avoid degraded convergence rates and achieve the theoretically optimal convergence rates. To illustrate this with an example, we set $\widetilde{f}(x_1, x_2) = -0.001(x_1^2 + x_2^2 + \varepsilon)$ where ε is assumed to be a standard Gaussian random variable. The tuning sequences of the KW-algorithm are set as $a_k^{(n)} = 1/n$, $c_k^{(n)} = 1/n^{1/4}$ for $k = 1, 2$. Panel (a) of Figure 1 shows a typical sample path of the iterates of KW-algorithm under this setting. As seen from the figure, even iterate 5000 is still very far from x^*.

In order to prevent rate degradation due to the "too small" value of α, we scale up α_k in $a_k^{(n)} = \alpha_k/(n + \beta_k)$, and by doing so we force the iterates to "oscillate" between boundaries of the truncation region along every dimension; the meaning of this will be explained shortly. The number of oscillations h_0, is a parameter of the algorithm. In numerical examples, we use $h_0 = 4$ as the default value.

A *boundary hit* is registered when the iterate hits an *admissible boundary* along every dimension at least once. This constitutes an oscillation. The definition of an admissible boundary depends on the position of the iterate and the truncation region. Along dimensions where the iterate is inside the truncation region, both lower and upper boundaries are considered as admissible boundaries. If the iterate is at the boundary of the truncation region along one dimension, then the opposite boundary is admissible along that dimension, i.e., if it is at the lower boundary, say along dimension k, then the admissible boundary along that dimension is $u_k - c^{(n)}$ and vice versa.

In each iteration, the gradient $\widehat{\nabla}\widetilde{f}_k(X^{(n)}) := \left(\widetilde{f}(X^{(n)} + c_k^{(n)} e_k) - \widetilde{f}(X^{(n)})\right)/c_k^{(n)}$ is estimated for

every $k = 1, 2, \ldots, d$. Then for each dimension where the iterate has not hit an admissible boundary yet, a scale-up factor α'_k is calculated such that the iterate would hit the admissible boundary along that dimension, if the scaled up sequence were to be used. In other words, the value of α'_k is such that $X_k^{(n)} + \alpha'_k a_k^{(n)} \widehat{\nabla} \widetilde{f}(X^{(n)})$ hits the admissible boundary. In order to prevent scaling up the $\{a_k^{(n)}\}$ sequences too much due to a noisy gradient estimate, an upper bound on the scale up per iteration is imposed using the parameter φ_a (default value is 10). So, at the end of the iteration, we set $X^{(n+1)} = \Pi_{I^{(n+1)}}(X_k^{(n)} + \alpha'_k a_k^{(n)} \widehat{\nabla} \widetilde{f}(X^{(n)}))$ and $\alpha_k \leftarrow \alpha_k \max(1, \min(\alpha'_k, \varphi_a))$. (See step 2.b in Appendix A for specifics.) This first phase of the algorithm is called the "scaling phase."

Panel (b) in Figure 1 show iterates 50, 500 and 5000 from the SS-KW algorithm under the same setting as in panel (a). The adaptive SS-KW algorithm scales up the $\{a_k^{(n)}\}$ sequences into $a_1^{(n)} = 4032/n$ and $a_2^{(n)} = 4031/n$ and this scale-up in the sequences result in much faster convergence to $x^* = (0, 0)$.

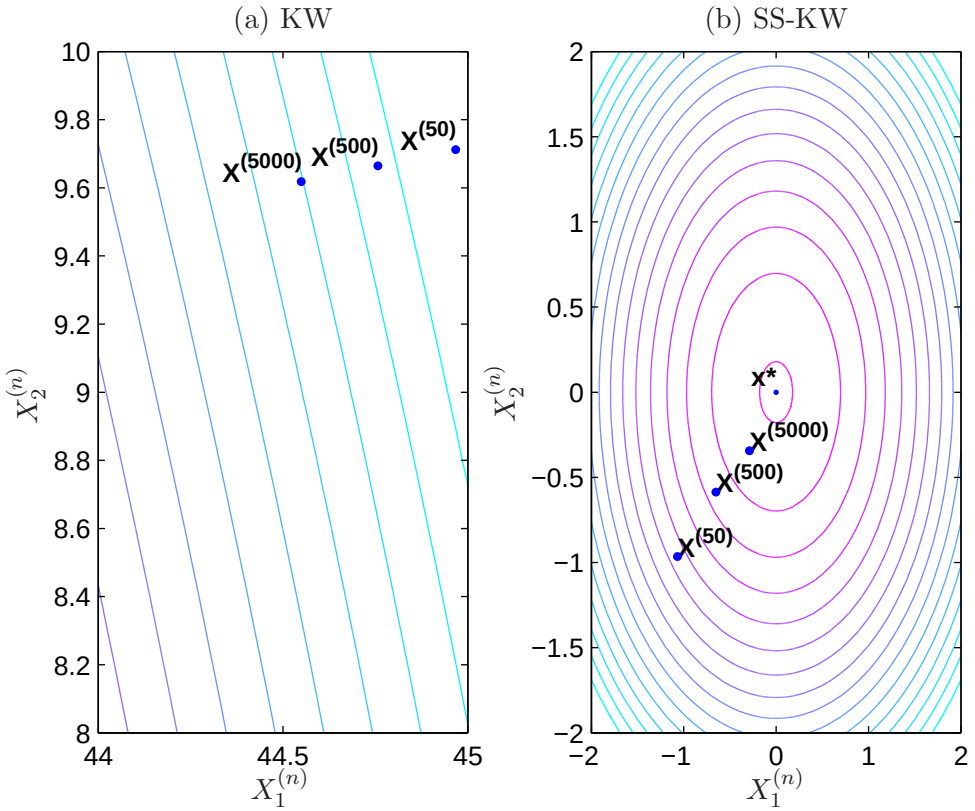

Figure 1: Scaling up $\{a^{(n)}\}$ sequences. Panel (a) illustrates a sample path of the KW-algorithm for the function $f(x_1, x_2) = -0.001(x_1^2 + x_2^2)$ starting from $X^{(0)} = (45.3, 9.8)$. As the magnitude of the gradient of this function is very small, using $a_k^{(n)} = 1/n$ for $k = 1, 2$ results in a degraded convergence rate. On the other hand, the SS-KW algorithm, shown in panel (b) scales up the sequences to $a_1^{(n)} = 4032/n$ and $a_2^{(n)} = 4031/n$ and significantly improves convergence.

2.3 Shifting the $\{a^{(n)}\}$ sequence

The second issue that needs to be addressed is a long oscillatory period caused by the $\{a^{(n)}\}$ sequence being "too large" relative to the magnitude of the gradient. Note that a "large" $\{a^{(n)}\}$ sequence may be a result of excessive scaling up in the first phase, or simply due to the original specification of $a_k^{(n)} = 1/n$ not being suitable for the steepness in the underlying function. (If it is the latter, then the $\{a_n\}$ sequence is typically not scaled up during the first phase.)

In order to show the long oscillatory period problem on an example, we set $\widetilde{f}(x_1, x_2) = -(x_1^4 + x_2^4 + \varepsilon)$ where ε is assumed to be a standard Gaussian random variable. The tuning sequences are initiated as $a_k^{(n)} = 1/n$, $c_k^{(n)} = 1/n^{1/4}$ for $k = 1, 2$. Panel (a) of Figure 2 shows a typical sample path of the iterates of KW-algorithm under this setting. As seen from the figure, the iterate moves back and forth between two corners of the truncation region and this behavior is observed for $n = 50$, $n = 500$ and even $n = 5000$. Actually, under this setting, the iterates oscillate between the corners of the truncation region for 6665 iterations before starting to converge to $x^* = (0, 0)$.

In order to prevent long periods of oscillation between opposite boundaries of the truncation region, an index shift is done in the step-size sequence $\{a^{(n)}\}$. If the k^{th} coordinate of the iterate $X^{(n)}$ is at the boundary of the truncation region, and if $X_k^{(n)} + a_k^{(n)}\widehat{\nabla f}(X^{(n)})$ is beyond the opposite boundary along the same dimension, then the smallest integer β_k' is calculated such that using $a_k^{(n+\beta_k')}$ at iteration n keeps the next iterate within the truncation region along dimension k. For instance, suppose $X^{(n)} = u_k - c_k^{(n)}$ ($X^{(n)}$ is at the upper boundary of the truncation region along dimension k), and $X_k^{(n)} + a_k^{(n)}\widehat{\nabla}\widetilde{f}(X^{(n)}) < l_k$ (the initial calculation for the next iterate is beyond the lower boundary along the same dimension), then $\beta_k' = \inf\{s : X_k^{(n)} + a_k^{(n+s)}\widehat{\nabla}\widetilde{f}(X^{(n)}) \geq l_k\}$. After β_k' is calculated, the actual shift is set to be the minimum of β_k' and υ_k^a, a parameter of the algorithm imposing an upper bound on the shift, i.e., $\beta_k \leftarrow \beta_k + \min(\beta_k', \upsilon_k^a)$ and the step-size sequence becomes $a_k^{(n)} = \alpha_k/(n + \beta_k)$. The main aim of this upper bound is to prevent overly large shifts due to very noisy estimates of the gradient. If the upper bound is in effect, namely, $\min(\beta_k', \upsilon_k^a) = \upsilon_k^a$, then the value of υ_k^a is doubled. In the numerical tests, we have set $\upsilon_k^a = 10$ for all $k = 1, \ldots, d$ as the default initial value. We call this second phase of the algorithm the "shifting phase."

Applying the shifting idea to the illustrative example above results in $a_k^{(n)} = 1/(n + 9730)$ for $k = 1, 2$ on the same sample path shown in panel (a) of Figure 2. The oscillatory period length decreases to only 14 iterations in SS-KW (down from 6665 in KW). The resulting sample path is shown in panel (b) of the same figure.

A natural question that arises is whether to shift the $\{a^{(n)}\}$ sequence or scale it down to get a smaller magnitude sequence. The answer is that by shifting instead of scaling down one preserves the "energy" in the sequence. We illustrate this with an example. Suppose, the step-size sequence

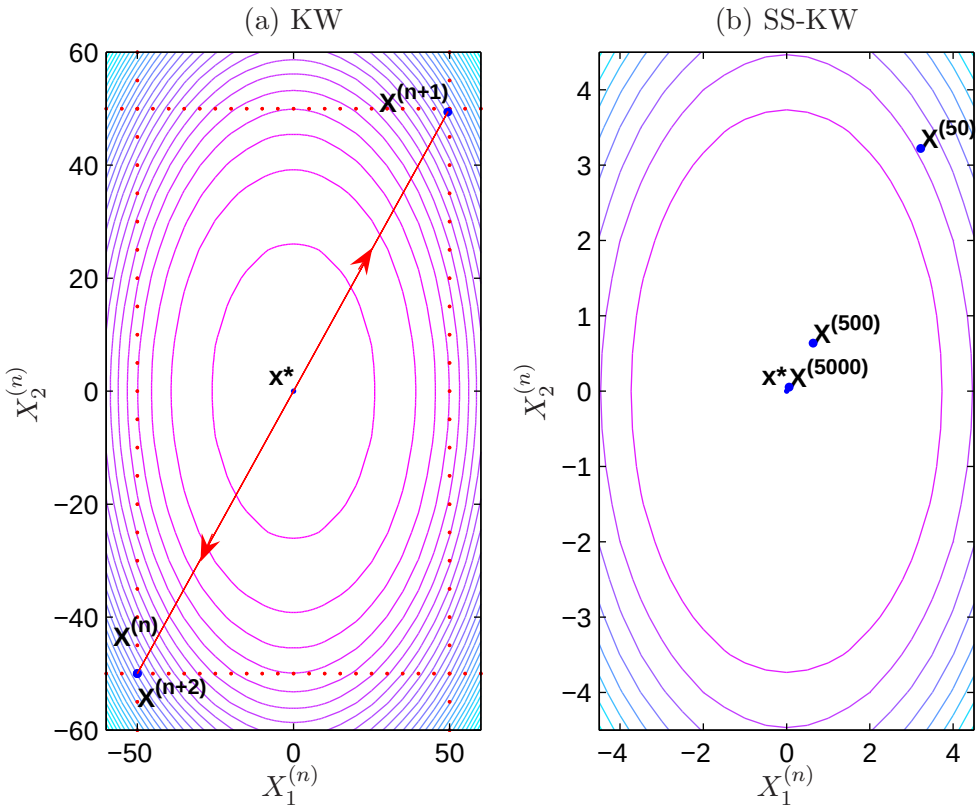

Figure 2: Shifting $\{a^{(n)}\}$ sequences illustrated for the function $f(x_1, x_2) = -(x_1^4 + x_2^4)$. Panel (a) illustrates the oscillation between corner points of the truncation function when the magnitude of the gradient is very large. The boundaries of the initial truncation interval, $I^{(0)} = [-50, 50]^2$ are shown with red dots in Panel (a). The iterates of the KW-algorithm exhibit the illustrated oscillatory behavior for $n = 50$, $n = 500$ and even $n = 5000$. The sample path of iterates of the SS-KW algorithm shown in panel (b) shows much smoother convergence behavior after shifting the sequences to $a_k^{(n)} = 1/(n + 9730)$, for $k = 1, 2$. The length of the oscillatory period in SS-KW algorithm is only 14 iterations (down from 6665 iterations in KW-algorithm).

is initialized as $a^{(n)} = 1/n$ which implies $a^{(2)} = 1/2$ whereas in fact $a^{(2)} = 1/20$ is the value necessary to stay within the truncation region. Proceeding as prescribed above and shifting the $\{a^{(n)}\}$ sequence by 18, we get $a^{(n)} = 1/(18 + n)$ which gives $a^{(3)} = 1/21$. If instead the $\{a^{(n)}\}$ sequence is scaled down by a factor of 10 then $a^{(n)} = 1/10n$ and the next step-size value would be $a^{(3)} = 1/30$ which is smaller than $1/21$. Preserving "energy" in the $\{a^{(n)}\}$ sequence and not decreasing it more than necessary is important to avoid degrading the convergence rate. The intuition is made precise in [9].

2.4 Scaling up the $\{c^{(n)}\}$ sequence

This adaptive adjustment is only pertinent to stochastic approximation algorithms that use finite-difference estimates of the gradient, e.g., KW, SPSA, etc. as reviewed in the introduction. For

these KW-type algorithms poor finite-time performance may be due to a $\{c^{(n)}\}$ sequence whose magnitude is "too small." This, in turn, affects two conflicting error terms in gradient estimation. To explain this, let us assume $\widetilde{f}(x) = f(x)+\varepsilon$ for some random variable ε. Then the finite-difference estimate of the gradient along dimension k at iteration n can be written as

$$\begin{aligned} \widehat{\nabla f}(X^{(n)}) &= \frac{f(X^{(n)} + c_k^{(n)} e_k) - f(X^{(n)})}{c_k^{(n)}} + \frac{\varepsilon_1 - \varepsilon_2}{c_k^{(n)}} \\ &= f'_k(X^{(n)}) + O(c_k^{(n)}) + \frac{\varepsilon_1 - \varepsilon_2}{c_k^{(n)}} \end{aligned} \tag{5}$$

where the second equality follows simply by Taylor expansion, and for a sequence $\{b_n\}$ we write $d_n = O(b_n)$ if $\limsup_{n\to\infty} d_n/b_n < \infty$. The magnitude of the $\{c^{(n)}\}$ sequence should be chosen so that the finite-difference estimation bias (the middle term on the RHS of (5)), which is of order $c^{(n)}$, is balanced with the error due to estimation noise (the last term on the RHS of (5)), which is of order $1/c^{(n)}$. If $\{c^{(n)}\}$ is chosen too small, then the estimation noise becomes the dominant term in the gradient estimate. Panel (a) in Figure 3 shows a sample path of KW-algorithm for $\widetilde{f}(x_1, x_2) = 1000(\cos(\pi x_1/100) + \cos(\pi x_2/100) + \varepsilon)$ where ε is standard Gaussian random variable. With $c_k^{(n)} = 1/n^{1/4}$, for $k = 1, 2$ (and $a_k^{(n)} = 1/n$, for $k = 1, 2$ as before) iterates of the KW-algorithm move in random directions governed mainly by the noise. On the illustrated sample path, the iterates get closer to $x^* = (0, 0)$ from iteration 50 to 500; but then moves away from it from iteration 500 to 5000.

In the SS-KW algorithm, during the scaling and the shifting phases, the $\{c^{(n)}\}$ sequences are adaptively adjusted to the noise level. If the iterate $X^{(n)}$ is at the boundary of the truncation region along dimension k, and if the gradient estimate along the same dimension is pointing outward of the truncation region, then the $\{c_k^{(n)}\}$ sequence is scaled up by a factor of γ_0 in the $c_k^{(n)} = \gamma_k/n^{1/4}$ sequence, where γ_0 is a parameter of the algorithm. In order to prevent the magnitude of the $\{c^{(n)}\}$ sequence from growing too large, an upper bound that depends on the width of the initial region is used. In particular, we require $c_k^{(n)} \le c^{(0)}(u_k - l_k)$ and once $\{c_k^{(n)}\}$ achieves this upper bound, it is not scaled up any further. The default values for the two algorithm parameters are set to be $\gamma_0 = 2$ and $c^{(0)} = 0.2$.

Panel (b) in Figure 3 illustrates a sample path of the SS-KW algorithm in the same setting as panel (a). After the adaptations, the sequences become $c_1^{(n)} = 6.7/n^{1/4}$ and $c_2^{(n)} = 4/n^{1/4}$ ($a_1^{(n)} = 9.4/(n + 15)$, $a_1^{(n)} = 1.4/(n + 2)$). The scale-up in the $\{c_k^{(n)}\}$, $k = 1, 2$ sequences decreases the dominant error and improves the gradient estimate, which in turn improves the convergence behavior of the iterates.

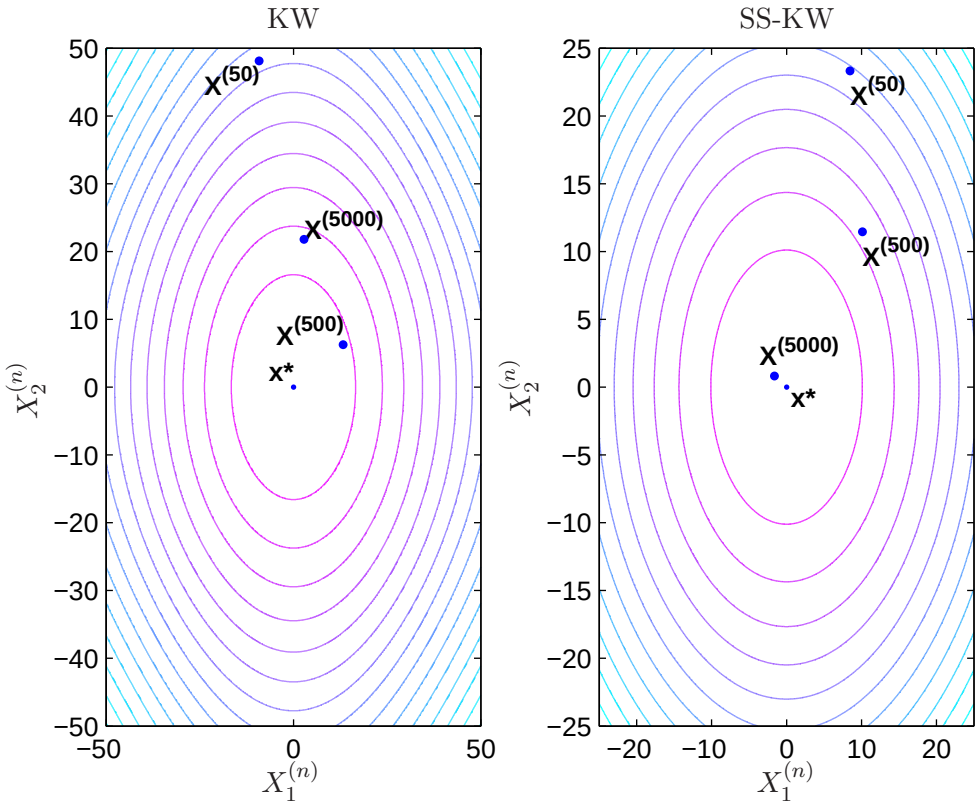

Figure 3: Scaling up $\{c^{(n)}\}$ sequences. Panel (a) plots the 50^{th}, 500^{th} and 5000^{th} iterates of KW-algorithm for the function $\widetilde{f}(x_1, x_2) = 1000(\cos(\pi x_1/100) + \cos(\pi x_2/100) + \varepsilon)$ where ε is standard Gaussian random variable. With $c_k^{(n)} = 1/n^{1/4}$, for $k = 1, 2$ iterate 500 is closer to $x^* = (0, 0)$ than iterate 50, but then iterate 5000 is further. Given the contours of the underlying function, the error in gradient estimates is dominated by the noise in function evaluations rather than the finite-difference estimation error. Panel (b) shows the sample path for the SS-KW algorithm under the same setting, which decreases the noise in the gradient estimates by scaling up the sequences to $c_1^{(n)} = 6.7/n^{1/4}$ and $c_2^{(n)} = 4/n^{1/4}$.

3 Numerical Examples

Practical applications of stochastic approximation algorithms requires the user to manually or experimentally adjust tuning sequences to underlying problem-specific features. This process typically requires some pre-algorithm experimentation with the objective function followed by ad-hoc sequence adjustments prior to the application of the stochastic approximation procedure. The cost of any extra function evaluations is typically not taken into account in the overall performance evaluation of the algorithm.

In this section, we apply the scaling and shifting ideas introduced earlier and test their efficiency. We first consider scaled and rotated multidimensional quadratic functions in §3.1. With the scaling and the rotation, these multidimensional quadratic functions replicate various prototypical behavior of objective functions commonly found in variety of settings. We then consider more realistic

problems arising in several application domains. We primarily focus on the finite-time performance of standard stochastic approximation algorithms, such as RM, KW and SPSA, and compare them to scaled-and-shifted versions (denoted as SS-RM, SS-KW and SS-SPSA, respectively).

Remark 1 *Unless otherwise noted, in all the numerical experiments, we initialize the* $\{a_k^{(n)}\}$ *sequences as* $a_k^{(n)} = 1/n$ *and the constant* γ_k *in the* $\{c_k^{(n)}\}$ *sequence is set to* $\gamma_k = (u_k - l_k)/20$ *along every dimension* $k = 1, \ldots, d$ *for the scaled-and-shifted versions of the algorithms. Recall that* u_k *and* l_k *denote the upper and lower boundaries of initial interval along direction* k*. Since the same tuning sequence is used along every dimension in the RM, KW and SPSA algorithms, the initial choice for the sequences in these algorithms are* $a^{(n)} = 1/n$ *and* $c^{(n)} = \gamma/n^{1/4}$ *with* $\gamma = \min_{1 \le k \le d}(u_k - l_k)/20$*. With this choice of sequences, the optimal convergence rate for the MSE of RM-type algorithms is* $O(1/n)$*, and for KW-type algorithms is* $O(1/\sqrt{n})$*.*

We consider two main performance metrics. The first is the MSE at the n^{th} iteration, namely, $\mathbb{E}(X^{(n)} - x^*)^2$, and the associated rate at which the MSE converges to zero. Specifically, this rate refers to the slope of the $\log(MSE)$; the minimal rate therefore being $-1/2$. To estimate the latter, we omit the initial 10% of the iterations. For instance, the convergence rate for an algorithm with 20,000 iterations is estimated using iterations 2,000 to 20,000. The second measure of performance is the estimate of the average error in function values, $|f(x^*) - \mathbb{E}f(X^{(n)})|$, at the terminal point and the associated convergence rate. In order to estimate this rate, we again ignore the initial 10% of iterations and after this take every one-fiftieth iteration, and finally average the error values over independent runs of the algorithms. For example, for an algorithm with 20,000 iterations, this convergence rate is calculated using average function error at iterations 2,000, 2,400, 2,800, ... , 20,000. Throughout the paper, the numbers in parenthesis are the standard error of the estimates.

We define the *oscillatory period length*, T, as the last iteration in which an iterate has moved from one boundary of the truncation region to the opposite boundary along any dimension. In other words, we define $T = \max_{1 \le k \le d} T_k$ where

$$T_k = \sup \Big\{ n \ge 2 \quad : \quad (X_k^{(n)} = u_k - c_k^{(n)} \text{ and } X_k^{(n-1)} = l_k + c_k^{(n-1)}) \text{ or } (X_k^{(n)} = l_k + c_k^{(n)} \text{ and } X_k^{(n-1)} = u_k - c_k^{(n-1)}) \Big\}.$$

In order to compute more reliable estimates for the asymptotic convergence rates in cases where the iterates oscillate persistently, the number of iterations that we ignore in rate calculations is the maximum of the oscillatory period among all paths plus 10% of the total number of iterations. In these instances the median oscillatory period length is also reported.

3.1 Quadratic Functions

Consider the function $\widetilde{f}(x) = -(Kx)'Ax + \sigma Z$ where Z is a standard normal random variable. We assume that the rotation matrix A is of the following form:

$$A = \begin{pmatrix} 1 & \rho & \rho^2 & \dots & \rho^{d-1} \\ \rho & 1 & \rho & \dots & \rho^{d-2} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \rho^{d-1} & \rho^{d-2} & \rho^{d-3} & \dots & 1 \end{pmatrix},$$

for some $\rho \in (-1, 1)$. We further assume the scaling matrix K is a diagonal matrix with $K(i,i) = k_0 k_i$ for some $k_0, k_i \in \mathbb{R}$ and set $k_0 = k_i = 1$ for all $i = 1, 2, \dots, d$ unless otherwise noted. The function $f(x) := \mathbb{E}\widetilde{f}(x) = -(Kx)'Ax$ has a unique maximum at $x^* = (0, \dots, 0) \in \mathbb{R}^d$ provided that the Hessian matrix $(AK + KA)$ is positive definite. For the RM and SS-RM algorithms, we assume $\widetilde{g}(x) = (AK + KA)'x + \sigma Z$. (Note that $g(x) := \mathbb{E}\widetilde{g}(x) = (AK + KA)'x$ is the gradient of $f(x)$). Under the assumption that $(AK + KA)$ is positive definite, the unique zero crossing of the function is also at $x^* = (0, \dots, 0) \in \mathbb{R}^d$.

We apply the RM and SS-RM algorithms to find the zero-crossing of the function $g(x)$, and KW, SPSA, SS-KW and SS-SPSA algorithms to find the maximizer of the function $f(x)$ for various A and K matrices and noise levels. Five different problem settings that we consider are given in Table 1. In all instances, the initial truncation region, I_0, is assumed to be of the form $I^{(0)} = [-a, a]^d$ where d is the dimension of the problem.

Table 1: Parameters for the scaled and rotated test functions. For the RM and SS-RM algorithms, the function is $g(x) = (AK + KA)'x$ where for the other algorithms (KW, SS-KW, SPSA and SS-SPSA), it is $f(x) = -(Kx)'Ax$.

Problem #	d	ρ	k_0	k_1	k_2	σ	a
1	2	0	1	100	0.01	0.01	1
2	3	0.1	1000	1	1	10	1
3	4	0.5	0.01	1	1	0.001	1
4	5	0.5	0.1	1	1	10	100
5	10	0.5	0.1	1	1	0.05	1

We run each of the algorithms for a total of 20,000 function evaluations. For a d-dimensional problem the RM and the SS-RM algorithms have 20,000 iterations, where the KW and the SS-KW algorithms have $\lfloor 20{,}000/(d+1) \rfloor$ iterations. Since at every iteration the SPSA and SS-SPSA algorithms require only two function evaluations independent of the dimension of the problem, these algorithms run for 10,000 iterations. In all numerical tests, the starting point is assumed to be uniformly distributed on the initial hypercube, $I^{(0)}$. The averages are estimated using 1000 independent replications of the algorithms.

Problem 1: The first instance is an example of an objective function which has different characteristics along its two dimensions. The values $k_1 = 100$ and $k_2 = 0.01$ cause the objective function to be very steep along the first dimension, and very flat along the second. As seen in Table 2, the RM-algorithm suffers from a degraded convergence rate estimate with respect to the optimal rate of $O(1/n)$. This is caused by the $\{a_2^{(n)}\}$ sequence being too small relative to the magnitude of the gradient along the second dimension. The SS-RM recovers the optimal convergence rate of both performance measures. The KW-algorithm also suffers from a degraded convergence rate estimate for the iterates, but because the second dimension does not influence the function values significantly, we do not observe a significant rate degradation in terms of function values. However, the SS-KW algorithm improves performance in both measures. Looking at the last two rows of the table, we observe that scaling and shifting does not improve the degraded convergence rate of SPSA. For the SPSA and SS-SPSA algorithms, the iterates quickly converge to the point of optimum along the first dimension and then the error both in the iterates and the function values are dominated by the behavior along the second dimension. The SPSA suffers from the small $\{a_2^{(n)}\}$ as the RM and KW algorithms. The SS-SPSA cannot improve the degraded convergence rate since the scaling and shifting is done based on the gradient estimates along the diagonal, not the individual dimensions separately. Because the difference in function values along any diagonal is dominated by the first dimension, the tuning sequences are not adjusted properly by this scaling and shifting approach.

Table 2: Results for problem 1. This table provides the values for $\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$ and $f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$ after 20,000 function evaluations and convergence rate estimate. (The numbers in parenthesis are the standard errors of estimates.)

Algorithm	$\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$	Conv. Rate
RM	0.22 (0.006)	-0.04 (2×10^{-05})	0.002 (6×10^{-05})	-0.04 (2×10^{-05})
SS-RM	9.2×10^{-05} (6×10^{-06})	-1.00 (0.06)	9.2×10^{-07} (6×10^{-08})	-0.99 (0.07)
KW	0.23 (0.007)	-0.04 (0.003)	0.005 (8×10^{-05})	-0.36 (0.01)
SS-KW	0.04 (0.003)	-0.36 (0.06)	0.004 (4×10^{-05})	-0.48 (0.01)
SPSA	0.27 (0.007)	-0.08 (0.006)	0.003 (7×10^{-05})	-0.08 (0.006)
SS-SPSA	0.14 (0.004)	-0.08 (0.001)	0.001 (4×10^{-05})	-0.09 (0.001)

Problem 2: The second problem instance is an example of a very steep objective function with $k_0 = 1000$. The RM, KW and SPSA algorithms suffer from long oscillatory periods, a problem mitigated by shifting the $\{a^{(n)}\}$ sequence in the SS-RM, SS-KW and SS-SPSA algorithms. The median oscillatory period for the RM is 1148 and is decreased to 31 in SS-RM algorithm. The corresponding values are 1921 and 180 for the KW and SS-KW, respectively. Scaling and shifting brings the median oscillatory period length from 5977 in the SPSA to 60 in the SS-SPSA algorithm. The benefit of the shorter oscillatory period can be seen in the MSE and the function error values given in the second and fourth columns of Table 3. In order to better estimate the convergence rate for the RM, KW and SPSA algorithms (shown in bold in the table) we run 250 independent

replications of these algorithms for 150,000 iterations. Then we ignore the oscillatory period plus 10% of the total number of iterations (15,000 iterations in this example) and use the remaining iterations to estimate the convergence rates. The maximum oscillatory period lengths among the 250 independent runs are 1154 for the RM-algorithm, 2456 for the KW and 7438 for the SPSA algorithms. So in order to estimate the convergence rates, as prescribed before, we use iterations 16,154 to 150,000 in the RM-algorithm, 17,456 to 150,000 in the KW and 22,438 to 150,000 for the SPSA algorithms. As seen from these estimates, all the algorithms eventually achieve the theoretical convergence rates, but the original algorithms start showing convergent behavior only after a long oscillatory period.

Table 3: Results for problem 2. This table provides the values of $\widehat{\mathbb{E}}\|X^{(n)}-x^*\|^2$ and $f(x^*)-\widehat{\mathbb{E}}f(X^{(n)})$ after 20,000 function evaluations and convergence rate estimates. Because of the long oscillatory period, the RM, KW and SPSA algorithms are run for 150,000 iterations to better estimate the convergence rates; the numbers in parenthesis are the standard errors of the estimates. The last two columns contain oscillatory period length statistics and show the shorter oscillatory period of scaled-and-shifted algorithms. $\mathrm{T}_{\mathrm{med}}$ is the median oscillatory period length calculated using the independent runs of the algorithms, where $\mathrm{T}_{\max}$ is the maximum oscillatory period length among all runs.

Algorithm	$\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$	Conv. Rate	$\mathrm{T}_{\mathrm{med}}$	$\mathrm{T}_{\max}$
RM	1.6×10^{-06} (4×10^{-08})	-1.02 (0.005)	0.002 (4×10^{-05})	-1.06 (0.04)	1154	1148
SS-RM	1.6×10^{-06} (4×10^{-08})	-0.99 (0.002)	0.001 (4×10^{-05})	-0.98 (0.02)	31	56
KW	0.13 (0.004)	-0.52 (0.004)	138 (4)	-0.51 (0.05)	1914	2456
SS-KW	0.11 (0.004)	-0.45 (0.004)	119 (4)	-0.45 (0.02)	180	1128
SPSA	0.06 (0.002)	-0.61 (0.002)	59 (2)	-0.61 (0.02)	6014	7438
SS-SPSA	$8.0\times10^{-04}(3\times10^{-05})$	-0.48 (0.01)	0.78 (0.02)	-0.50 (0.03)	60	935

Problem 3: The third problem instance is an example of a very flat objective function. The RM, KW and SPSA algorithms suffer from a degraded convergence rate as seen in Table 4. The scaled-and-shifted versions recover the optimal convergence rates, while the improved finite-time behavior, due to scaling up the $\{a^{(n)}\}$ sequences, is also observed in the MSE and function error values.

Problem 4: The input for the scaled-and-shifted stochastic approximation algorithm is an initial interval that is known to contain the point of optimum. In some cases, lack of information about the underlying function may lead to a "wide," say $[-100, 100]$ initial interval, and this example illustrates the performance of scaling and shifting in this case. As seen in Table 5, scaled-and-shifted versions improve the finite-time behavior with respect to all the measures given in the table.

Problem 5: The performance of scaled-and-shifted stochastic approximation algorithms is examined in the case of a ten-dimensional quadratic function. The benefits of scaling and shifting can be seen in Table 6: scaled and shifted stochastic approximation algorithms have smaller MSE and function error values, and improved convergence rate estimates.

Table 4: Results for problem 3. This table provides the values for $\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$ and $f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$ after 20,000 function evaluations and convergence rate estimates. (The numbers in parenthesis are the standard errors of the estimates.)

Algorithm	$\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$	Conv. Rate
RM	0.88 (0.01)	-0.03 (0.003)	0.007 (0.0001)	-0.05 (0.003)
SS-RM	3.4×10^{-06} (1×10^{-07})	-1.00 (0.04)	2.5×10^{-08} (1×10^{-09})	-1.00 (0.04)
KW	0.92 (0.01)	-0.04 (0.003)	0.008 (0.0001)	-0.05 (0.003)
SS-KW	0.05 (0.002)	-0.52 (0.05)	0.0004 (2×10^{-05})	-0.52 (0.04)
SPSA	0.66 (0.01)	-0.06 (0.003)	0.005 (8×10^{-05})	-0.09 (0.005)
SS-SPSA	0.01(0.0006)	-0.47 (0.04)	5.9×10^{-05} (3×10^{-06})	-0.48 (0.03)

Table 5: Results for problem 4. This table provides the values for $\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$ and $f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$ after 20,000 function evaluations and convergence rate estimates. (The numbers in parenthesis are the standard errors of the estimates.)

Algorithm	$\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$	Conv. Rate
RM	1475 (35)	-0.18 (0.004)	66 (1)	-0.20 (0.005)
SS-RM	4.6 (0.2)	-0.99 (0.03)	0.32 (0.02)	-1.00 (0.04)
KW	1913 (42)	-0.19 (0.004)	89 (2)	-0.22 (0.006)
SS-KW	25 (3)	-0.50 (0.03)	2.2 (0.1)	-0.50 (0.05)
SPSA	489 (13)	-0.33 (0.005)	20.2 (0.5)	-0.36 (0.007)
SS-SPSA	13(2)	-0.58 (0.09)	0.58 (0.08)	-0.58 (0.08)

Table 6: Results for problem 5. This table provides the values for $\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$ and $f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$ after 20,000 function evaluations and convergence rate estimates. (The numbers in parenthesis are the standard errors of the estimates.)

Algorithm	$\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}f(X^{(n)})$	Conv. Rate
RM	0.31 (0.005)	-0.18 (0.001)	0.01 (2×10^{-04})	-0.19 (0.001)
SS-RM	0.0002 (6×10^{-06})	-1.00 (0.03)	1.4×10^{-05} (4×10^{-07})	-1.00 (0.03)
KW	1.08 (0.02)	-0.18 (0.01)	0.08 (0.001)	-0.28 (0.03)
SS-KW	0.37 (0.008)	-0.44 (0.04)	0.03 (0.0008)	-0.46 (0.04)
SPSA	0.50 (0.009)	-0.28 (0.01)	0.02 (0.0004)	-0.33 (0.01)
SS-SPSA	0.07 (0.002)	-0.48 (0.05)	0.004 (0.0001)	-0.49 (0.04)

3.2 Multidimensional Newsvendor Problem

In this section, we illustrate the performance of the SS-KW algorithm on a multidimensional newsvendor problem introduced by Kim [16] (Section 2, page 159). A manufacturer produces q products using p different resources. He decides on a non-negative resource vector $x \in \mathbb{R}^p_+$ and then observes the random demand, $D \in \mathbb{R}^q_+$, for each of the products. Once the demand is known, the manager solves an optimization problem to determine the product mix to manufacture. Assuming $c \in \mathbb{R}^p_+$ is the cost vector for resources, the objective function of the manager is to maximize the expected profit from the best product mix, $f(x) = \mathbb{E}\widetilde{f}(x) = \mathbb{E}[v^T y^*(x, D)] - c^T x$. Here, y^* is

the solution of the optimization problem:

$$\begin{array}{lrcll} \max & v^T y & & & \\ s.t. & Ay & \leq & X & \text{(capacity constraints)} \\ & y & \leq & D & \text{(demand constraints)}, \\ & y & \geq & 0 & \end{array}$$

where $v \in \mathbb{R}_+^q$ is the vector of profit margins for each product and A is a $p \times q$ matrix whose $(i,j)^{th}$ component specifies the amount of resource i required to produce one unit of product j.

For our numerical experiments, we set $v = (6, 5, 4, 3, 2)$, c is a vector of ones and A is a matrix with all lower triangular entries equal to one and others zero. We assume the demand for the five products has a multivariate normal distribution with mean $(10, 15, 5, 8, 10)$, standard deviation $(5, 10, 2, 3, 6)$ and correlation matrix

$$\rho = \begin{pmatrix} 1 & -0.2 & 0.3 & 0.5 & 0.1 \\ -0.2 & 1 & -0.1 & -0.3 & -0.1 \\ 0.3 & -0.1 & 1 & 0.6 & 0.2 \\ 0.5 & -0.3 & 0.6 & 1 & 0.05 \\ 0.1 & -0.1 & 0.2 & 0.05 & 1 \end{pmatrix}.$$

In order to evaluate the function at any given resource level, we use 1000 independent draws of the demand and estimate the expected profit from the best product mix.

The solution of the linear optimization problem is given by $y_i^* = \min\{D_i, \min_{i \leq j \leq p}(X_j - \sum_{k=1}^{i-1} y_k^*)\}$ for $i = 1, \ldots, q$. The point of maximum of $f(\cdot)$ is estimated to be $x^* = (15, 30, 34, 41, 51)$ via an exhaustive grid search over integers, and the optimum objective function value is $f(x^*) = \$193.9$ estimated with a standard error of 0.2. The algorithms are set to run for 10,000 iterations and we use $15,000$ independent replications to estimate the MSE. The lower and upper bounds of the initial interval I_0 are set to be the resource levels required to meet the 30 and 99 percentile of the demand distribution, respectively. (The upper bound is (22, 61, 71, 86, 110) and the lower bound is (8, 18, 22, 29, 36).) Because no prior information about the objective function is assumed, the initial starting point of the algorithm, $X^{(1)}$, is set to be uniformly distributed between the upper and lower boundaries.

Table 7: Multidimensional newsvendor problem. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(10,000)} - x^*\|^2$, estimate of the MSE convergence rate, the terminal error in expected profit estimate, i.e., $f(x^*) - \widehat{\mathbb{E}}\widetilde{f}(X^{(10,000)})$, and the convergence rate of the error values.

	$\widehat{\mathbb{E}}\|X^{(10,000)} - x^*\|^2$	Conv. Rate	$f(x^*) - \widehat{\mathbb{E}}\widetilde{f}(X^{(10,000)})$	Conv. Rate
KW	123.3 (0.3)	-0.18 (0.0004)	\$5.89 (0.02)	-0.18 (0.002)
SS-KW	2.70 (0.03)	-0.59 (0.003)	\$0.15 (0.02)	-0.65 (0.08)

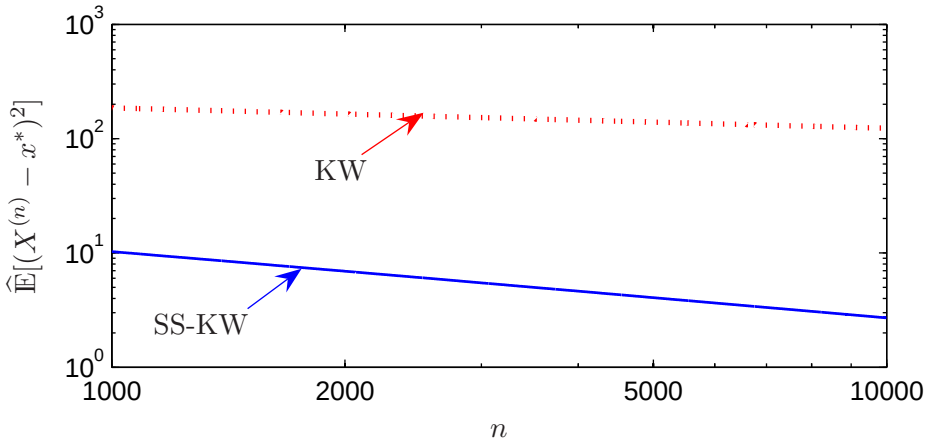

Figure 4: Multidimensional newsvendor problem. The log-log plot shows the improved convergence rate of the SS-KW algorithm over the KW-algorithm. The KW-algorithm converges at a rate estimate of -0.18 with standard error of estimate of 0.0004 where the SS-KWx algorithm improves this rate estimate to -0.59 with standard error of estimate of 0.003.

Figure 4 shows the improved convergence rate estimate for the MSE for the KW and the SS-KW algorithms, and exhibits the improvement in the convergence rate estimate as well as the MSE itself. Table 7 compares the two algorithms with respect to the two performance measures described before. The improved performance is achieved mainly by adjusting the $\{a^{(n)}\}$ sequences. The slow convergence of the KW-algorithm is caused by $\{a_k^{(n)}\}$ sequences being too small relative to the gradient, which leads to a degradation in the convergence. Scaling the constant in this sequence in the first phase of the SS-KW algorithm improves the performance; the estimates for median scale-up values for for the sequences $a_k^{(n)} = \alpha_k/(n+\beta_k)$ are $\alpha_1 = 54$, $\alpha_2 = 124$, $\alpha_3 = 189$, $\alpha_4 = 829$, $\alpha_5 = 286$. (The median shift values are $\beta_1 = 2$, $\beta_2 = 1$, $\beta_3 = 12$, $\beta_4 = 30$, $\beta_5 = 3$; the median scale-up values for the $\{c_k^{n)}\}$ sequences are 1.)

3.3 Ambulance Base Locations

This problem is introduced and described in detail by Pasupathy and Henderson [20]. The problem is to choose the best two base locations for two ambulances to minimize the long run average response time to emerging calls. We assume calls for an ambulance arrive according to a Poisson process at a constant rate $\lambda = 0.15$/hr. Independent of this Poisson arrival process, call locations are independent and identically distributed (i.i.d.) on the unit square $[0,1]^2$ (distances are measured in units of 30 kilometers), with density function proportional to $f(x,y) = 1.6 - (|x-0.8| + |y-0.8|)$. (Call location and arrival processes define the "call process," C.) The amount of time an ambulance spends at the call location (the service process, S) has normal distribution with mean $\mu_s = 45$ minutes and standard deviation $\sigma_s = 15$ minutes. (The original problem in [20] assumes gamma service distribution but we changed this assumption to a normal distribution since it is faster to

generate normally distributed random variables.) The lower tail of the service time distribution is truncated at 5 minutes. Each of the two ambulances has its own designated base. On the way to the call, an ambulance travels at a constant speed $\nu_f = 60$km/hr and at a constant speed $\nu_s = 40$km/hr on the way back. All travel is assumed to be in "Manhattan fashion," i.e., the distance between (x_1, y_1) and (x_2, y_2) is $|x_1 - x_2| + |y_1 - y_2|$.

When a call arrives, the closest free ambulance is directed to the call. If there is no available ambulance, the call is added to a queue which is handled in FIFO order. Once an ambulance arrives at the scene, it stays there during the service time and then is free for new calls. After the service is completed, if there are any calls waiting in the queue, the ambulance travels directly to the next call. If there are no calls waiting, the ambulance proceeds back to its base. If there is a new call arrival while an ambulance is on its way back to its base, the ambulance en route to its base can be re-routed to the call if it is the closest free ambulance at the time of call arrival. The response time is defined as the time from when a call is received until the arrival of the ambulance at the scene. The objective is to minimize the expected response time, taken with respect to the service time distribution, the call arrival process and call locations.

The objective function has multiple optima due to problem symmetry. If the point (x_1, x_2, x_3, x_4) is optimal ((x_1, x_2) is the location of the first ambulance and (x_3, x_4) is the second), then so are (x_3, x_4, x_1, x_2), (x_2, x_1, x_4, x_3) and (x_4, x_3, x_2, x_1). In order to avoid problems caused by multiple optima, in our numerical examples, we minimize the expected value of the function $\widetilde{f}(x_1, x_2, x_3, x_4) = W(x, C, S) + 35(x_1 - \min(x_1, x_2, x_3, x_4))^2$ where $W(x, C, S)$ denotes the average waiting time; a function of the ambulance locations x, service process, S and the call process, C, and the second term is the penalty to assure the problem has unique optimum. The average waiting time when both ambulances are located at the origin – a clearly bad configuration – is estimated as 32.63 (with a standard error of 0.08) minutes. For this objective function, the optimum point is $x^* = (0.360, 0.670, 0.778, 0.545)$ which is found by estimating the average waiting time over a four-dimensional grid with precision 0.005. The optimum expected waiting time is estimated as $f(x^*) = -11.3175$ (with a standard error of 0.0008) minutes.

In the experiments, we assume initial base locations of $(0.5, 0.5)$ for both ambulances. We simulate the system for 2,000 hours of operation to estimate the long term average response time for each configuration of ambulance locations. We compare performances of the KW, SS-KW, SPSA and SS-SPSA algorithms. The KW and SS-KW algorithms are run for 4,000 iterations where SPSA and SS-SPSA are run for 10,000 iterations. In other words, all algorithms are run for 20,000 function evaluations. The performance measures are calculated using 300 independent replications and results are summarized in Table 8. Here and in the rest of the numerical examples, in order to reduce variance, we use common random numbers to estimate the $\mathbb{E}[\widetilde{f}(x^*) - \widetilde{f}(X^{(n)})]$. For a given sample path of $\{X^{(n)}\}$, we calculate the difference $\widetilde{f}(x^*) - \widetilde{f}(X^{(n)})$ using the same

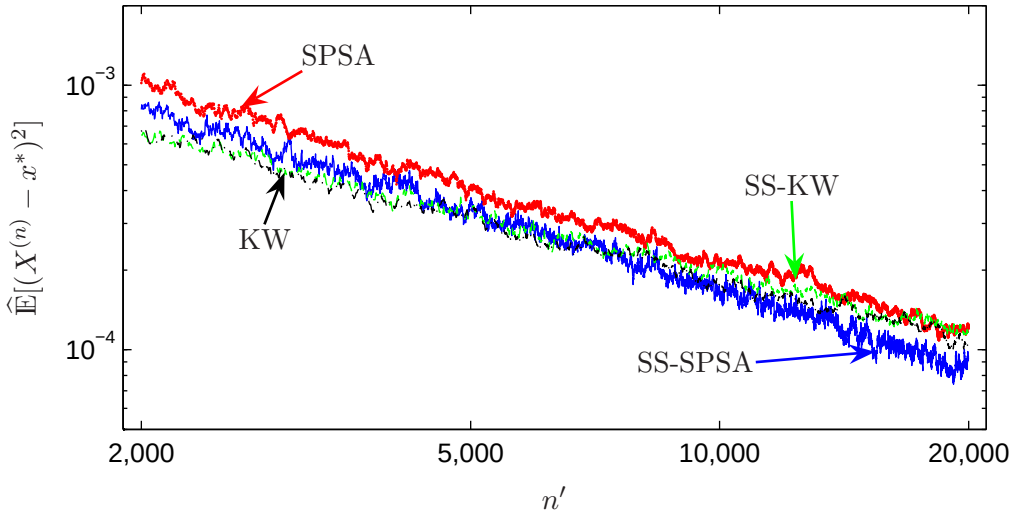

Figure 5: Ambulance base locations. The log-log plot shows the convergence plots of the KW, SS-KW, SPSA and SS-SPSA algorithms. The x-axis shows $\log(n')$ where n' is the number of function evaluations at iteration n of the algorithms, i.e., $n' = 5n$ for KW and SS-KW algorithms and $n' = 2n$ for SPSA and SS-SPSA algorithms. The convergence rate is estimated as $-0.77(0.04)$, $-0.75(0.02)$, $-0.92(0.03)$, $-0.99(0.02)$ for KW, SS-KW, SPSA and SS-SPSA, respectively.

random numbers in the calculation of $\widetilde{f}(x^*)$ and $\widetilde{f}(X^{(n)})$, take this difference and average over all 300 paths to estimate the expectation of this difference for given n.

As seen from Figure 5, the convergence rate estimates for all three algorithms are on par with each other, with SS-SPSA exhibiting marginally better results. In other words, the original choice of the sequences for the ambulance base location problem does not cause any of the performance issues described in Section 2.1. This example also illustrates that if the original choice of the sequences "matches" the characteristics of the underlying problem, then scaling and shifting them does not degrade performance. In this example, the median values for adaptations of the SS-KW algorithm along the four dimensions are $\alpha = 1$ for all dimensions, $\beta_1 = \beta_2 = 10$, $\beta_3 = 11$ and $\beta_4 = 13$ in the $\{a^{(n)}\}$ sequence, and $\gamma = 1$ for all dimensions in the $\{c^{(n)}\}$ sequence. For the SS-SPSA algorithm, the values are $\alpha = 1$ for all dimensions, $\beta_1 = \beta_4 = 91$, $\beta_2 = 90$ and $\beta_3 = 89$ in the $\{a^{(n)}\}$ sequence, and $\gamma_1 = 7.3$, $\gamma_2 = \gamma_3 = \gamma_4 = 7.1$ in the $\{c^{(n)}\}$ sequence.

Table 8: Ambulance base locations. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(n)} - x^*\|^2$, the MSE convergence rate, the final error in average waiting time in minutes, i.e., $\widehat{\mathbb{E}}\widetilde{f}(X^{(n)}) - f(x^*)$ and the convergence rate of this error values.

	$\widehat{\mathbb{E}}\|X^{(2000)} - x^*\|^2$	Rate	$\widehat{\mathbb{E}}\widetilde{f}(X^{(2000)}) - f(x^*)$	Rate
KW	1.0×10^{-04} (4.5×10^{-06})	-0.77 (0.04)	0.0016 (0.0002)	-0.71 (0.05)
SS-KW	1.2×10^{-04} (5.1×10^{-06})	-0.75 (0.02)	0.0029 (0.0002)	-0.73 (0.02)
SPSA	1.2×10^{-04} (5.3×10^{-06})	-0.92 (0.03)	0.0013 (0.0002)	-0.97 (0.03)
SS-SPSA	9.2×10^{-05} (6.1×10^{-06})	-0.99 (0.02)	0.0012 (0.0001)	-0.96 (0.04)

3.4 Airline Seat Protection Levels

This revenue management problem is described in detail by McGill and van Ryzin [25]. The problem is to determine the airline seat protection levels to maximize expected revenue. There are four fare classes on a single-leg flight. We assume that demands for different fare classes are mutually independent and low-fare classes book strictly before higher fare classes. There are no cancellations or no-shows, and the demand as well as the seat capacity are both assumed to be real-valued.

Fixed protection levels are defined as the vector (x_1, x_2, x_3) with $x_1 \leq x_2 \leq x_3$ where x_i is defined as the number of seats reserved for all classes $1, 2, 3$. There is no protection level for the lowest fare class. Reservations for fare class $i+1$ are accepted only if the remaining number of seats is strictly greater than the protection level, x_i, $i = 1, 2$.

The problem can be summarized as follows:

$$\begin{aligned}
\max_{x_1,x_2,x_3} f(x) &= \mathbb{E}\widetilde{f}(x) = \mathbb{E}\left[\sum_{i=1}^{4} p_i s_i\right] \\
\text{s.t.} \quad s_i &= [\min(D_i, c_i - x_{i-1})]^+ \text{ for } i = 1, \ldots, 4 \\
c_{i-1} &= c_i - s_i \text{ for } i = 2, 3, 4 \\
c_4 &= 164 \text{ (total capacity of the leg)} \\
x_0 &= 0
\end{aligned}$$

where p_i is the fare for class i, s_i is the number of seats sold to class i, c_i is the remaining capacity for class i, and D_i is the demand for class i. The fares for first through fourth classes are assumed to be \$1050, \$567, \$527, and \$350, respectively. The demand for each fare class is normally distributed with means 17.3, 45.1, 73.6 and 19.8, and standard deviations 5.8, 15.0, 17.4 and 6.6 for classes $i = 1, 2, 3, 4$, respectively. The total seat capacity is assumed to be 164. For this set of parameters, the optimal protection levels are $x^* = (16.7175, 43.9980, 132.8203)$ and the optimal revenue is $f(x^*) = \$85{,}055$ (the standard error of estimate is \$3).

Brumelle and McGill [10] show that when the demand distributions are continuous (which is assumed here), the optimal protection levels (x_1^*, x_2^*, x_3^*) satisfy $r_{i+1} = \mathbb{P}(A_i(x^*, D))$ for $i = 1, 2, 3$ where $r_{i+1} = p_{i+1}/p_1$ and

$$\begin{aligned}
A_1(x, D) &= \{D_1 > x_1\} \\
A_2(x, D) &= \{D_1 > x_1, \quad D_1 + D_2 > x_2\} \\
A_3(x, D) &= \{D_1 > x_1, \quad D_1 + D_2 > x_2, \quad D_1 + D_2 + D_3 > x_3\}.
\end{aligned}$$

Using this characterization of the optimal point, McGill and van Ryzin [25] propose an RM-type algorithm. They define

$$\widetilde{g}_i(x, D) = r_{i+1} - \mathbb{1}(A_i(x, D)) \tag{6}$$

and use the recursion $X_i^{(n+1)} = X_i^{(n)} - a_i^{(n)} \widetilde{g}_i(x, D)$ along each dimension $i = 1, 2, 3$ to estimate x^*. Under suitable conditions on the $\{a^{(n)}\}$ sequences, they prove that $X^{(n)} \to x^*$ as $n \to \infty$. In their numerical tests, for the set of parameters given here, they use $a^{(n)} = 200/(n + 10)$, claiming that this choice of the sequence appeared to provide good performance on a range of examples.

We test the SS-RM algorithm against RM algorithms with $a^{(n)} = 1/n$ and $a^{(n)} = 200/(n + 10)$ using $\widetilde{g}(x, D)$ as defined in (6). We use 2,000 flight departures in total (i.e., 2,000 iterations of each algorithm) and, the reported numbers are calculated using 1,000 independent runs of the algorithms. Iterates are truncated below at $(0, 15, 65)$ and above at $(35, 110, 164)$. These are the low and high starting levels given in [25]. The performance results are summarized in Table 9. We use common random numbers to estimate $\mathbb{E}[\widetilde{f}(x^*) - \widetilde{f}(X^{(2,000)})]$.

Table 9: Airline seat protection levels. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(2,000)} - x^*\|^2$, the MSE convergence rate, the estimate for the final error in expected total revenue, i.e., $\widehat{\mathbb{E}}[\widetilde{f}(x^*) - \widetilde{f}(X^{(2,000)})]$, and the convergence rate of this value.

	$\widehat{\mathbb{E}}\|X^{(2,000)} - x^*\|^2$	Conv. rate	$\widehat{\mathbb{E}}[\widetilde{f}(x^*) - \widetilde{f}(X^{(2,000)})]$	Conv. rate
RM ($a^{(n)} = 1/n$)	2268 (55)	-0.01 (0.0007)	\$3283 (120)	-0.03 (0.001)
RM ($a^{(n)} = 200/(n + 10)$)	2.42 (0.08)	-1.08 (0.04)	\$3.0 (0.1)	-0.97 (0.02)
SS-RM	8.9 (0.3)	-0.99 (0.02)	\$5.6 (0.2)	-1.12 (0.03)

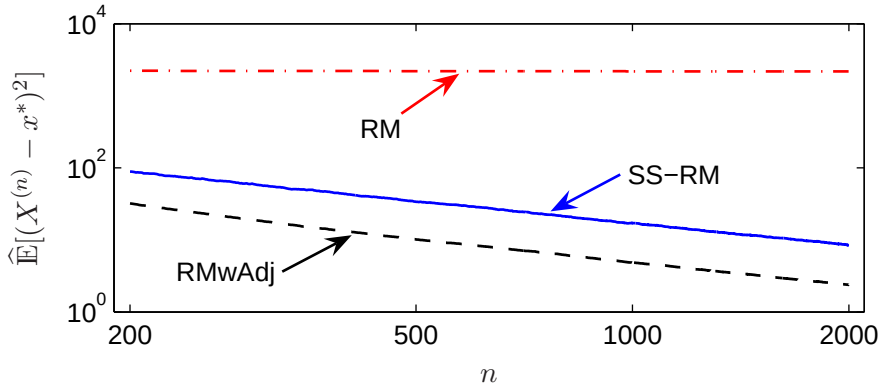

Figure 6: Airline seat protection levels. The log-log plot shows the improved convergence rate of the SS-RM algorithm over the RM-algorithm with $a^{(n)} = 1/n$ (the estimated rate of convergence for the latter is -0.01). The performance of the SS-RM algorithm in terms of MSE values is also close to that of the RM-algorithm using the pre-adjusted sequence of ($a^{(n)} = 200/(n+10)$) (dubbed RMwAdj in the figure). The SS-RM algorithm converges at an estimated rate of -0.99 and at the rate -1.08 for RMwAdj (see Table 9 for standard errors).

The RM-algorithm with $a^{(n)} = 1/n$ converges very slowly (rate is estimates as -0.01) as shown in the first line of Table 9. The "hand-picked" sequence in [25] ($a^{(n)} = 200/(n + 10)$) significantly improves on that and recovers the theoretically optimal rate of -1.0. The SS-RM algorithm, which

does not require any extra pre-processing for sequence tuning also recovers the optimal convergence rate of -1.0. Since the adaptive adjustments of the tuning sequences requires some iterations at the beginning of the algorithm (the scaling and shifting phases), the MSE results for the SS-RM algorithm are slightly worse than the RM-algorithm with the hand-picked sequences (see the second line in Table 9), but this of course ignores the cost of pre-experimentation. Convergence plots for MSE of each algorithm are given in Figure 6. The SS-RM algorithm achieves the improvement in convergence rate by scaling up the $\{a^{(n)}\}$ sequences. After the adaptations are finished in the SS-RM algorithm, the median scale-up in the $\{a^{(n)}\}$ sequences are $\alpha_1 = 757$, $\alpha_2 = 763$ and $\alpha_3 = 1144$.

3.5 Multiechelon Production-Inventory Systems

A central problem within supply chain management is the minimization of total average inventory cost in multiechelon production-inventory systems. We focus here on a variant of the problem studied by Glasserman and Tayur [11]. In this multiechelon system, every stage follows a base stock policy for echelon inventory. We assume a fixed lead time from production to inventory of one period, continuous demand, periodically reviewed inventories and we allow backlogging for any unfilled order. There are three stages: Stage one supplies external demands and stage i supplies components for stage $i-1$, $i = 2, 3$. Stage 3 draws raw materials from an unlimited supply. Each stage needs to specify an echelon-inventory base-stock policy x_i. Within each period, first pipeline inventories arrive to the next stage, then demand is realized at stage one and is either fulfilled or backlogged. Finally, the production level for the current period n, at each stage $i = 1, 2, 3$, R_n^i is set. In period n ($n = 1, \ldots, N$) stage i sets production to try to restore the echelon inventory position, $\sum_{j=1}^{i}(I_n^j + T_n^j) - D_n$ to its base-stock level, x_i where I_n^i is the installation inventory at stage i for $i = 2, 3$, and I_n^1 is the inventory at stage one minus the backlog in period n, T_n^i is the pipeline inventory, and D_n is the demand in that period. The production at stage i, $i < m$, is constrained by available component inventory I_n^{i+1}, as well as the capacity limit of that stage, which is assumed to be $p^i = 60$ units for each stage. For $i > 1$, the amount removed in period n from storage at stage i is the downstream production level. The objective function is the expected average cost which is comprised of the holding cost and the backholding costs. The problem can

be summarized as follows:

$$\begin{aligned}
\min_{x\in\mathbb{R}^3} f(x) &= \mathbb{E}\widetilde{f}(x) = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}\left[C_n(x,D)\right] \\
\text{s.t.} \quad C_n &= [I_n^1]^- b + ([I_n^1]^+ + T_n^1)h^1 + \sum_{i=1}^{4}(I_n^i + T_n^i)h^i \\
I_{n+1}^i &= I_n^i - R_n^{i-1} + R_{n-1}^i \\
R_n^i &= \max\{p^i, \left[x_i + D_n - \sum_{j=1}^{i}(I_n^j + T_n^j)\right]^+, I_n^{i+1}\} \\
T_{n+1}^i &= T_n^i + R_n^i - R_{n-1}^i,\ i = 1,2,3,\ n = 1,\ldots,N
\end{aligned}$$

where $[y]^- = -\min(y,0)$ and $[y]^+ = \max(y,0)$ for any $y \in \mathbb{R}$. The associated holding cost per unit at each stage is $(h^1, h^2, h^3) = (\$20{,}000, \$10{,}000, \$5{,}000)$. Backorders at the first stage are penalized at $b = \$40{,}000$ per unit backordered. The demand is assumed to have an Erlang distribution with mean 50 and squared coefficient of variation 0.25. The initial conditions are assumed to be $I_1^1 = x_1$ and $I_1^i = x_i - x_{i-1}$ for $i = 2, 3$ and all other variables are set to zero. In order to estimate the total average cost at each iteration, we delete the first 500 periods as a transient phase and we use the next 10,000 periods (10,500 periods in total). With these parameters, the optimal base-stock levels are $x^* = (81, 200, 320)$ units and the resulting expected minimum cost is estimated as \$377,290 (the standard error of estimate is \$319).

We run the SPSA and SS-SPSA algorithms using an initial interval of $I^{(0)} = [20, 200] \times [40, 300] \times [60, 400]$. Both algorithms are run for 2,000 function evaluations (1,000 iterations) and the MSE estimates are calculated using 100 independent runs of the algorithm. The reason for running the SPSA and SS-SPSA algorithms and not the KW and SS-KW stems from the problem structure. In particular, calculating the gradient estimate using finite differences along each dimensions separately (as in KW and SS-KW) requires changing only one of the base-stock levels while keeping others the same. Because the production at each stage is constrained by the downstream inventory level, changing the base-stock level at a single stage alone typically does not change the objective function value, so the gradient estimate becomes zero. In order to avoid this, we run the SPSA and SS-SPSA algorithms which calculate the gradient along the diagonals, i.e., by changing all three base-stock levels together.

The SPSA algorithm is observed to have a long oscillatory period in this problem. The median oscillatory period for the SPSA algorithm is 414. The SS-SPSA algorithm brings this down to 27. As seen in the performance results summarized in Table 10, the decrease in the oscillatory period length improves the MSE by an order of magnitude and the function error by a factor of 3. The

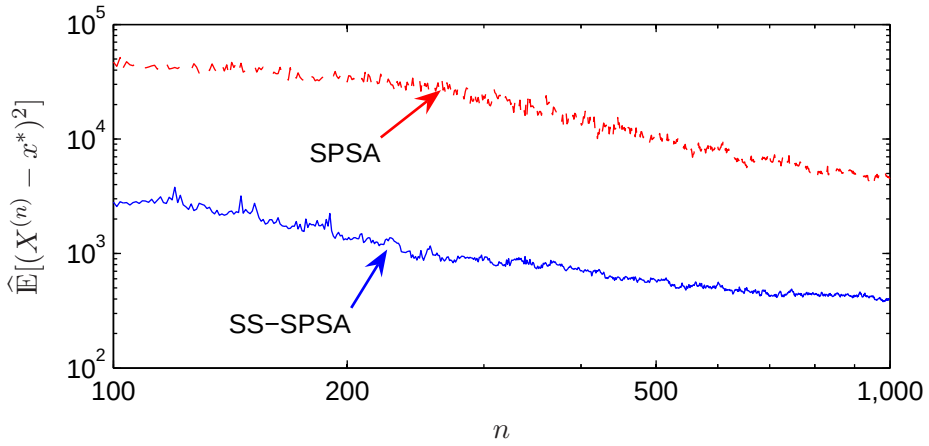

Figure 7: Multiechelon production-inventory systems. The log-log plot shows the improved convergence of the SS-SPSA algorithm over the SPSA algorithm.

decrease in the oscillatory period is achieved mainly by shifting the $\{a^{(n)}\}$ sequences: the median shifts values are 759, 436 and 317 in each dimension, respectively. Moreover, we observe a median scale up of 7.3 in the $\{c^{(n)}\}$ sequences along all three dimensions which decreases the noise level in gradient estimates. For this example, because of the initial oscillatory behavior of the iterates in the SPSA algorithm, we do not report the convergence rate estimates for this algorithm.

Table 10: Multiechelon production-inventory systems. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(1,000)} - x^*\|^2$ and the final error in expected cost, i.e., $\widehat{\mathbb{E}}\widetilde{f}(X^{(1,000)}) - f(x^*)$. The rate estimates for the SPSA algorithm are not provided since eliminating this transient period leaves only iterations 709 to 1,000 to estimate the rate, which is not long enough. The last two columns of the table report statistics about the length of the oscillatory period and shows improved finite-time performance of SS-SPSA algorithm over SPSA. $\mathrm{T}_{\mathrm{med}}$ is the median oscillatory period length calculated using 100 independent runs of the algorithms, and $\mathrm{T}_{\mathrm{max}}$ is the maximum oscillatory period length among all these runs. Numbers in parenthesis are the standard errors of estimates.

	$\widehat{\mathbb{E}}\|X^{(1,000)} - x^*\|^2$	Rate	$\widehat{\mathbb{E}}\widetilde{f}(X^{(1,000)}) - f(x^*)$	Rate	$\mathrm{T}_{\mathrm{med}}$	$\mathrm{T}_{\mathrm{max}}$
SPSA	4595 (428)	N/A	\$87,761 (11,318)	N/A	414	609
SS-SPSA	402 (99)	-0.87 (0.13)	\$30,429 (1225)	-0.63 (0.05)	27	171

3.6 Merton Jump-Diffusion Model Calibration

We consider the problem of calibrating the Merton jump-diffusion model parameters to European put options on a non-dividend paying asset. The aim is to choose model parameters to minimize the difference between the model and the given market prices. The model described in the seminal paper by Merton [18] assumes that the underlying asset price is governed by a geometric Brownian motion with drift, together with a compound Poisson process to model the jump times. Under the risk-neutral probability measure, the stochastic differential equation governing the underlying asset

price, S_t is

$$\frac{dS_t}{S_t} = (r - \lambda\mu_J)dt + \sigma dW_t + dJ_t.$$

Here, $J_t = \sum_{i=1}^{N_t}(Y_i - 1)$ where N_t is a Poisson process with constant intensity λ. When a jump occurs at time t^-, the stock price becomes $S_{t^+} = S_{t^-} \times Y$ where Y is lognormally distributed with mean $\mathbb{E}[Y] = 1 + \mu_J$ and variance σ_J^2, with (μ_J, σ_J^2) parameters of the model.

We first assume "true values" for the four model parameters, $\sigma = 10\%$, $\lambda = 1.51$, $\mu_J = -6.85\%$ and $\sigma_J = 6\%$ which are the values given in [7]. For the initial price $S_0 = \$100$ and interest rate $r = 4.5\%$, we calculate the prices of the European put options with maturities and strikes given in Table 11. The market price is estimated using the closed-form formula for the Merton model given in [18], which expresses the Merton price as an infinite weighted sum of Black-Scholes prices with modified interest rate and volatility:

$$p^{JD}(S_0, K, \sigma, r, T, \lambda, \mu_J, \sigma_J) = \sum_{k=0}^{\infty} \frac{e^{\lambda' T}(\lambda' T)^k}{k!} p^{BS}(S_0, K, \sigma_k, r_k, T)$$

where p^{JD} is the Merton jump-diffusion model price, p^{BS} is the Black-Scholes price, $\lambda' = \lambda(1+\mu_J)$, $\sigma_k^2 = \sigma^2 + k\sigma_J^2/T$ and $r_k = r - \lambda\mu_J + k\ln(1+\mu_J)/T$. We truncate the infinite sum when the last term in the sum on the right-hand-side is smaller than 10^{-10} and get the numbers reported in Table 11. These are used as the market prices for these options. Market Black-Scholes (BS) implied volatilities, $\sigma_i^{BS}(p_i)$ for each option $i = 1, \ldots, 11$ are calculated as defined in Section 13.11 of [13], and are reported in the last column of Table 11.

In our numerical experiment, for given values of the three parameters, σ, λ, μ_J and σ_J we first simulate the diffusion part of the underlying process up to the earliest maturity in one time

Table 11: European put options used in Merton calibration: This table provides the maturity, strike and market prices for the eleven put options used. The maturity used in calculations, T is calculated using $T_i = t_i/365.25$.

Maturity, t (days)	Strike, K (\$)	Price, p (\$)	Implied Vol., $\sigma^{BS}(p)$ (%)
5	95	0.0670	27.21
5	100	0.5100	11.58
33	90	0.1411	22.45
33	95	0.4207	18.02
33	100	1.4187	13.48
33	105	4.7521	11.78
124	90	0.5564	17.50
124	95	1.2874	16.17
124	100	2.7143	14.83
124	105	5.2248	13.71
124	110	8.9109	12.93

increment. We then add the jump part of the process to get the price of the underlying for the first maturity for each path. Then, we take this as the "initial price" and repeat the process to simulate the underlying price at the next maturity. Using 10,000 simulations of the final stock price, $S_{T_i}^{(j)}$ for $j = 1, \ldots, 10{,}000$ we estimate the prices for each options, $\widetilde{p}_i(\sigma, \lambda, \mu_J, \sigma_J) = \sum_{j=1}^{10{,}000} \max(K_i - S_{T_i}^{(j)}, 0)/10{,}000$ for $i = 1, \ldots, 11$. The simulated price for option i may be zero if none of the paths end up in the money, i.e., if $S_{T_i}^{(j)} \geq K_i$ for all $j = 1, \ldots$ 10,000. Also, even if the simulated option price, $\widetilde{p}_i(\sigma, \lambda, \mu_J, \sigma_J)$ is positive, it may be lower than $\delta_i := \max(0, K_i e^{-rT_i} - S_0)$ for $i = 1, \ldots, 11$, which is a lower bound on each option price. In order to avoid numerical issues in calculating BS-implied volatilities, we implement a modified implied volatility function. For a fixed small number ε (we use $\varepsilon = 0.01$), if $\widetilde{p}_i(\sigma, \lambda, \mu_J, \sigma_J) \leq \delta_i + \varepsilon$ then we estimate the implied volatility for option i using linear extrapolation. If $\widetilde{p}_i(\sigma, \lambda, \mu_J, \sigma_J) \leq \delta_i + \varepsilon$, we define the simulated implied volatility as $\sigma_i^{sim}(\widetilde{p}_i) = m\widetilde{p}_i + n$ where $m := [\sigma_i^{BS}(\delta_i + 2\varepsilon/3) - \sigma_i^{BS}(\delta_i + \varepsilon/3)]/(\varepsilon/3)$ and n is chosen so that it satisfies $\sigma_i^{BS}(\delta_i + \varepsilon) = m(\delta_i + \varepsilon) + n$. If $\widetilde{p}_i(\sigma, \lambda, \mu_J, \sigma_J) > \delta_i + \varepsilon$, we set $\sigma_i^{sim}(\widetilde{p}_i) = \sigma_i^{BS}(\widetilde{p}_i)$ in our simulations[1]. The objective is to minimize the implied volatility error defined as the average of squared differences between the market implied volatility, $\sigma_i^{BS}(p_i)$ and simulated implied volatility, $\sigma_i^{sim}(\widetilde{p}_i)$.

Close examination of the dependence of the option prices and implied volatilities to the parameters λ and μ_J in the Merton model reveals that the option prices are typically sensitive to the product of these two terms, and not to the values of λ and μ_J individually. Therefore, there is more than one set of values for these two parameters that generates option values which are very close to each other and this makes it hard to separately identify these two parameters. In other words, if we fix one of these parameters, then we can use the other to match the target option prices very closely. Because of the difficulty in distinguishing these two parameters, we fix the jump intensity to be $\lambda = 1$ jump per year. So, the objective function used in KW and SS-KW algorithm is $f(\sigma, \mu_J, \sigma_J) = \mathbb{E}(\widetilde{f}(\sigma, \mu_J, \sigma_J)) = \mathbb{E}\sum_{i=1}^{11}(\sigma_i^{BS}(p_i) - \sigma_i^{sim}(\widetilde{p}_i(\sigma, 1, \mu_J, \sigma_J)))^2/11$. For this function, the point of optimum is accurately estimated using standard non-linear optimization methods as $(\sigma^*, \mu_J^*, \sigma_J^*) = (10.28\%, -9.74\%, 4.52\%)$. The test problem was designed so that it has known solution, but mimics the properties of real problems with similar but more complicated cash flows. With these optimal parameter values, the root-mean-squared implied volatility error for the 11 options in Table 11 is 0.036%. The KW and SS-KW algorithms are run to minimize expected implied volatility error using the initial intervals $\sigma \in [1\%, 30\%]$, $\mu_J \in [-30\%, 0\%]$ and $\sigma_J \in [1\%, 15\%]$. We run the algorithms for 3,000 iterations and the estimates of MSE are calculated using 500 independent runs of the algorithm.

The calibration problem described above has an objective function that is quite steep at the

[1]Here is a numerical example showing how $\sigma_i^{sim}(\widetilde{p}_i)$ is calculated. Assume $S_0 = 100$, $K = 95$, $r = 4.5\%$ and $T = 5/365.25$. Under this setting, $\delta_i = 0$, and assuming $\varepsilon = 0.01$, we get $m = (\sigma_i^{BS}(0.02/3) - \sigma_i^{BS}(0.01/3))/(0.01/3) = 4.45$ and $n = 0.15$. For any simulated price less than ε, i.e., if $\widetilde{p}_i < 0.01$, we set $\sigma_i^{sim}(\widetilde{p}_i) = 4.45\widetilde{p}_i + 0.15$.

boundaries and very flat near the point of optimum. Due to this, the KW-algorithm exhibits a degraded rate of convergence as seen in the first row of Table 12. In the SS-KW algorithm, the median scale-up and shift values in $a_k^{(n)} = \alpha_k/(n+\beta_k)$ sequences are $\alpha_1 = 28$, $\alpha_2 = 33$, $\alpha_3 = 16$; and $\beta_1 = \beta_2 = 10$, $\beta_3 = 2$; whereas the median scale-up in $\{c_k^{(n)}\}$ for $k = 1, 2, 3$ equal to 1. Even though the $\{a^{(n)}\}$ sequences are scaled up, the achieved convergence rate is still slower than the theoretically best values. The scale up factors are calculated based on the magnitude of the gradient at the boundaries, but when the iterates are close to the point of optimum, these scale up factors become inadequate; hence the slow convergence rate. On the other hand, when we look at the performance measures in terms of function error, as seen in the last two columns of the Table 12, we observe that even relatively minor adaptations improve the performance. Both the function error after 3000 iterations and the convergence rate of this error measure are improved by a factor of two.

Table 12: Merton jump-diffusion model calibration problem. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(3000)} - x^*\|^2$, the MSE convergence rate, the final error in function value, i.e., $\widehat{\mathbb{E}}\widetilde{f}(X^{(3000)}) - f(x^*)$ and the convergence rate of these error values. The average implied volatility error after 3000 iterations, $\sqrt{\widehat{\mathbb{E}}\widetilde{f}(X^{(3000)})}$ is 0.6% for the KW and 0.4% for the SS-KW algorithms. Numbers in parenthesis are standard errors of estimates.

	$\widehat{\mathbb{E}}\|X^{(3000)} - x^*\|^2$	Rate	$\widehat{\mathbb{E}}\widetilde{f}(X^{(3000)}) - f(x^*)$	Rate
KW	3.9×10^{-03} (1.5×10^{-04})	-0.05 (9.0×10^{-04})	3.1×10^{-05} (1.2×10^{-06})	-0.13 (0.004)
SS-KW	6.9×10^{-04} (2.4×10^{-05})	-0.08 (0.02)	1.4×10^{-05} (5.3×10^{-07})	-0.27 (0.03)

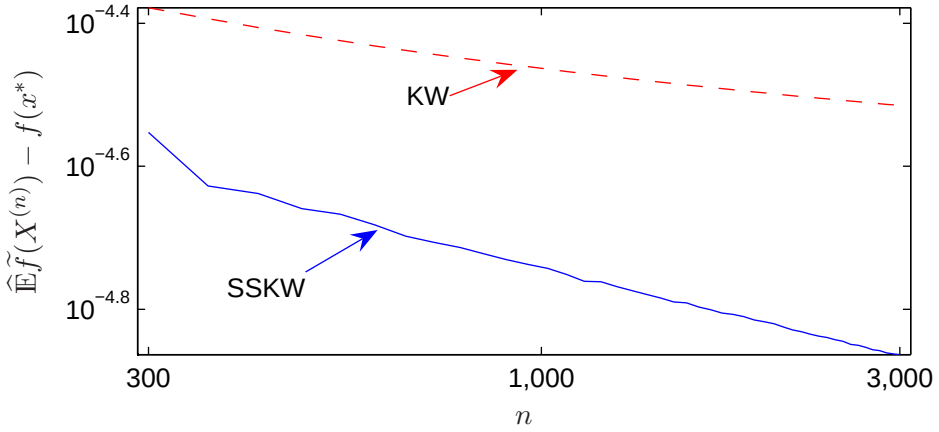

Figure 8: Merton jump-diffusion model calibration: The log-log plot shows the improved convergence of $\widehat{\mathbb{E}}\widetilde{f}(X^{(n)}) - f(x^*)$ for the SS-KW algorithm over the KW-algorithm.

3.7 Call Center Staffing

We consider a call center staffing problem described in detail by Harrison and Zeevi [12]. There are two customer classes and two server pools. Callers of each class arrive according to a non-

homogeneous Poisson process with stochastic intensities $\Lambda_1(t)$ and $\Lambda_2(t)$. Servers in pool 1 can only serve calls of class 1 while servers in pool 2 can serve both classes of customers. So there are three processing activities: server 1 serving class 1, server 2 serving class 1 and server 2 serving class 2. There are x_k servers in pool k for $k = 1, 2$. Service times are assumed to be exponentially distributed with rate $\mu_j = 0.25$ customer per minute for each processing activity $j = 1, 2, 3$. We assume services can be interrupted at any time and later resumed from the point of preemption without incurring any penalty costs. Customers of both classes who are waiting in the queue abandon the queue with exponentially distributed inter-abandonment times with mean one minute and the abandonment process is independent of the arrival and service time processes. Each abandonment by a class one customer incurs $p_1 = \$5$ penalty cost, and by a class two customer incurs $p_2 = \$10$. We set the length of the planning horizon to be $T = 480$ minutes. The cost of employing a server for a work day (480 minutes) is $c_1 = \$160$ and $c_2 = \$240$ for servers one and two, respectively. The problem is to choose the number of servers in each pool, x_1 and x_2 to minimize the total staffing and penalty costs. We define $h(x) = \mathbb{E}\widetilde{h}(x) = c^T x + \mathbb{E}[p^T q(x, S, A)]$ where $q(x, S, A) \in \mathbb{Z}^2$ represents the vector of number of abandonments depending on the random service time process, S and the arrival process, A for a given vector of pool sizes, $x \in \mathbb{Z}^2$. Here, the expectation is taken with respect to both processes.

The problem in its original form is an integer-variable problem but the stochastic approximation algorithms require problems to have a continuous feasible set. In order to be able to use stochastic approximation algorithms in this problem, we use two-dimensional linear interpolation to define function values at non-integer points (we use the `interp2` function in MATLAB in our implementation). In order to evaluate the objective function at a point (x_1, x_2), we run four simulations at points $(\lceil x_1 \rceil, \lceil x_2 \rceil)$, $(\lceil x_1 \rceil, \lfloor x_2 \rfloor)$, $(\lfloor x_1 \rfloor, \lceil x_2 \rceil)$ $(\lfloor x_1 \rfloor, \lfloor x_2 \rfloor)$ using common random numbers and we use these function evaluations to approximate the function value at (x_1, x_2). In other words, the final objective is to minimize

$$f(x) = \mathbb{E}\widetilde{f}(x) = \mathbb{E}\ \mathbb{I}\{h[(\lceil x_1 \rceil, \lceil x_2 \rceil)], h[(\lceil x_1 \rceil, \lfloor x_2 \rfloor)], h[(\lfloor x_1 \rfloor, \lceil x_2 \rceil)], h[(\lfloor x_1 \rfloor, \lfloor x_2 \rfloor)]\}$$

over $x_1, x_2 \geq 0$, where $\mathbb{I}\{\cdot\}$ is the two-dimensional linear interpolation operator.

In order to model the call arrival intensity processes, we follow [12] and split the planning horizon into 16 subintervals of length $\Delta = T/16$ minutes each. We assume $Z = (Z_n : n \in \mathbb{Z})$ is an i.i.d. sequence of $\mathbb{R}^2$-valued normally distributed random variables with zero mean and covariance matrix

$$\Sigma = 0.25 \begin{pmatrix} 1 & \rho \\ \rho & 1 \end{pmatrix},$$

where $\rho = 0.5$. Let $Y_n = (Y_n : n \in \mathbb{Z})$ be given by the recursion $Y_{n+1} = 1.5e + 0.5Y_n + Z_{n+1}$ where $e = [1, 1]'$. Then $\{Y_n\}$ is a stationary autoregressive process in $\mathbb{R}^2$ with a bivariate normal marginal

distribution with mean $[3,3]'$ and covariance matrix $\Sigma_Y = 4/3\Sigma$. Considering the first coordinate of Y to be the instantaneous arrival rate of class 1 and the second coordinate of class 2, the call arrivals exhibit both interclass correlation and temporal correlation. Using this autoregressive process, the call arrival intensity process is $\Lambda_1(t) = 15Y_{1,j}/3$ and $\Lambda_2(t) = 10Y_{2,j}/3$ for all t in the j^{th} subinterval. The expected number of arrivals over the entire day is 15 calls per minute for class 1 and 10 customers per minute for class 2. With this parameter setting, the optimal server sizes are $x^* = (59, 54)$ resulting in a total cost of \$24.62K which are determined via an exhaustive grid search over integers.

Although this problem can easily be extended into higher dimensions, the benefit of having this example in two dimensions is the ability, with the current choice of problem parameters, to decouple the staffing problem from the dynamic scheduling decisions which involve allocating the servers to incoming and waiting calls. In particular, with this set of parameters, it is optimal to give priority to class 2 customers. So when a class 2 customer arrives, if all the servers in the second pool are busy and if one of them is serving a class 1 customer, then that server interrupts the service and starts serving a class 2 customer. If there is any idle server in the first pool, then we assume the service of the interrupted customer is continued by this server in first pool.

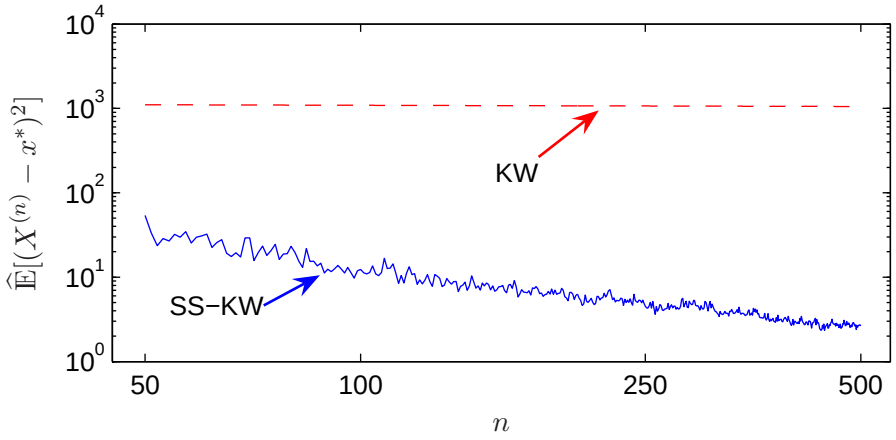

Figure 9: Call center staffing: The log-log shows the improved convergence rate of the SS-KW algorithm over the KW-algorithm.

In order to solve this problem we run KW and SS-KW algorithms for 500 iterations using initial interval of $I^{(0)} = [15, 100]^2$. We stop the runs at iteration 500 because beyond this point, the value of the $\{c^{(n)}\}$ sequences are less than one and since the original problem is only defined on integers, once the $\{c^{(n)}\}$ values are less than one, continuing the algorithm does not further improve the convergence. The performance results are estimated using 250 independent replications of the algorithms and are given in Table 13. As seen in the first row of the table, the KW-algorithm suffers from a degraded convergence rate. On the other hand, the SS-KW algorithm improves the convergence rate significantly by scaling up the $a_k^{(n)} = \alpha_k/(n + \beta_k)$, $k = 1, 2$ sequences by a median

scale up factor of $\alpha_1 = 1047$ and $\alpha_2 = 1461$. The improved performance also is observed in the function error values and convergence rates of function errors (see the last two columns of Table 13).

Table 13: Call center staffing. This table provides estimates for the final MSE value, i.e., $\widehat{\mathbb{E}}\|X^{(500)} - x^*\|^2$, the MSE convergence rate, the final error in function value, i.e., $\widehat{\mathbb{E}}\widetilde{f}(X^{(500)}) - f(x^*)$ and the convergence rate of these error values.

	$\widehat{\mathbb{E}}\|X^{(500)} - x^*\|^2$	Rate	$\widehat{\mathbb{E}}\widetilde{f}(X^{(500)}) - f(x^*)$	Rate
KW	1053 (43)	-0.02 (0.002)	6.3 (0.4)	-0.04 (0.003)
SS-KW	2.6 (0.2)	-1.0 (0.1)	0.02 (0.005)	-1.28 (0.28)

4 Conclusion

We have extended the scaling and shifting ideas first introduced in [9] to multiple dimensions. We tested the algorithms on a range of applications and found that the scaled and shifted stochastic approximation algorithms SS-RM, SS-KW and SS-SPSA have superior finite-time performance when compared to regular versions RM, KW and SPSA in most cases. If the original choice of the tuning sequences "matches" the characteristics of the underlying function (as in the ambulance base locations problem of Section 3.3), then the adaptations do not change the sequences significantly and do not deteriorate the performance. In most of the previous applications of stochastic approximation algorithms where good finite-time performance is achieved, the user must typically adjust the tuning sequences manually, by hand-picking the constants in the sequences. Scaled and shifted stochastic approximation algorithms effectively eliminate the need for this pre-processing.

A Algorithm Description

A.1 Multidimensional Scaled-and-Shifted KW Algorithm

Here we present the multidimensional SS-KW algorithm for a d-dimensional problem. The MATLAB implementation can be downloaded from `<www.columbia.edu/∼mnb2/broadie/research.html>`.

Step 1. Setting algorithm parameters and initialization. For $k = 1, \ldots, d$

- h_0: the number of forced hits to boundary points l_k and $u - c_k^{(n)}$ for all $k = 1, \ldots, d$ by scaling up the $\{a_k^{(n)}\}$ sequence (sample default: 4).
- γ_0: the scaling up factor for the $\{c_k^{(n)}\}$ sequence (sample default: 2).
- k_a: an upper bound on the number of shifts in the $\{a_k^{(n)}\}$ sequence (sample default: 50).

- υ_k^a: an initial upper bound on the per iteration shift in the $\{a_k^{(n)}\}$ sequence (sample default: 10).
- φ_a: an upper bound on the per iteration scale up in the $\{a_k^{(n)}\}$ sequence (sample default: 10).
- k_c: an upper bound on the number of scale-ups in the $\{c_k^{(n)}\}$ sequence (sample default: 50).
- $c^{(0)}$: the parameter defining the maximum possible value of the $\{c_k^{(n)}\}$ sequence after scale-ups: we require $c_k^{(n)} \leq c_k^{max} = c^{(0)}(u_k - l_k)$ for all $k = 1, \ldots, d$, $n \geq 1$ (sample default: 0.2).
- g_{max}: the maximum number of gradient estimations allowed to achieve oscillation along each dimension (sample default: 20).
- m^{max}: an upper bound on the iteration number of the last adaptation in the sequences, i.e., after iteration m^{max} no scaling or shifting is done; we require $m^{max} \geq h_0$ (sample default: total number of iterations).
- **Initialization:**
  - $\{a_k^{(n)}\} = \alpha_k/(n + \beta_k)$ for all $k = 1, \ldots, d$: the step-size sequence along dimension k, set $\alpha_k = 1$, $\beta_k = 0$.
  - $\{c_k^{(n)}\} = \gamma_k/n^{1/4}$ for all $k = 1, \ldots, d$: the sequence used in finite-difference estimation of the gradient along dimension k, set $\gamma_k = (u_k - l_k)/20$.
  - $X^{(1)}$: the initial starting point; can be random or deterministic but must satisfy $X_k^{(1)} \in [l_k, u_k - c_k^{(1)}]$ for all $k = 1, \ldots, d$.
  - $sh_k = 0$ for all $k = 1, \ldots, d$: the variable for the number of shifts in the $\{a_k^{(n)}\}$ sequence.
  - $sc_k = 0$ for all $k = 1, \ldots, d$: the variable for the number of scale-ups in the $\{c_k^{(n)}\}$ sequence.

Step 2: The scaling phase. Set $n = 1$, for $m \leq h_0$,

(a) Calculate $X^{(n+1)}$ using the recursion given in (4).

(b) Scale up the $\{a_k^{(n)}\}$ sequences for each $k = 1, \ldots, d$, if necessary, ensuring an admissible boundary, (the opposite end of the truncation interval) is hit in every dimension. Set $g = 1$ (counter for the number of gradient estimations) and $S_{osc} = \{1, \ldots, d\}$ (the set of dimensions along which oscillation is not complete). While $g \leq g_{max}$ and $|S_{osc}| > 0$. for each $k \in S_{osc}$

(i) If $X_k^{(n+1)} < u_k - c_k^{(n)}$ and $X_k^{(n+1)} > X_k^{(n)}$, find the scale-up factor for the $\{a_k^{(n)}\}$ sequence that makes $X_k^{(n+1)} = u_k - c_k^{(n+1)}$; i.e., set $\alpha'_k = \min(\varphi_a, (u_k - c_k^{(n+1)} - X_k^{(n)})/(X_k^{(n+1)} - X_k^{(n)}))$ and $\alpha_k \leftarrow \alpha'_k \alpha_k$ for the rest of the iterations. Set $X_k^{(n+1)} = u_k - c_k^{(n+1)}$, $S_{osc} = S_{osc} \setminus \{k\}$.

(ii) If $X_k^{(n+1)} > l_k$ and $X_k^{(n+1)} < X_k^{(n)}$, find the scale-up factor for the $\{a_k^{(n)}\}$ sequence that makes $X_k^{(n+1)} = l_k$; i.e., set $\alpha'_k = \min(\varphi_a, (l_k - X_k^{(n)})/(X_k^{(n+1)} - X_k^{(n)}))$ and $\alpha_k \leftarrow \alpha'_k \alpha_k$ for the rest of the iterations. Set $X_k^{(n+1)} = l_k$, $S_{osc} = S_{osc} \setminus \{k\}$.

(iii) If $n \leq m^{max}$ and $sc_k \leq \kappa_c$, scale up the $\{c_k^{(n)}\}$ sequence whenever the gradient estimate is too noisy: If $X_k^{(n+1)} > X_k^{(n)} = u_k - c_k^{(n)}$ or $X_k^{(n+1)} < X_k^{(n)} = l_k$ then calculate $\gamma'_k = \min\{\gamma_0, c^{max}/c_k^{(n)}\}$ and set $\gamma_k \leftarrow \gamma'_k \gamma_k$ for the rest of the iterations.

(iv) Update $X_k^{(n+1)} = \min\{u_k - c_k^{(n+1)}, \max\{X_k^{(n+1)}, l_k\}\}$ for all $k = 1, \ldots, d$. Increment n and g.

Figure 10 shows a typical oscillation in the scaling phase of the algorithm. Figure 11 illustrates an instance, where due to a noisy gradient estimate, one oscillation takes more than two iterations.

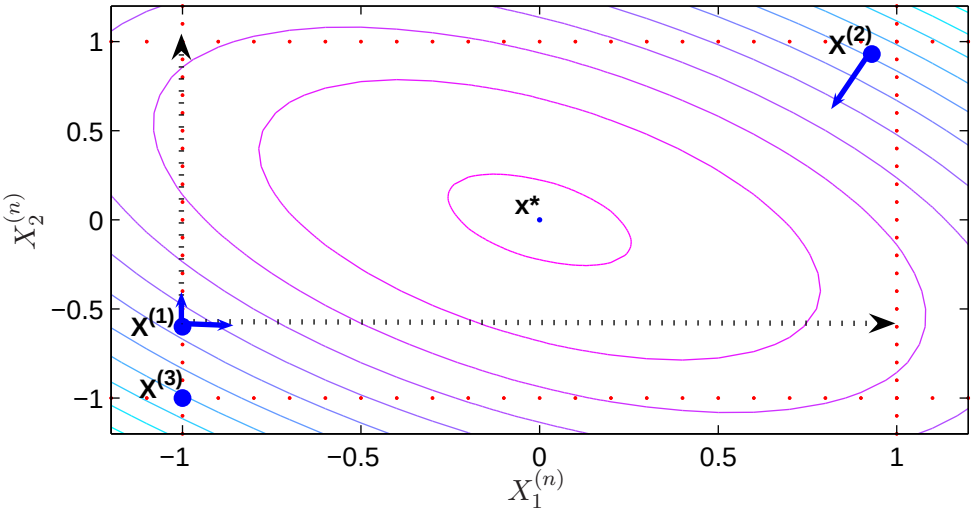

Figure 10: Typical oscillations. The figure shows the contours of the underlying function and how the first two oscillations are achieved by scaling up the $\{a_k^{(n)}\}$ sequences. The iterate $X^{(1)}$ is at the lower boundary of the truncation region along the x-axis and in the interior along the y-axis. Therefore the only admissible boundary along the x-axis is the upper boundary where both boundaries are defined to be admissible along y-axis. The gradient estimate at this point in each direction is positive (as shown by the solid arrows) and points towards an admissible boundary. Scaling up the $\{a_1^{(n)}\}$ and $\{a_2^{(n)}\}$ sequences (shown by the dotted arrows) an admissible boundary is hit along both dimensions. (The next iterate is at the point $(1 - c_1^{(2)}, 1 - c_2^{(2)})$ which is the upper boundary of the truncation region.) The first oscillation is achieved by moving from $X^{(1)}$ to $X^{(2)}$. At the next iteration, the gradient is estimated at $X^{(2)}$, and it points towards the admissible boundaries along both dimensions, so the $\{a^{(n)}\}$ sequences are scaled up and the next iterate is at the point $X^{(3)}$. This completes the second oscillation.

Step 3: The shifting phase. Until the termination of the algorithm,

(a) Calculate $X^{(n+1)}$ using the recursion given in (4).

(b) If $n \leq m^{max}$ and $sh_k \leq \kappa_a$, shift the $\{a_k^{(n)}\}$ sequence for all $k = 1, \ldots, d$, if necessary, to

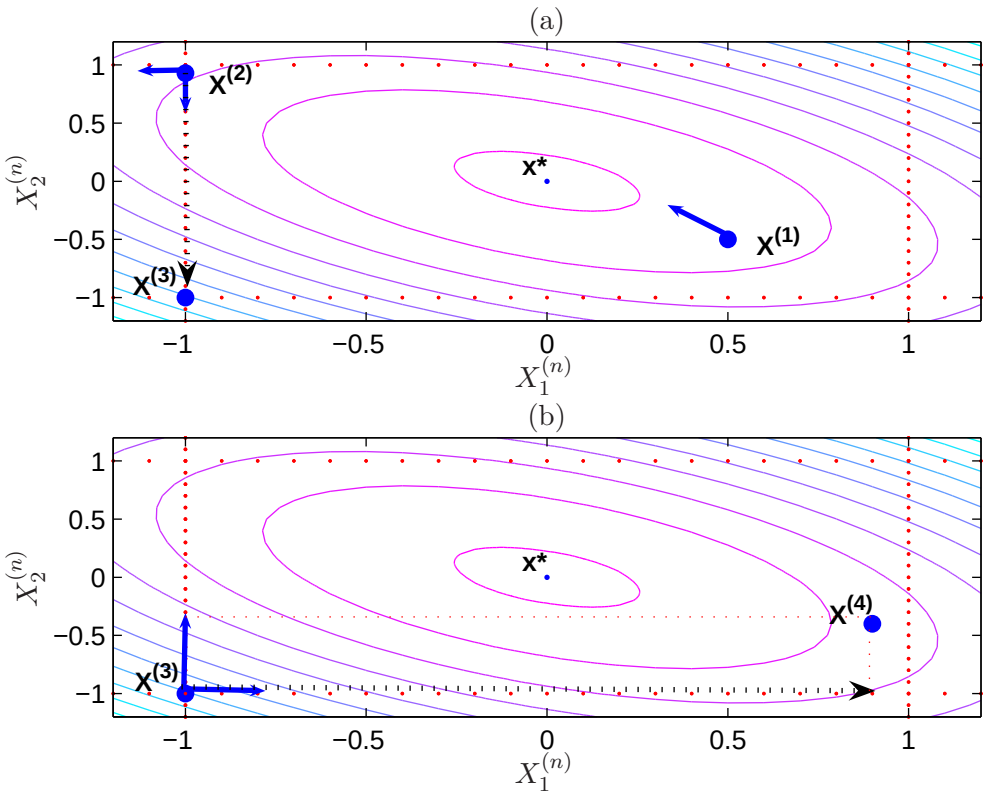

Figure 11: Two oscillations in three iterations. As explained in Figure 10, since the iterate $X^{(1)}$ is in the interior of the truncation region along both dimensions, both boundaries are admissible in each direction. Therefore the first oscillation is completed by scaling up both of the $\{a_1^{(n)}\}$ and the $\{a_2^{(n)}\}$ sequences and moving from $X^{(1)}$ to $X^{(2)}$. Now let us consider the gradient estimate at $X^{(2)}$ along the two axes separately (the two solid arrows shown here). Along the x-axis, even though the true gradient is positive, due to high noise level we get a negative gradient estimate which points outwards along the x-axis. But since the iterate $X^{(2)}$ is already at the lower boundary of the truncation region along this dimension, the only admissible boundary is the upper boundary. Therefore the iterate cannot hit the admissible boundary along this dimension by scaling up the $\{a_1^{(n)}\}$ sequence. On the other hand, the gradient along the y axis points toward the lower boundary, which is the admissible boundary and moving from point $X^{(2)}$ to $X^{(3)}$, only $\{a_2^{(n)}\}$ sequence which corresponds to the sequence used along the y-axis is scaled up. (The scale up is shown in dotted arrow in panel (a).) Then the gradient at $X^{(3)}$ is estimated and it points towards the opposite boundary along both dimensions as seen in panel (b). Moving from $X^{(3)}$ to $X^{(4)}$, the $\{a_2^{(n)}\}$ sequence is not scaled up since an admissible boundary is already hit along this dimension within the second oscillation. Only the $\{a_1^{(n)}\}$ sequence is scaled up to hit the admissible boundary and this completes the second oscillation.

prevent iterates exiting the truncation interval, but use the upper bound parameter υ_k^a to prevent a too large shift.

(i) If $X_k^{(n+1)} > u_k - c_k^{(n+1)}$, $X_k^{(n)} = l_k$ then find the minimum shift in $\{a_k^{(n)}\}$ sequence that makes $X_k^{(n+1)} \leq u_k - c_k^{(n+1)}$; i.e.,

- Solve $(u_k - c_k^{(n+1)} - X_k^{(n)}) / \left(\frac{\widetilde{f}(X^{(n)} + c_k^{(n)} e_k) - \widetilde{f}(X^{(n)})}{c_k^{(n)}} \right) = a_k^{(n+\beta_k')}$ for β_k'.
- Set $\beta_k \leftarrow \beta_k + \min(\upsilon_k^a, \lceil \beta_k' \rceil)$ for the rest of the iterations. If $\min(\upsilon_k^a, \lceil \beta_k' \rceil)) = \upsilon_k^a$,

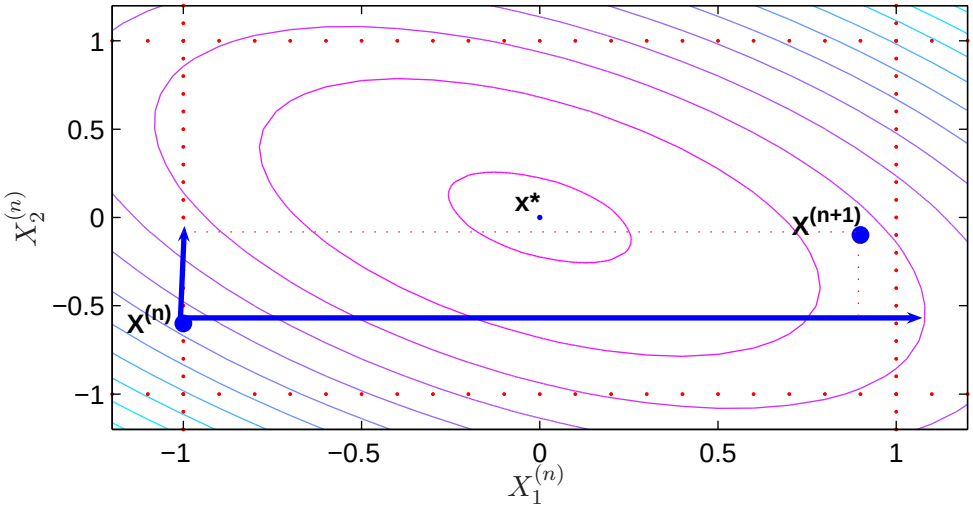

Figure 12: Shifting $\{a^{(n)}\}$ sequence. The figure shows the contours of the underlying function and an instance where the $\{a_1^{(n)}\}$ sequence is shifted to decrease the step-size sequence. The iterate $X^{(n)}$ is at the lower end of the truncation region along the x-axis. Using the original value of the $\{a_1^{(n)}\}$ sequence with the estimated gradient (shown in solid arrow) pushes the iterate beyond the upper end of the truncation region, which is at $1 - c_1^{(n)}$ along the same dimension. Because the iterate moves from one boundary to the opposite along the x-axis, the $\{a_1^{(n)}\}$ sequence is shifted so that the next iterate, $X^{(n+1)}$ is within the truncation region after the shift. Only the $\{a_1^{(n)}\}$ sequence is shifted since the iterate does not move from one boundary to the opposite along the y-axis.

then set $\upsilon_k^a = 2\upsilon_k^a$.

(ii) If $X_k^{(n+1)} < l_k$, $X_k^{(n)} = u_k - c_k^{(n)}$ then find the minimum shift in $\{a_k^{(n)}\}$ sequence that makes $X_k^{(n+1)} \leq l_k$; i.e.,

- Solve $(l_k - X_k^{(n)}) / \left(\frac{\widetilde{f}(X^{(n)} + c_k^{(n)} e_k) - \widetilde{f}(X^{(n)})}{c_k^{(n)}} \right) = a_k^{(n+\beta_k')}$ for β_k'.
- Set $\beta_k \leftarrow \beta_k + \min(\upsilon_k^a, \lceil \beta_k' \rceil)$ for the rest of the iterations. If $\min(\upsilon_k^a, \lceil \beta_k' \rceil)) = \upsilon_k^a$, then set $\upsilon_k^a = 2\upsilon_k^a$.

(c) Repeat Step 2(b)(iii). Set $X_k^{(n+1)} = \min\{u_k - c_k^{(n+1)}, \max\{X_k^{(n+1)}, l_k\}\}$ for all $k = 1, \ldots, d$. Increment n.

Figure 12 illustrates how the $\{a_k^{(n)}\}$ sequence is shifted.

A.2 Illustrative Example

We provide here an illustrative example of the adaptations in the tuning sequences. Suppose $\widetilde{f}(x) = -(Kx)'Ax + \sigma Z$ where

$$A = \begin{pmatrix} 1 & 0.5 \\ 0.5 & 1 \end{pmatrix} \quad \text{and} \quad K = \begin{pmatrix} 0.001 & 0 \\ 0 & 0.001 \end{pmatrix},$$

and Z is a standard normal random variable, $\sigma = 0.01$. The initial interval is assumed to be $I(0) = [-1, 1]^2$. We set the tuning sequences as $a_k^{(n)} = \alpha_k/(n + \beta_k)$ and $c_k^{(n)} = \gamma_k/n^{1/4}$ for $k = 1, 2$ and set $\alpha_1 = \alpha_2 = 1$, $\beta_1 = \beta_2 = 0$ and $\gamma_1 = \gamma_2 = 0.1$ initially. (We initialized as $\gamma_k = 0.1$ instead of $\gamma_k = 1$ since the width of the initial interval is two.)

The initial starting point for the illustrated path is $X^{(1)} = (0.9, -0.54)$. Here's an explanation of what happens in each iteration (see Table 14 for exact numbers):

- **Iteration 1:** The gradient points towards opposite boundary along both dimensions; we scale up both the $\{a_1^{(n)}\}$ and the $\{a_2^{(n)}\}$ by 10 (which is the upper bound on the scale up factor per iteration) and move to the northeast corner of the truncation interval. This completes the first oscillation. Note that the truncation interval at this iteration is $[-1, 1-c_1^{(2)}]$x$[-1, 1-c_2^{(2)}]$ which is why the next iterate along second dimension is at $X_2^{(2)} = 0.92$, not 1.

- **Iteration 2:** The gradient estimate pushes the iterates out of the truncation interval along both dimensions, so we cannot hit the opposite boundary along any dimensions. The $\{c_k^{(n)}\}$ sequences are scaled up along both dimensions.

- **Iteration 3:** The gradient estimate along the first dimension points outwards, so we scale up the $\{c_1^{(n)}\}$ sequence. We hit the opposite boundary only along the second dimension by scaling up the $\{a_2^{(n)}\}$ sequence by 9.63.

- **Iteration 4:** Gradient estimate pushes the iterate outside of the truncation interval along the first dimension again, so we scale up the $\{c_1^{(n)}\}$ sequence to its maximum value. After this iteration the $\{c_1^{(n)}\}$ sequence is not scaled up anymore. The iterate still cannot complete the second oscillation. The next iterate is still at the same boundary as before along the first dimension. It's now in the interior of the truncation interval along the second dimension.

- **Iteration 5:** We get a gradient estimate pointing towards the opposite boundary along the first dimension, so we scale up the $\{a_1^{(n)}\}$ sequence. This completes the second oscillation. Note that $X_1^{(6)} = 1 - c_1^{(6)}$.

- **Iteration 6:** The $\{a_2^{(n)}\}$ sequence is scaled up and the opposite boundary is hit along the second dimension.

- **Iteration 7:** The $\{a_1^{(n)}\}$ sequence is scaled up, the third oscillation is completed.

- **Iteration 8:** The opposite boundary is hit along the first dimension, without any scale up in the $\{a_1^{(n)}\}$ sequence. We scale up the $\{c_2^{(n)}\}$ sequence since the iterate was already at the upper end of the truncation interval along the second dimension and the gradient estimate was positive. Although the iterate was at the lower boundary before and was pushed beyond the upper boundary along the first dimension, we do not shift the $\{a_1^{(n)}\}$ sequence since we are still in the scaling phase.

- **Iteration 9,10 and 11:** The gradient estimate pushes the iterate outside the interval. The $\{c_2^{(n)}\}$ sequence is scaled up at iteration 9. It hits its upper bound and is not scaled up anymore.

- **Iteration 12:** $\{a_2^{(n)}\}$ is scaled up and the last oscillation is completed. By the end of this iteration, the scaling phase of the algorithm ends.

- **Iteration 13:** The iterate was at the upper boundary along the first dimension and overshoot the lower boundary along the same dimension. So, we shift the $\{a_1^{(n)}\}$ sequence by the maximum shift of 10. The new upper bound on the shift along the first dimension is updated as $\upsilon_1^a = 20$. By this shift the algorithm is correcting for a large scale up in the $\{a_1^{(n)}\}$ sequence due to noisy observation in the scaling phase.

- **Iteration 14:** The iterate oscillates from the lower boundary to the upper boundary along the second dimension, so the $\{a_2^{(n)}\}$ sequence is shifted by 10. We update $\upsilon_2^a = 20$.

Table 14: A sample path of iterates illustrating the adaptations. The first column is the iteration number. The second and third columns represent where the next iterate would be without the adaptations, i.e. $\widetilde{X_k^{(n+1)}} = X_k^{(n)} + a_k^{(n)}\widehat{\nabla}\widetilde{f}_k(X^{(n)})$. for $k = 1, 2$, respectively. The next four columns contain the new sequences to be used for future iterations after adaptations. The last two columns are the next iterates after the adaptations in the sequences are in place. The four oscillations are completed at the end of iterations 1, 5, 7 and 12 which means the scaling phase of the algorithm takes 12 iterations. Then shifting phase starts; we shift the $\{a_1^{(n)}\}$ sequence at iteration 13, and the $\{a_2^{(n)}\}$ sequence at iteration 14. During both phases, we also scale up the $\{c_k^{(n)}\}$ sequences as prescribed in Section 2.4. There is no more scale up in the $\{c_1^{(n)}\}$ sequence after iteration 4 and no more in the $\{c_2^{(n)}\}$ sequence after iteration 9 since at these iterations, the sequences hit the upper bound.

n	$\widehat{\nabla}\widetilde{f}_1(X^{(n)})$	$\widehat{\nabla}\widetilde{f}_2(X^{(n)})$	$\widetilde{X_1^{(n+1)}}$	$\widetilde{X_2^{(n+1)}}$	$a_1^{(n)}$	$a_2^{(n)}$	$c_1^{(n)}$	$c_2^{(n)}$	$X_1^{(n+1)}$	$X_2^{(n+1)}$
1	-0.12	0.06	0.78	-0.48	$10/n$	$10/n$	$0.1/n^{1/4}$	$0.1/n^{1/4}$	-1.00	0.92
2	-0.17	0.11	-1.85	1.45	$10/n$	$10/n$	$0.2/n^{1/4}$	$0.2/n^{1/4}$	-1.00	0.85
3	-0.08	-0.06	-1.27	0.6	$10/n$	$96.3/n$	$0.4/n^{1/4}$	$0.2/n^{1/4}$	-1.00	-1.00
4	-0.01	0.04	-1.03	0.01	$10/n$	$96.3/n$	$0.6/n^{1/4}$	$0.2/n^{1/4}$	-1.00	0.01
5	0.07	0.03	-0.86	0.67	$100/n$	$96.3/n$	$0.6/n^{1/4}$	$0.2/n^{1/4}$	0.62	0.67
6	0.02	-0.01	1.00	0.57	$100/n$	$963/n$	$0.6/n^{1/4}$	$0.2/n^{1/4}$	0.63	-1.00
7	-0.02	0.03	0.34	3.54	$553/n$	$963/n$	$0.6/n^{1/4}$	$0.2/n^{1/4}$	-1.00	0.88
8	0.06	0.25	3.04	30.74	$553/n$	$963/n$	$0.6/n^{1/4}$	$0.4/n^{1/4}$	0.65	0.77
9	0.04	0.08	3.26	9.53	$553/n$	$963/n$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	0.66	0.60
10	0.002	0.05	0.79	5.59	$553/n$	$963/n$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	0.67	0.61
11	0.03	0.03	2.22	3.13	$553/n$	$963/n$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	0.68	0.62
12	0.02	-0.003	1.40	0.34	$553/n$	$5661/n$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	0.69	-1.00
13	-0.08	-0.03	-2.58	-14.66	$553/(n+10)$	$5661/n$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	-1.00	-1.00
14	-0.04	0.01	-2.02	4.62	$553/(n+10)$	$5661/(n+10)$	$0.6/n^{1/4}$	$0.7/n^{1/4}$	-1.00	0.64

References

[1] S. N. Abdelhamid. Transformation of observations in stochastic approximation. *The Annals of Statistics*, 1:1158–1174, 1973.

[2] S. Andradóttir. A stochastic approximation algorithm with varying bounds. *Operations Research*, 43(6):1037–1048, 1995.

[3] S. Andradóttir. A scaled stochastic approximation algorithm. *Management Science*, 42(4):475–498, 1996.

[4] S. Asmussen and P.W. Glynn. *Stochastic Simulation*. Springer-Verlag, New York, 2007.

[5] A. Benveniste, M. Métivier, and P. Priouret. *Adaptive Algorithms and Stochastic Approximations*. Springer-Verlag, New York Berlin Heidelberg, 1990.

[6] J.R. Blum. Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics*, 25(4):737–744, 1954.

[7] M. Broadie, M. Chernov, and M. Johannes. Understanding index option returns. *Review of Financial Studies*, 22(11):4493–4529, 2009.

[8] M. Broadie, D.M. Cicek, and A. Zeevi. An adaptive multidimensional version of the Kiefer-Wolfowitz stochastic approximation algorithm. In M.D. Rossetti, R.R. Hill, B. Johansson, A. Dunkin, and R.G. Ingalls, editors, *Proceedings of the 2009 Winter Simulation Conference, Austin, Texas*, 2009.

[9] M. Broadie, D.M. Cicek, and A. Zeevi. General bounds and finite-time improvement for the Kiefer-Wolfowitz stochastic approximation algorithm. *To appear in Operations Research*, 2009.

[10] S. L. Brumelle and J. I. McGill. Airline seat allocation with multiple nested fare classes. *Operations Research*, 41:127–136, 1993.

[11] V. Dupac. O Kiefer-Wolfowitzově approximační methodě. *Časopis pro Pěstování Matematiky*, 82:47–75, 1957.

[12] J. M. Harrison and A. Zeevi. A method for staffing large call centers based on stochastic fluid models. *Manufacturing & Service Operations Management*, 7(1):20–36, 2005.

[13] J.C. Hull. *Options, Futures, and Other Derivatives*. Prentice Hall, New Jersey, 6^{th} edition, 2005.

[14] H. Kesten. Accelerated stochastic approximation. *The Annals of Mathematical Statistics*, 29(1):41–59, 1958.

[15] J. Kiefer and J. Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[16] S. Kim. Gradient-based simulation optimization. In L.F. Perrone, F.P. Wieland, J. Liu, B.G. Lawson, D.M. Nicol, and R.M. Fujimoto, editors, *Proceedings of the 2006 Winter Simulation Conference, Monterey, California*, 2006.

[17] H.J. Kushner and G.G. Yin. *Stochastic Approximation and Recursive Algorithms and Applications*. Springer-Verlag, New York, 2003.

[18] R. Merton. Option pricing when underlying stock returns are discontinuous. *J. Financial Economics*, 3:125–144, 1976.

[19] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM Journal on Optimization*, 19(4):1574–1609, 2009.

[20] R. Pasupathy and S.G. Henderson. A testbed of simulation-optimization problems. In L.F. Perrone, F.P. Wieland, J. Liu, B.G. Lawson, D.M. Nicol, and R.M. Fujimoto, editors, *Proceedings of the 2006 Winter Simulation Conference, Monterey, California*, 2006.

[21] B.T. Polyak. New stochastic approximation type procedures. *Automat. i Telemekh*, 7:98–107, 1990.

[22] H. Robbins and S. Monro. A stochastic approximation method. *The Annals of Mathematical Statistics*, 22(3):400–407, 1951.

[23] J. C. Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[24] R. Vaidya and S. Bhatnagar. Robust optimization of random early detection. *Telecommunication Systems*, 33(4):291–316, 2006.

[25] G. van Ryzin and J. McGill. Revenue management without forecasting or optimization: An adaptive algorithm for determining airline seat protection levels. *Management Science*, 46(6):760–775, 2000.