Multiscale Chaotic SPSA and Smoothed Functional Algorithms for Simulation Optimization

Shalabh Bhatnagar
Department of Computer Science and Automation
Indian Institute of Science
Bangalore 560 012
India
shalabh@csa.iisc.ernet.in

Vivek S. Borkar
School of Technology and Computer Science
Tata Institute of Fundamental Research
Homi Bhabha Road
Mumbai 400 005
India

The authors propose a two-timescale version of the one-simulation smoothed functional (SF) algorithm with extra averaging. They also propose the use of a chaotic simple deterministic iterative sequence for generating random samples for averaging. This sequence is used for generating the N independent and identically distributed (i.i.d.), Gaussian random variables in the SF algorithm. The convergence analysis of the algorithms is also briefly presented. The authors show numerical experiments on the chaotic sequence and compare performance with a good pseudo-random generator. Next they show experiments in two different settings—a network of *M/G/1* queues with feedback and the problem of finding a closed-loop optimal policy (within a prespecified class) in the available bit rate (ABR) service in asynchronous transfer mode (ATM) networks, using all the algorithms. The authors observe that algorithms that use the chaotic sequence show better performance in most cases than those that use the pseudo-random generator.

Keywords: Smoothed functional algorithm, SPSA algorithm, chaotic iterative sequence, simulation optimization, hidden Markov model

1. Introduction

Perturbation analysis-type simulation optimization schemes have attracted considerable attention in recent years [1–4]. These schemes are typically used for obtaining optimal or near-optimal measures of performance in discrete event systems [5]. Traditional schemes, however, require data to be aggregated over certain regeneration epochs [2, 3]. Moreover, an interchange between the "expectation" and "gradient" operators is needed that severely constrains in most cases the class of systems to which such approaches are applicable. In Bhatnagar and Borkar [6, 7], two alternative schemes, each of which use two timescales, were proposed. In the algorithm in [6], the parameter is updated at deterministic epochs that are obtained using two timescales. These epochs are frequent initially while they become increasingly sparse as time progresses. On the other hand, in the algorithm in [7], the parameter is updated at each instant by using two different step-size schedules. Here, averaging is intrinsically achieved through the use of coupled stochastic recursions that are individually driven by different timescales. These algorithms do not require the above interchange between the gradient and expectation operators and are thus applicable to a much broader class of systems than those to which traditional perturbation analysis schemes apply. However, both of these schemes use finite difference gradient estimates that require $(N+1)$ parallel simulations to estimate an N-vector parameter, which makes these schemes less efficient computationally when N is large.

In Spall [8], the simultaneous perturbation stochastic approximation (SPSA) algorithm was proposed. It is

SIMULATION, Vol. 79, Issue 9, September 2003 xxx-xxx

DOI: xxx

a random-direction version of the Kiefer-Wolfowitz algorithm wherein all parameter components are simultaneously perturbed by generating N independent and identically distributed (i.i.d.) symmetric random variables that are distributed (commonly) according to the Bernoulli distribution. The form of the gradient estimate is such that the algorithm requires only two parallel simulations while updating all component directions at each instant. In Bhatnagar et al. [9], the SPSA variants of the algorithms in Bhatnagar and Borkar [6, 7] were developed. Numerical experiments showed a significant improvement in performance by using these variants. In high-dimensional settings, however, the variant of [7] has obvious advantages over the variant of [6] because of regular update epochs in the former. The Bernoulli random variables in SPSA algorithms (like those with other distributions) are usually generated by using a pseudo-random number generator. The quality of the estimates thus depends highly on the quality of the generated sequence of pseudo-random numbers [10, 11]. Furthermore, most good generators require a nontrivial amount of computation for generating a pseudo-random sequence. We propose in this article the use of a rapidly mixing chaotic sequence that uses a simple deterministic recursion that converges rapidly to the uniform distribution. Instances of its use are available in the chaos literature (see, e.g., Boyarski and Gora [12] and the references therein). We shall use this iterative sequence for generating the N i.i.d., Bernoulli random variables at each iteration. While the primary approach in this study corresponds to Bhatnagar and Borkar [7], we incorporate features of the approach in Bhatnagar and Borkar [6] by implementing the slower "parameter update" recursion along a subsample (even though evenly spaced) of the times at which the faster "data aggregation" recursion is updated. This is seen to improve performance.

Next we turn our attention to the smoothed functional scheme originally due to Katkovnik and Kulchitsky [13] (see also Rubinstein [14]). Here, the idea is to approximate the gradient of expected performance by its convolution with a multinormal distribution. Upon integrating by parts in the latter integral and readjusting terms, one obtains the same as a scaled version of the convolution of the multinormal distribution and the expected performance itself. Thus, by using this scheme, one essentially requires only one simulation for estimating the gradient of the average cost, irrespective of the parameter dimension N. In Bharath and Borkar [15], a two-timescale analog of this scheme using the algorithm in Bhatnagar and Borkar [7] was proposed, and experiments were shown for the scalar parameter case. In this study, we propose a variant of the above algorithm for vector parameters (particularly those of higher dimension), which has the added advantage of an extra averaging (by combining features of both Bhatnagar and Borkar [6, 7], as explained above) that is not present in the above algorithm. Furthermore, we propose the use of the chaotic iterative sequence described above for generating samples from the multinormal distribution at each iteration for averaging in the latter algorithm. Our focus in this study is on developing efficient algorithms that require less computation, particularly in high-dimensional settings. We show numerical experiments on the chaotic generator (proposed here for averaging) and show comparisons with a good pseudo-random generator (described in Park and Miller [16]). We observe that the chaotic generator shows comparable performance as the pseudo-random generator. Next we show numerical experiments on a network of *M/G/1* queues with feedback with parameters of varying dimensions using all the (above) proposed algorithms. We observe that in most cases, algorithms that use the chaotic generator for averaging show improved performance than those that use the pseudo-random generator. In particular, the variant of the SPSA algorithm that uses the chaotic generator shows the best performance among all algorithms. Finally, we also consider the problem of finding the closed-loop feedback optimal policies in the available bit rate (ABR) service in asynchronous transfer mode (ATM) networks. This problem has been treated in detail in Bhatnagar et al. [17]. We consider it here to provide numerical comparisons on the "optimal" feedback policies that each of our algorithms computes. We observe again here that the variant of the SPSA algorithm that uses the chaotic generator shows the best performance.

The rest of the article is organized as follows: in the next section, we present the problem formulation. In section 3, we present the rapidly mixing chaotic sequence that we use to generate uniform random numbers for averaging and summarize all the algorithms. Section 4 presents a brief analysis. Numerical experiments are presented in detail in section 5. Finally, section 6 provides the concluding remarks.

2. Problem Formulation

The process that we seek to optimize is a parameterized hidden Markov model (HMM) given by the coupled iterations:

$$X(n+1) = f(X(n), Y(n), \xi(n), \theta),$$

$$Y(n+1) = g(X(n), Y(n), \xi'(n), \theta), \qquad (1)$$

$n \geq 0$. Here $\{\xi(n)\}$, $\{\xi'(n)\}$ are i.i.d. sequences in $\mathcal{R}^l$, $\mathcal{R}^m$, respectively, independent of each other. Also $f : \mathcal{R}^p \times \mathcal{R}^q \times \mathcal{R}^l \times \mathcal{R}^N \to \mathcal{R}^p$ and $g : \mathcal{R}^p \times \mathcal{R}^q \times \mathcal{R}^m \times \mathcal{R}^N \to \mathcal{R}^q$ are measurable maps. $\{Y(n)\}$ is the observation process, and the process $\{X(n)\}$ is unobserved. For each fixed (parameter) $\theta \in \mathcal{R}^N$, the joint process $\{(X(n), Y(n))\}$ is assumed to be ergodic Markov with transition probabilities $p_\theta(x, y; dx', dy')$. In what follows, we make θ tunable; that is, the dynamics of (1) is replaced by

$$X(n+1) = f(X(n), Y(n), \xi(n), \widetilde{\theta}_n),$$

$$Y(n+1) = g(X(n), Y(n), \xi'(n), \widetilde{\theta}_n), \tag{2}$$

$n \geq 0$, and where $\widetilde{\theta}_n$ is the value of the parameter at instant n. We assume θ takes values in a compact rectangle $C \triangleq \prod_{i=1}^{N}[\theta_{i,\min}, \theta_{i,\max}] \subset \mathcal{R}^N$. Let $\mathcal{G}_n \triangleq \sigma(X(j), Y(j), \xi(j), \xi'(j), \widetilde{\theta}_j, j \leq n)$, $n \geq 0$, represent the natural filtration generated by $\{X(j), Y(j), \xi(j), \xi'(j), \widetilde{\theta}_j, j \geq 0\}$. We assume that any sequence $\{\widetilde{\theta}_j\}$ satisfies the following assumption:

(Assumption 1)

$$P(X(n+1) \in A, Y(n+1) \in B \mid \mathcal{G}_n) = p_{\widetilde{\theta}_n}(X(n), Y(n); A, B),$$

for any A, B Borel in $\mathcal{R}^p$ and $\mathcal{R}^q$, respectively.

Our aim is to find a $\theta \in C$ that minimizes the long-run average cost

$$J(\theta) = \lim_{M\to\infty} \frac{1}{M} \sum_{j=0}^{M-1} h(Y(j)), \tag{3}$$

where $h(\cdot)$ is a given bounded and continuous cost function. Note that (3) is well defined because of ergodicity of the process $\{(X(n), Y(n))\}$ for each fixed θ. The average cost $J(\theta)$ is assumed to satisfy the following assumption:

(Assumption 2)

$J(\theta)$ is continuously differentiable in θ.

Note that (A1) is required to make the tuning of $\{\widetilde{\theta}_n\}$ nonanticipative. All our stochastic approximation algorithms in the next section are easily seen to satisfy this requirement. (A2) is a standard assumption in stochastic gradient-based approaches. Verification of (A2) for many systems is, however, a nontrivial task. A possible approach is to show that the parameterized stationary distribution of the process $\{(X(n), Y(n))\}$ is continuously differentiable in θ. In the case of systems of finite state Markov processes, a sufficient condition for the latter is to show that the transition probability matrix is continuously differentiable in the parameter. The latter can be shown using the perturbation results of Schweitzer [18] (see, e.g., Bhatnagar et al. [17]). For systems with general state space, sufficient conditions for differentiability of the stationary distribution are available in Vazquez-Abad and Kushner [19]. These are, however, difficult to verify in practice.

Finally, our stochastic approximation algorithms (in the next section) use two timescales or step-size schedules $\{a(n)\}$ and $\{b(n)\}$, respectively, with $a(n), b(n) \geq 0$, $\forall n$. We make the following assumption on these sequences:

(Assumption 3)

$$\sum_{n=0}^{\infty} a(n) = \sum_{n=0}^{\infty} b(n) = \infty, \quad \sum_{n=0}^{\infty} a(n)^2, \sum_{n=0}^{\infty} b(n)^2 < \infty,$$

$$a(n) = o(b(n)).$$

(A3) is satisfied by many sequences—for instance, $\{a(n)\}$ defined by $a(0) = \hat{a} > 0$ and $a(n) = \hat{a}/n$, $n \geq 1$, and $\{b(n)\}$ defined by $b(0) = \hat{b} > 0$ and $b(n) = \hat{b}/n^{\alpha}$, for some $\alpha \in (1/2, 1)$.

3. Algorithms

Let $\Gamma(x)$ for $x = (x_1, \ldots, x_N)^T \in \mathcal{R}^N$ represent the projection of x onto the set C. Thus, $\Gamma(x) \triangleq (\Gamma_1(x_1), \ldots, \Gamma_N(x_N))^T$, with each $\Gamma_i(x_i)$, $i = 1, \ldots, N$ defined by $\Gamma_i(x_i) = \max(\min(x_i, \theta_{i,\max}), \theta_{i,\min})$. In Bhatnagar and Borkar [7], the following two-timescale stochastic approximation algorithm was proposed and analyzed:

Algorithm 1

- Step 0 (initialize): Fix $\theta_1(0), \theta_2(0), \ldots, \theta_N(0)$ and form the parameter vector $\theta(0) \triangleq (\theta_1(0), \ldots, \theta_N(0))^T$. Fix M large and set $n := 0$. Form N parameter vectors $\theta^1(0) \triangleq (\theta_1(0) + \delta, \theta_2(0), \ldots, \theta_N(0))^T, \ldots, \theta^N(0) \triangleq (\theta_1(0), \ldots, \theta_{N-1}(0), \theta_N(0) + \delta)^T$. Set $Z(0) = Z^1(0) = \cdots = Z^N(0) = 0$.
- Step 1: Generate $(N+1)$ parallel simulations $(X(n), Y(n))$, $(X^1(n), Y^1(n)), \ldots, (X^N(n), Y^N(n))$, respectively governed by $\theta(n), \theta^1(n), \ldots, \theta^N(n)$. Update

$$Z(n+1) = Z(n) + b(n)(h(Y(n)) - Z(n)),$$

$$Z^1(n+1) = Z^1(n) + b(n)(h(Y^1(n)) - Z^1(n)),$$

$$\vdots$$

$$Z^N(n+1) = Z^N(n) + b(n)(h(Y^N(n)) - Z^N(n)),$$

and for $i = 1, \ldots, N$,

$$\theta_i(n+1) = \Gamma_i\left(\theta_i(n) + a(n)\left(\frac{Z(n) - Z^i(n)}{\delta}\right)\right).$$

Next, set $n := n + 1$. If $n = M$, go to Step 2; else, form parameter vectors $\theta(n) \triangleq (\theta_1(n), \ldots, \theta_N(n))^T$, $\theta^1(n) \triangleq (\theta_1(n) + \delta, \theta_2(n), \ldots, \theta_N(n))^T, \ldots, \theta^N(n) \triangleq (\theta_1(n), \ldots, \theta_{N-1}(n), \theta_N(n) + \delta)^T$ and repeat step 1.

- Step 2 (termination): Terminate algorithm and output $\theta(M) \triangleq (\theta_1(M), \ldots, \theta_N(M))^T$ as the final parameter vector.

Note that because of the fact that $a(n) = o(b(n))$, the recursions $Z(n)$, $Z^i(n)$, $i = 1, \ldots, N$ average out faster than the parameter updates themselves; as a result, in the parameter update recursion (last equation in step 1 above), $Z(n), Z^i(n), i = 1, \ldots, N$ correspond to cost averages for the corresponding simulations. The algorithm terminates after M iterations for some large, arbitrarily chosen value of M. In practice, one can replace step 2 with other termination conditions if required. As mentioned, this algorithm requires $N + 1$ parallel simulations for any N-vector parameter like any Kiefer-Wolfowitz algorithm with forward finite difference gradient estimates. In Bhatnagar et al. [9], a two-simulation SPSA variant was developed that requires only two parallel simulations.

3.1 SPSA and C-SPSA Algorithms

The basic algorithm is as follows:

Algorithm 2

- Step 0 (initialize): Set $Z^-(0) = Z^+(0) = 0$. Fix $\theta_1(0)$, $\theta_2(0), \ldots, \theta_N(0)$ and form the parameter vector $\theta(0) = (\theta_1(0), \ldots, \theta_N(0))^T$. Fix L and (large) M arbitrarily. Set $n := 0$ and $m := 0$. Generate i.i.d. random variables $\triangle_1(0), \triangle_2(0), \ldots, \triangle_N(0)$, with each $\triangle_j(0) = \pm 1$ w.p. $1/2$, $j = 1, \ldots, N$. Set $\theta_j^1(0) \stackrel{\triangle}{=} \theta_j(0) - \delta\triangle_j(0)$ and $\theta_j^2(0) \stackrel{\triangle}{=} \theta_j(0) + \delta\triangle_j(0)$, $j = 1, \ldots, N$, respectively.
- Step 1: Generate simulations (X^-_{nL+m}, Y^-_{nL+m}) and (X^+_{nL+m}, Y^+_{nL+m}), respectively governed by $\theta^1(n) \stackrel{\triangle}{=} (\theta_1^1(n), \ldots, \theta_N^1(n))^T$ and $\theta^2(n) \stackrel{\triangle}{=} (\theta_1^2(n), \ldots, \theta_N^2(n))^T$. Next update

$$Z^-(nL+m+1) = Z^-(nL+m) + b(n)(h(Y^-_{nL+m}) - Z^-(nL+m)),$$

$$Z^+(nL+m+1) = Z^+(nL+m) + b(n)(h(Y^+_{nL+m}) - Z^+(nL+m)).$$

  If $m = L - 1$, set $nL := nL + L$, $m := 0$ and go to step 2;

  else, set $m := m + 1$ and repeat step 1.

- Step 2: For $i = 1, \ldots, N$,

$$\theta_i(n+1) = \Gamma_i\left(\theta_i(n) + a(n)\left[\frac{Z^-(nL) - Z^+(nL)}{2\delta\triangle_i(n)}\right]\right).$$

  Set $n := n + 1$. If $n = M$, go to step 3;

  else, generate i.i.d. random variables $\triangle_1(n), \triangle_2(n), \ldots, \triangle_N(n)$ (independent of previous samples), with each $\triangle_j(0) = \pm 1$ w.p. $1/2$, $j = 1, \ldots, N$. Set $\theta_j^1(n) := \theta_j(n) - \delta\triangle_j(n)$, $\theta_j^2(n) := \theta_j(n) + \delta\triangle_j(n)$, $j = 1, \ldots, N$, and go to step 1.

- Step 3 (termination): Terminate algorithm and output $\theta(M) \stackrel{\triangle}{=} (\theta_1(M), \ldots, \theta_N(M))^T$ as the final parameter vector.

In the above, $Z^-(nL + m)$ and $Z^+(nL + m)$, $m = 0, 1, \ldots, L - 1$, $n \geq 0$, are defined according to their corresponding recursions in step 1 and are used for averaging the cost function in the algorithm. As stated earlier, we use only two simulations here for any N-vector parameter. Note that in the above, we also allow for an additional averaging over L (possibly greater than 1) epochs in the two simulations. Without an additional averaging, for large N, the scheme otherwise does not show good performance if the parameter is updated at every epoch. It was argued in Bhatnagar et al. [9] that perhaps because we are using only two simulations, the system finds it difficult to adapt to the new parameter update before it changes again. The choice of L is completely arbitrary, though. In all our experiments, we found $L = 100$ to be a good choice.

Generation of $\triangle_k(n)$, $k = 1, \ldots, N$, $n \geq 0$

Recall that the inverse transform technique is most commonly used for generating random variates with a known cumulative distribution function (c.d.f). Thus, for generating i.i.d., Bernoulli-distributed random variates $\triangle_k(n)$ above, one first generates i.i.d., uniform random numbers $U_k(n)$ on (0, 1) and sets

$$\triangle_k(n) = I\{U_k(n) \leq 0.5\} - I\{U_k(n) > 0.5\}, \qquad (4)$$

where $I\{\cdot\}$ is the indicator function. Thus, $\triangle_k(n)$, $k = 1, \ldots, N$, $n \geq 0$ are i.i.d., and

$$P(\triangle_k(n) = +1) = P(U_k(n) \leq 0.5) = 1/2,$$

$$P(\triangle_k(n) = -1) = P(U_k(n) > 0.5) = 1/2,$$

as desired. However, the quality of the $\triangle_k(n)$ samples crucially depends on the quality of the corresponding $U_k(n)$ samples. Moreover, generation of $U_k(n)$ samples usually requires considerable computation.

SPSA Algorithm

In steps 0 and 2 of Algorithm 2, we use the well-known linear congruential pseudo-random generator whose C-code is given below (see, e.g., Park and Miller [16, p. 1195]) for generating $U_k(n)$, $k = 1, \ldots, N$, $n \geq 0$, which in turn required for generating $\triangle_k(n)$ as described above.

Pseudo-Random Generator

```
double rnd(seed)        /* random number generator */
long *seed;
{
double u;
*seed=16807*(*seed%127773)-2836*(*seed/127773);
```

```
if (*seed<= 0)
*seed+=2147483647;
u = (double)*seed/2147483647;
return u;
}
```

It is easy to see that this generator requires substantial computation.

C-SPSA Algorithm

For generating $\triangle_k(n)$, $k = 1, \ldots, N$, $n \geq 0$, in steps 0 and 2, respectively, of Algorithm 2, we use a deterministic iterative sequence $\{\hat{U}_n\}$, defined as follows [12]:

Chaotic Generator

Choose $\hat{U}_0$ randomly (say, according to a given random-number generator). For any $n \geq 1$,

$$\hat{U}_{n+1} = (\pi + \hat{U}_n)^5 \bmod 1. \tag{5}$$

The sequence $\hat{U}_n$ is rapidly mixing and converges in distribution to an almost uniform distribution (see Boyarski and Gora [12] for details). Thus, the chaotic SPSA (C-SPSA) is similar to SPSA, except that here we generate $\triangle_k(n)$ samples in (4) using $\hat{U}_k(n) \stackrel{\triangle}{=} \hat{U}_{Nn+k}$, as in (5), in place of $U_k(n)$.

Next we describe the smoothed functional (SF) and chaotic smoothed functional (C-SF) algorithms.

3.2 SF and C-SF Algorithms

We explain the key idea first. For some scalar constant $\beta > 0$, let

$$DJ_\beta(\theta) = \int G_\beta(\theta - \eta)\nabla J(\eta)d\eta \tag{6}$$

represent the convolution of the gradient of average cost with the N-dimensional multinormal p.d.f.

$$G_\beta(\theta - \eta) = \frac{1}{(2\pi)^{N/2}\beta^N} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}\frac{(\theta_i - \eta_i)^2}{\beta^2}\right),$$

where $\theta, \eta \in \mathcal{R}^N$ with $\theta \stackrel{\triangle}{=} (\theta_1, \ldots, \theta_N)^T$ and $\eta \stackrel{\triangle}{=} (\eta_1, \ldots, \eta_N)^T$. Now integrating by parts in (5), it is easy to see that

$$DJ_\beta(\theta) = \int \nabla_\theta G_\beta(\theta - \eta)J(\eta)d\eta$$

$$= \int \nabla_\eta G_\beta(\eta)J(\theta - \eta)d\eta. \tag{7}$$

One can easily check that $\nabla_\eta G_\beta(\eta) = \frac{-\eta}{\beta^2}G_\beta(\eta)$. Substituting the last **[LAST PART?]** and $\eta' = \frac{\eta}{\beta}$ in (7), one obtains

$$DJ_\beta(\theta) = \frac{1}{\beta}\int -\eta' \frac{1}{(2\pi)^{N/2}} \exp\left(-\frac{1}{2}\sum_{i=1}^{N}(\eta_i')^2\right) J(\theta - \beta\eta')d\eta'. \tag{8}$$

In the above, we use the fact that $\eta = \beta\eta' = (\beta\eta_1', \ldots, \beta\eta_N')^T$ (written componentwise), and hence $d\eta = \beta^N d\eta_1' \cdots d\eta_N' = \beta^N d\eta'$. Upon substituting $\bar{\eta} = -\eta'$, the form of the gradient estimator suggested by (8) is (for M large) the following:

$$\nabla J(\theta(n)) \approx \frac{1}{\beta}\frac{1}{M}\sum_{n=1}^{M}\bar{\eta}(n)J(\theta(n) + \beta\bar{\eta}(n)),$$

where $\bar{\eta}(n) \stackrel{\triangle}{=} (\bar{\eta}_1(n), \ldots, \bar{\eta}_N(n))^T$, with $\bar{\eta}_i(n)$, $i = 1, \ldots, N$, $n \geq 0$, being independent $N(0, 1)$ distributed random variables.

We propose the following basic algorithm to implement the above:

Algorithm 3

- Step 0 (initialize): Set $Z_1(0) = Z_2(0) = \cdots = Z_N(0) = 0$. Fix $\theta_1(0), \ldots, \theta_N(0)$ and let $\theta(0) \stackrel{\triangle}{=} (\theta_1(0), \ldots, \theta_N(0))^T$ denote the initial parameter vector. Fix L, M, and β. Set $n := 0$ and $m := 0$, respectively. Generate i.i.d., $N(0, 1)$ distributed random variables $\eta_1(0), \eta_2(0), \ldots, \eta_N(0)$, and set $\eta(0) \stackrel{\triangle}{=} (\eta_1(0), \ldots, \eta_N(0))^T$.
- Step 1: Generate the simulation $(X(nL+m), Y(nL+m))$ governed with parameter $(\theta(n) + \beta\eta(n))$. Update

$$Z_1(nL + m + 1) = Z_1(nL + m) + b(n)$$
$$\left(\frac{\eta_1(n)}{\beta}h(Y(nL + m)) - Z_1(nL + m)\right),$$

$$Z_2(nL + m + 1) = Z_2(nL + m) + b(n)$$
$$\left(\frac{\eta_2(n)}{\beta}h(Y(nL + m)) - Z_2(nL + m)\right),$$

$$\vdots$$

$$Z_N(nL + m + 1) = Z_N(nL + m) + b(n)$$
$$\left(\frac{\eta_N(n)}{\beta}h(Y(nL + m)) - Z_N(nL + m)\right).$$

If $m = L - 1$, set $nL := nL + L$, $m := 0$ and go to step 2;
else, set $m := m + 1$ and repeat step 1.

- Step 2: For $i = 1, \ldots, N$, update $\theta_i(n)$ according to

$$\theta_i(n+1) = \Gamma_i(\theta_i(n) - a(n)Z_i(nL)).$$

Set $n := n+1$ and $\theta(n) \stackrel{\triangle}{=} (\theta_1(n), \ldots, \theta_N(n))^T$. If $n = M$, go to step 3;

else, generate i.i.d., $N(0,1)$ distributed random variables $\eta_1(n), \ldots, \eta_N(n)$, independent of previous samples. Set $\eta(n) \stackrel{\triangle}{=} (\eta_1(n), \ldots, \eta_N(n))^T$ and go to step 1.

- Step 3 (termination): Terminate algorithm and output $\theta(M) \stackrel{\triangle}{=} (\theta_1(M), \ldots, \theta_N(M))^T$ as the final parameter vector.

In the above, $Z_1(nL+m), \ldots, Z_N(nL+m)$, $m = 0, 1, \ldots, L-1$, $n \geq 0$, are defined according to their corresponding recursions in step 1 and are used for averaging the cost function in the algorithm. In Bharath and Borkar [15], a similar algorithm as above but without the additional averaging over L epochs was stated for the scalar case. We observed that for high-dimensional parameters (i.e., large N), the version without the extra averaging (i.e., $L = 1$) does not show good numerical performance. This could again be because of the fact (as argued in Bhatnagar et al. [9] for the case of SPSA with two simulations) that since we use only one simulation here, the system is unable to adapt to the new parameter update before it changes again. The value of L is chosen arbitrarily. We take it as 100 in our simulations. For generating the Gaussian random variates in the above basic algorithm, we use the Box-Muller [20] algorithm below:

Box-Muller Algorithm for $N(0,1)$ Generation

- Step 1: Generate uniform sample U_1 from $U(0,1)$ and set $\Theta = 2\pi U_1$.
- Step 2: Generate uniform sample U_2 from $U(0,1)$ and set $E = -\ln U_2$, $R = \sqrt{2E}$.
- Step 3: $X = R\cos\Theta$, $Y = R\sin\Theta$ are independent $N(0,1)$ samples.

We show briefly that the Box-Muller algorithm generates $N(0,1)$ samples. First assume that X and Y are given independent $N(0,1)$ random variables. Then (X, Y) has joint p.d.f. $\frac{1}{2\pi}\exp(-\frac{1}{2}(x^2+y^2))$. Transforming (X, Y) to polar coordinates (R, Θ), the joint p.d.f. becomes $\frac{1}{2\pi} r \exp(-\frac{r^2}{2})$ on $(0,\infty) \times (0, 2\pi)$ when R and Θ are independent. It is easy to see that $S = R^2 = X^2 + Y^2$ is exponentially distributed with mean 2. Thus, $U = \exp(-\frac{S}{2})$ is distributed $U(0,1)$. The above thus transforms (X, Y) to (U, Θ) with U and Θ being independent and uniform random variables on $(0,1)$ and $(0, 2\pi)$, respectively. Upon reversing this transformation, (X, Y) can be seen to be independent $N(0,1)$ random variables.

SF Algorithm

This algorithm is the same as Algorithm 3 but with the uniform random numbers U_1 and U_2 for obtaining the $N(0,1)$ samples (in steps 0 and 2 of Algorithm 3), using the Box-Muller scheme that is obtained according to the pseudo-random generator of Park and Miller [16] described above.

C-SF Algorithm

This algorithm is again the same as Algorithm 3 but with the uniform samples U_1 and U_2 in the Box-Muller scheme generated according to the chaotic iteration (5), described in the C-SPSA algorithm above.

We thus propose the use of the chaotic generator for purposes of averaging in our algorithms. We briefly analyze our algorithms in the next section.

4. Sketch of Convergence

Consider the following ordinary differential equation (ODE):

$$\dot{\theta}(t) = \bar{\Gamma}(-\nabla J(\theta(t))), \tag{9}$$

where $\bar{\Gamma}(\cdot)$ is defined by $\bar{\Gamma}(v(y)) = \lim_{\epsilon \to 0}\left(\frac{\Gamma(y + \epsilon v(y)) - y}{\epsilon}\right)$ for any bounded, continuous $v(\cdot)$. The operator $\bar{\Gamma}(\cdot)$ limits the trajectory of $\theta(\cdot)$ in (9) to evolve within the constraint set C. The ODE in (9) is interpreted componentwise with $\bar{\Gamma}(x) \stackrel{\triangle}{=} (\bar{\Gamma}_1(x_1), \ldots, \bar{\Gamma}_N(x_N))^T$ for $x = (x_1, \ldots, x_N)^T$. The stable points of this ODE lie in the set $K = \{\theta \in C \mid \bar{\Gamma}(\nabla J(\theta)) = 0\}$, with $J(\cdot)$ itself as the associated strict Liapunov function. Let for given $\eta > 0$, $K^\eta \stackrel{\triangle}{=} \{\theta \in C \mid \| \theta - \theta_0 \| < \eta, \theta_0 \in K\}$ denote the set of points that are within a distance η from the set K. For algorithms SPSA and C-SPSA, we have the following result (a similar result is available in Bhatnagar et al. [9] for the SPSA algorithm; however, we sketch some details here to explain the convergence for the C-SPSA algorithm).

THEOREM 1. Given $\eta > 0$, $\exists \delta_0 > 0$ (a function of the particular algorithm used) such that $\forall \delta \in (0, \delta_0]$, $\{\theta(n)\}$, defined by either of the algorithms SPSA and C-SPSA, converges to K^η a.s. as $M \to \infty$.

Proof. One can show, as in Theorem 4.4 of Bhatnagar et al. [9], that

$$\| Z^-(nL) - J(\theta(n) - \delta\triangle(n)) \|,$$
$$\| Z^+(nL) - J(\theta(n) + \delta\triangle(n)) \| \to 0 \text{ a.s.},$$

as $n \to \infty$. Thus, one can replace the parameter update recursion in Algorithm 2 (written componentwise) by

$$\theta_i(n+1) = \Gamma_i\Bigg(\theta_i(n) - a(n) \tag{10}$$

$$\left(\frac{J(\theta(n)+\delta\triangle(n)) - J(\theta(n)-\delta\triangle(n))}{2\delta\triangle_i(n)}\right) + a(n)\epsilon_1(n)\Bigg),$$

$i = 1, \ldots, N$, where $\epsilon_1(n)$ is $o(1)$. Now a Taylor series expansion in the second term in parentheses on the right-hand side (RHS) above yields

$$\frac{J(\theta(n)+\delta\triangle(n)) - J(\theta(n)-\delta\triangle(n))}{2\delta\triangle_i(n)} = \nabla_i J(\theta(n))$$

$$+ \sum_{j=1, j\neq i}^{N} \frac{\triangle_j(n)}{\triangle_i(n)} \nabla_j J(\theta(n)) + o(\delta),$$

where $o(\delta)$ represents the higher order terms. Now for $\mathcal{F}_n \stackrel{\triangle}{=} \sigma(X(m), Y(m), \theta(m), \triangle(m-1), m \leq n)$,

$$E\left[\frac{J(\theta(n)+\delta\triangle(n)) - J(\theta(n)-\delta\triangle(n))}{2\delta\triangle_i(n)} \mid \mathcal{F}_n\right]$$

$$= \nabla_i J(\theta(n))$$

$$+ E\left[\sum_{j=1, j\neq i}^{N} \frac{\triangle_j(n)}{\triangle_i(n)} \nabla_j J(\theta(n)) \mid \mathcal{F}_n\right] + o(\delta)$$

$$= \nabla_i J(\theta(n)) + \sum_{j=1, j\neq i}^{N} E\left[\frac{\triangle_j(n)}{\triangle_i(n)}\right] \nabla_j J(\theta(n)) + o(\delta),$$

since $\nabla_j J(\theta(n))$, $j = 1, \ldots, N$, $j \neq i$, are measurable w.r.t. $\mathcal{F}_n$ and, by definition, $\dfrac{\triangle_j(n)}{\triangle_i(n)}$ is independent of $\mathcal{F}_n$. Now from the definition of $\{\triangle(n)\}$, $E\left[\dfrac{\triangle_j(n)}{\triangle_i(n)}\right] = 0, \forall j \neq i$. Thus, (10) can be written as

$$\theta_i(n+1) = \Gamma_i(\theta_i(n) - a(n)\nabla_i J(\theta(n)) + a(n)M(n+1)$$

$$+ a(n)\epsilon_1(n) + a(n)o(\delta)),$$

where $E[M(n+1) \mid \mathcal{F}_n] = 0$. If $\delta = 0$, a standard ODE analysis shows that $\{\theta(n)\}$, given by Algorithm 2, converges to K a.s. as $M \to \infty$. For $\delta > 0$, we can invoke the Hirsch lemma (Hirsch [21, Theorem 1, p. 339]) as in Bhatnagar et al. [9] to conclude that if δ is small enough (depending on η), the iterates converge a.s. to K^η as $M \to \infty$.

Note that in the above, the independence of $\dfrac{\triangle_j(n)}{\triangle_i(n)}$ w.r.t. $\mathcal{F}_n$ plays a crucial role in convergence. In algorithm SPSA, we use the pseudo-random generator described in Park and Miller [16] for generating $\triangle_k(n)$, $k = 1, \ldots, N$, $n \geq 0$, while in C-SPSA, we use the chaotic sequence (5) for the same. We show tests for independence and randomness using both generators (see section 5). The chaotic generator shows equivalent performance in the tests for independence as compared to the pseudo-random generator, and the values from these tests are close to the expected values. Hence, we conclude that the samples using both generators are independent. This completes the proof. □

Remark. The sequence $\{\hat{U}_n\}$, defined through the iterations (5), is shown to be rapidly mixing in Boyarski and Gora [12]. Thus, the foregoing convergence argument would go through, even without the explicit verification of independence, by the "averaging property" of stochastic approximation used in Bharath and Borkar [15] (see also Benveniste, Metivier, and Priouret [22]). This says that when an ergodic process enters the driving term of a stochastic approximation as a parameter, the limiting ODE sees it averaged w.r.t. its (unique) stationary distribution. The rapid mixing property also suggests that one could get closer to the independence hypothesis by using a subsample $\{\hat{U}_{nM}\}$ for some $M > 1$. We found that $M = 1$ works fine as is.

Next we turn our attention to the SF and C-SF algorithms. For these algorithms, we have the following convergence result.

THEOREM 2. Given $\eta > 0$, $\exists \beta_0 > 0$ (a function of the particular algorithm used) such that $\forall \beta \in (0, \beta_0]$, $\{\theta(n)\}$, defined by either of the algorithms SF and C-SF, respectively, converges to K^η a.s. as $M \to \infty$.

Proof. Let $Z(nL) \stackrel{\triangle}{=} (Z_1(nL), \ldots, Z_N(nL))^T$, with $Z_i(nL)$, $i = 1, \ldots, N$, as in Algorithm 3. From the discussion preceding Algorithm 3, one can show in a similar manner as in Theorem 4.4 of Bhatnagar et al. [9] that $\| Z(nL) - DJ_\beta(\theta(n)) \| \to 0$ as $n \to \infty$. Now note that since $G_\beta(\theta - \eta)$ in (6) converges to the Dirac-Delta functional in the limit as $\beta \to 0$,

$$\lim_{\beta \to 0} DJ_\beta(\theta) = \nabla J(\theta).$$

One can now replace the parameter update recursion in step 2 of Algorithm 3 (written in vector notation) by

$$\theta(n+1) = \Gamma(\theta(n) - a(n)\nabla J(\theta) + a(n)\gamma_1(n)$$

$$+ a(n)\gamma_2(n)),$$

where $\gamma_1(n) = \nabla J(\theta) - DJ_\beta(\theta)$, and $\gamma_2(n) = DJ_\beta(\theta) - Z(nL)$ is $o(1)$. Again, if $\beta = 0$, a standard ODE analysis shows that $\{\theta(n)\}$ converges to K a.s. If $\beta > 0$, one can again invoke the Hirsch lemma (Hirsch [21, Theorem 1, p. 339]), as in Borkar [23], to conclude that if β is small enough (depending on η), the iterates converge a.s. to K^η as the number of iterates $M \to \infty$.

Now in algorithm SF, we use the pseudo-random generator for generating the uniform random variables in the Box-Muller algorithm, while in algorithm C-SF, we use the chaotic generator for the same. Also, as shown in section 5.1, both chaotic and pseudo-random generators show comparable performance in terms of randomness and independence of samples, and thus we conclude that samples using both generators are i.i.d., $U(0, 1)$ distributed. □

This completes our discussion of the convergence analysis.

5. Numerical Experiments

We begin with some tests for the chaotic generator (5) and provide comparisons with the pseudo-random generator [16] described in section 4.

5.1 Experiments with Generators

We begin with the *gaps test for independence* (see Ripley [24, p. 43]) for both the chaotic and the pseudo-random generators: here one fixes constants $0 < \alpha < \gamma < 1$ and generates a string of numbers using (each of) the generators. Next one computes the lengths of substrings (of the individual strings) for which the successive generated numbers are outside the interval (α, γ). If the generated sequence is independent, the distribution of lengths of the substrings must be geometric with parameter $\gamma - \alpha$ and must also be independent.

We arbitrarily choose $\alpha = 0.4$ and $\gamma = 0.6$, respectively. Next we generate 10,000 samples using both the chaotic and the pseudo-random generators, respectively. We then perform the gaps test over disjoint strings, each of length 50 samples, and average over these. Thus, from the 10,000 samples in each case, we obtain 200 sets of values over which we average. Finally, we average over five different simulation runs starting with different initial seeds. The mean and standard error from these simulations for both the chaotic and pseudo-random generators, as well as the corresponding expected values of these, are shown in Table 1. It is easy to see that both the chaotic and pseudo-random generators show almost identical performance and have values that are quite close to the expected values for all cases. We thus conclude that both the chaotic and pseudo-random generators generate independent samples.

Next in Figures 1 and 2, we show plots of 10, 000 points in $[0, 1] \times [0, 1]$, each generated using the chaotic and pseudo-random generators, respectively. Also, Figures 3 and 4 show plots of the multinormal distribution (X, Y) with p.d.f. $\frac{1}{2\pi} \exp(-\frac{1}{2}(x^2 + y^2))$, each generated using the Box-Muller algorithm with chaotic and pseudo-random generators, respectively. It thus seems clear that the performance of the chaotic generator with a lesser computation is as good as the pseudo-random generator. Next we show experiments with the algorithms of this article.

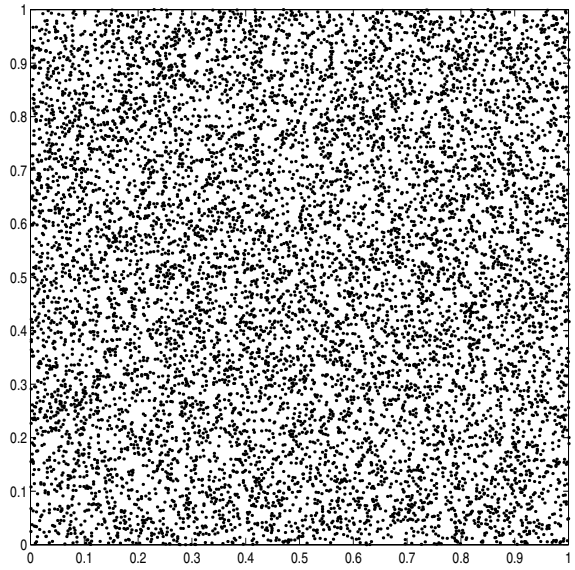

Figure 1. 10,000 points from the chaotic generator in [0,1] × [0,1]

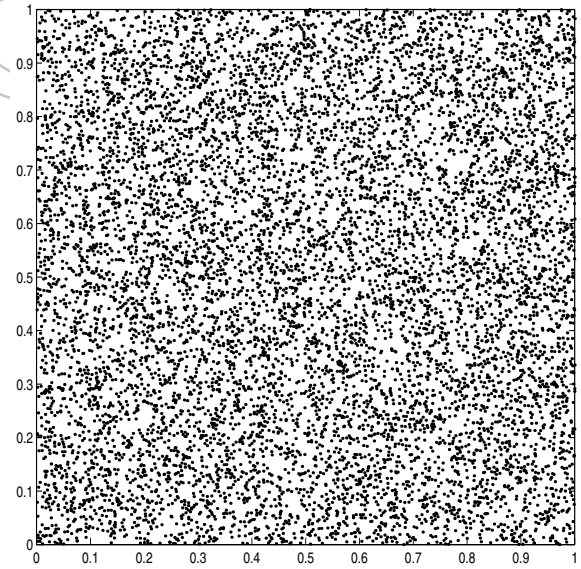

Figure 2. 10,000 points from the pseudo-random generator in [0,1] × [0,1]

5.2 Experiments with Algorithms

5.2.1 Network of M/G/1 Queues

We first consider a two-node network of *M/G/1* queues with feedback as in Figure 5. Nodes 1 and 2 in the network are fed with independent Poisson external arrival

Table 1. Gaps test for independence

Gap Length	0	1	2	3	4
Chaotic	2.16 ± 0.009	1.69 ± 0.016	1.34 ± 0.006	1.04 ± 0.012	0.82 ± 0.006
Pseudo-random	2.14 ± 0.016	1.70 ± 0.004	1.33 ± 0.002	1.04 ± 0.001	0.83 ± 0.003
Expected	2.4	1.92	1.54	1.23	0.98
Gap Length	**5**	**6**	**7**	**8**	**>8**
Chaotic	$0.65 \pm 0.0.006$	0.50 ± 0.001	0.40 ± 0.002	0.31 ± 0.024	1.10 ± 0.002
Pseudo-random	0.65 ± 0.033	0.50 ± 0.021	0.40 ± 0.002	0.31 ± 0.008	1.06 ± 0.041
Expected	0.79	0.63	0.50	0.40	1.61

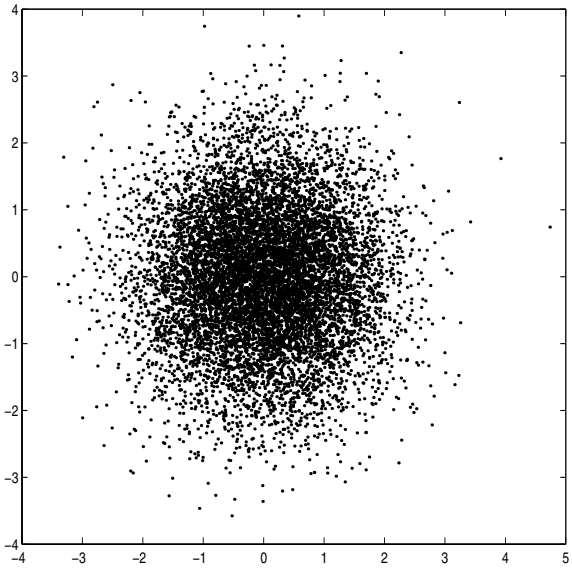

Figure 3. 10,000 Gaussian pairs (X, Y) using the chaotic generator

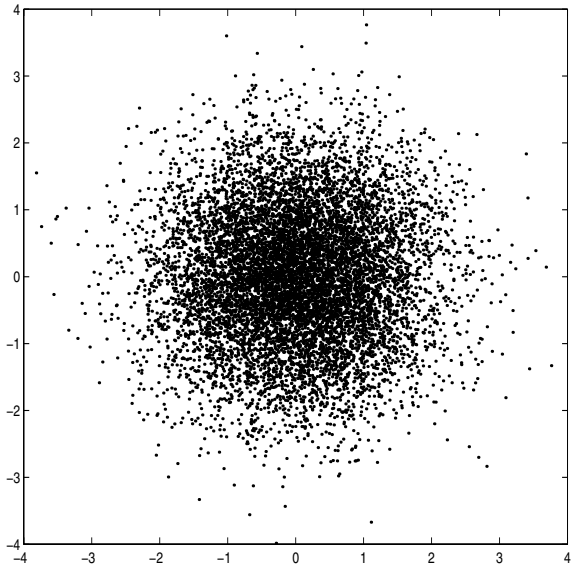

Figure 4. 10,000 Gaussian pairs (X, Y) using pseudo-random generator

streams with rates $\lambda_1 = 0.2$ and $\lambda_2 = 0.1$, respectively. The departures from node 1 enter node 2. Furthermore, departures from node 2 are fed back with probability $q = 0.6$ to the first node. The service time processes $\{S_n^i(\theta^i)\}$ at the two nodes $i = 1, 2$ are defined by

$$S_n^i(\theta^i) = U_n^i \left(1 + \prod_{j=1}^{P} |\theta_j^i(n) - \bar{\theta}_j^i|\right)/R_i,\ i = 1, 2,\ n \geq 1.$$

Here U_n^i represents a sample from the uniform distribution on [0, 1]. Also, $\theta_1^i(n), \ldots, \theta_P^i(n)$ represent the nth update of the parameter components of service time at node i, and $\bar{\theta}_1^i, \ldots, \bar{\theta}_P^i$ represent the target parameter components. We choose $R_1 = 10$ and $R_2 = 20$ to be given numbers. The cost function is chosen to be the sum of waiting times of individual customers at the two nodes. For the cost to be minimized, one expects $\theta_j^i(n)$ to converge to $\bar{\theta}_j^i$, $j = 1, \ldots, P$, $i = 1, 2$, as $n \to \infty$. We assume that each $\theta_j^i(n)$ is constrained according to $0.1 \leq \theta_j^i(n) \leq 0.6$, $j = 1, \ldots, P$, $i = 1, 2$, $\forall n$. We set $\bar{\theta}_j^i = 0.3$ for all $i = 1, 2$, $j = 1, \ldots, P$. The initial $\theta_j^1(0) = 0.4$, $j = 1, \ldots, P$, and $\theta_j^2(0) = 0.2$, $j = 1, \ldots, P$, respectively. The step-size sequences $\{a(n)\}$ and $\{b(n)\}$ are defined according to $a(0) = b(0) = 1$, $a(n) = 1/n$, $b(n) = 1/n^{2/3}$, $n \geq 1$. Also, $\delta = 0.1$ in all algorithms. We compare the performance of all algorithms SPSA, C-SPSA, SF, and C-SF, respectively. The value of L is chosen to be 100 in all algorithms.

We consider the cases $P = 2$ and 15, respectively. Thus, the parameter θ corresponds to 4 and 30 dimensional vectors, respectively. We define *distance from optimum* to be the performance measure $d(\theta(n), \bar{\theta}) \triangleq \left(\sum_{i=1}^{2}\sum_{j=1}^{P}\left(\theta_j^i(n) - \bar{\theta}_j^i\right)^2\right)^{1/2}$. From the previ-

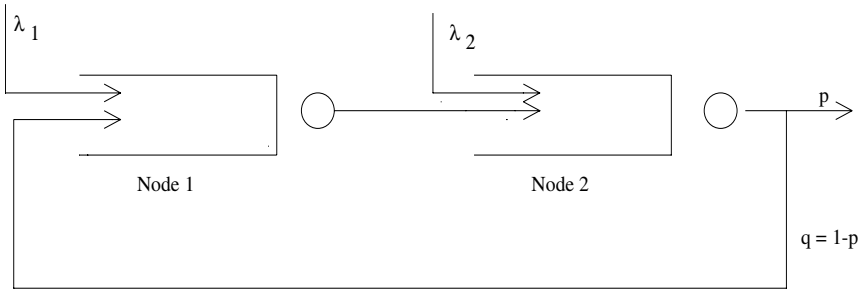

Figure 5. Queuing network

ous discussion, one expects $d(\theta(n), \bar{\theta})$ to converge to zero as $n \to \infty$. In Figures 6 and 7, we plot the trajectories of $d(\theta(n), \bar{\theta})$ for both cases, for all algorithms averaged over five independent simulation runs with different initial seeds. We terminate all simulation runs (for all algorithms) after 6×10^5 function evaluations (or measurements of the cost function value). Thus, in algorithms SF and C-SF, which require only one simulation for any value of P, the parameter is updated 6000 times during the course of the simulation. On the other hand, for algorithms SPSA and C-SPSA, which require two simulations each for any value of P, the parameter is updated 3000 times during the course of the simulation. The mean and standard error from these simulations for both cases $P = 2$ and 15, respectively, at the termination of the simulations are given in Table 2.

We observe from our experiments that C-SPSA shows the best performance among all algorithms, both when the parameter dimension is low (Fig. 6) and high (Fig. 7). Algorithm C-SF shows better performance than SF when $P = 2$, while the opposite is true for $P = 15$. The use of the chaotic generator thus improves performance in most cases. The two-simulation algorithms SPSA and C-SPSA show uniformly better performance than the one-simulation SF and C-SF algorithms over all cases tried.

5.2.2 Closed-Loop Feedback Control in ABR Service

We consider the setup in Figure 8. There are two input streams to the bottleneck node—a controlled stream, which is modelled as a controlled Poisson process, and an uncontrolled stream, which is a Markov-modulated Poisson process (MMPP). This problem finds applications in the closed-loop feedback control of ABR service in ATM networks [17]. Loosely stated, the basic goal of the control problem is that the queue should neither get empty (in which case, the server would idle), nor should there be a buffer overflow. In the ABR context, this amounts to maximizing throughput and minimizing delays simultaneously. The rate of the controlled Poisson process for the "next" T instants of time is computed using a feedback policy at the service end of the system, based on the queue length process, which is observed every nT, $n \geq 0$, time units. The computed rate information is fed back to the controlled Poisson (or the ABR) source with a delay D_b. Furthermore, there are propagation delays D_f for packets/customers from the controlled source to reach the node after being released. We assume the cost function to have the simple form $h(q_n) = |q_n - N_0|$, where N_0 is a given constant. We consider simple five-level feedback policies of the following type:

$$\lambda_c(n) = \begin{cases} \lambda_1 & \text{if} \quad q_n < N_0 - 2\epsilon \\ \lambda_2 & \text{if} \quad N_0 - 2\epsilon \leq q_n < N_0 - \epsilon \\ \lambda_3 & \text{if} \quad N_0 - \epsilon \leq q_n \leq N_0 + \epsilon \\ \lambda_4 & \text{if} \quad N_0 + \epsilon < q_n \leq N_0 + 2\epsilon \\ \lambda_5 & \text{if} \quad q_n > N_0 + 2\epsilon. \end{cases} \tag{11}$$

In the above, ϵ is also a constant in addition to N_0. The parameter to be optimized is thus $\theta \triangleq (\lambda_1, \ldots, \lambda_5)^T$. We use our stochastic approximation algorithms SPSA, C-SPSA, SF, and C-SF for performance comparisons in this setting. We choose the projection set C to be $[0.1, 3.0]^5$. The service times are assumed exponential with rate 1.0. We let $T = 1$, $D_b = 5$, $D_f = 10$, $N_0 = 10$, and $\epsilon = 1$, respectively. Also, $L = 100$ as before. We assume that the underlying Markov chain in the MMPP stream has just two states 1 and 2, respectively, with transition probabilities between these states given by $p_{11} = p_{22} = 0.4$ and $p_{12} = p_{21} = 0.6$, respectively. The rate of the Poisson process in the MMPP is 0.05 when in state 1 and 0.15 when in state 2. Thus, under stationarity, the uncontrolled

Table 2. Performance after 6×10^5 function evaluations

Algorithm	$d(\theta(n), \theta)$ $P = 2$	$d(\theta(n), \theta)$ $P = 15$
SPSA	0.0358 ± 0.017	0.1999 ± 0.0966
C-SPSA	0.0312 ± 0.0425	0.1646 ± 0.0176
SF	0.1345 ± 0.1070	0.3676 ± 0.0861
C-SF	0.1077 ± 0.0629	0.4045 ± 0.0952

Note. SPSA = simultaneous perturbation stochastic approximation; C-SPSA = chaotic SPSA; SF = smoothed functional; C-SF = chaotic SF.

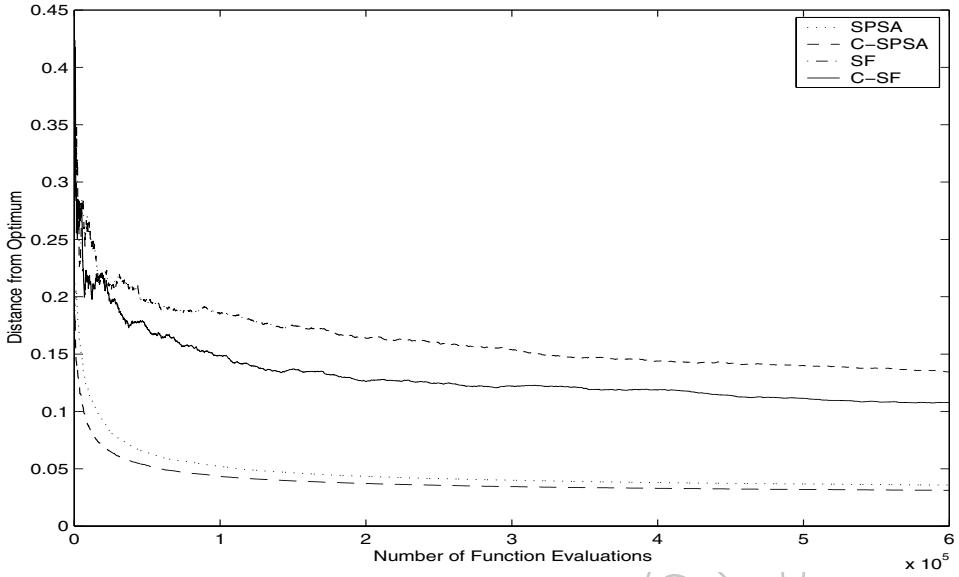

Figure 6. Performance comparisons for a 4-vector parameter

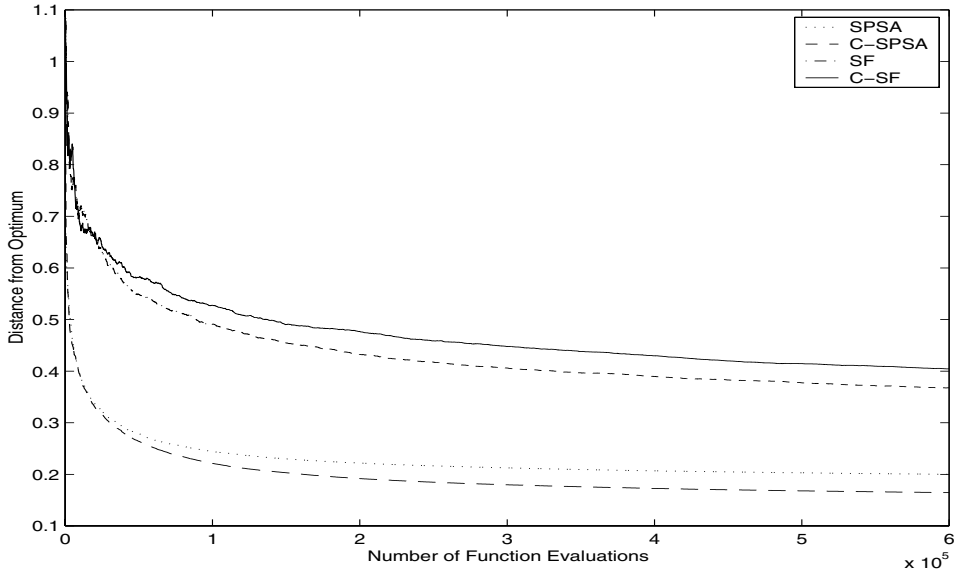

Figure 7. Performance comparisons for a 30-vector parameter

packets occupy 10% of the bandwidth. We run all simulations for 200,000 function evaluations for each algorithm. During the final 5000 evaluations, we also take estimates of the stationary mean queue length $E[q]$, the stationary variance of queue length $var(q)$, the stationary probability (P_B) that the queue length is in the band $[N_0-\epsilon, N_0+\epsilon]$, and the stationary mean ABR rate λ_c^*, respectively. We run all simulations with five different seeds and take averages. In Tables 3 and 4, we show the values of the optimal rates λ_1^*, $\ldots$, λ_5^*, as computed by the algorithms and the measures of performance, respectively.

Upon comparison of the various algorithms in this setting, it can be seen that C-SPSA shows the best performance and is closely followed by SPSA. Bandwidth utilization is a key performance criterion for ABR service, and using C-SPSA, more than 99% of the bandwidth

Table 3. [PLS. PROVIDE CAPTION]

Algorithm	λ_5^*	λ_4^*	λ_3^*	λ_2^*	λ_1^*
SPSA	0.14 ± 0.01	0.41 ± 0.02	1.28 ± 0.03	1.37 ± 0.03	2.11 ± 0.04
C-SPSA	0.19 ± 0.02	0.36 ± 0.02	1.18 ± 0.02	1.44 ± 0.03	2.23 ± 0.05
SF	0.16 ± 0.04	0.51 ± 0.05	0.97 ± 0.04	0.83 ± 0.04	1.99 ± 0.05
C-SF	0.10 ± 0.02	0.45 ± 0.04	0.87 ± 0.03	0.96 ± 0.03	1.91 ± 0.04

Note. SPSA = simultaneous perturbation stochastic approximation; C-SPSA = chaotic SPSA; SF = smoothed functional; C-SF = chaotic SF.

Table 4. [PLS. PROVIDE CAPTION]

Algorithm	*E[q]*	*var(q)*	P_B	λ_c^*
SPSA	10.95 ± 0.09	22.23 ± 0.08	0.25 ± 0.01	0.88 ± 0.01
C-SPSA	10.91 ± 0.08	21.62 ± 0.07	0.27 ± 0.01	0.89 ± 0.01
SF	7.14 ± 0.07	18.87 ± 0.07	0.20 ± 0.02	0.86 ± 0.01
C-SF	7.01 ± 0.05	17.51 ± 0.06	0.19 ± 0.02	0.85 ± 0.01

Note. SPSA = simultaneous perturbation stochastic approximation; C-SPSA = chaotic SPSA; SF = smoothed functional; C-SF = chaotic SF.

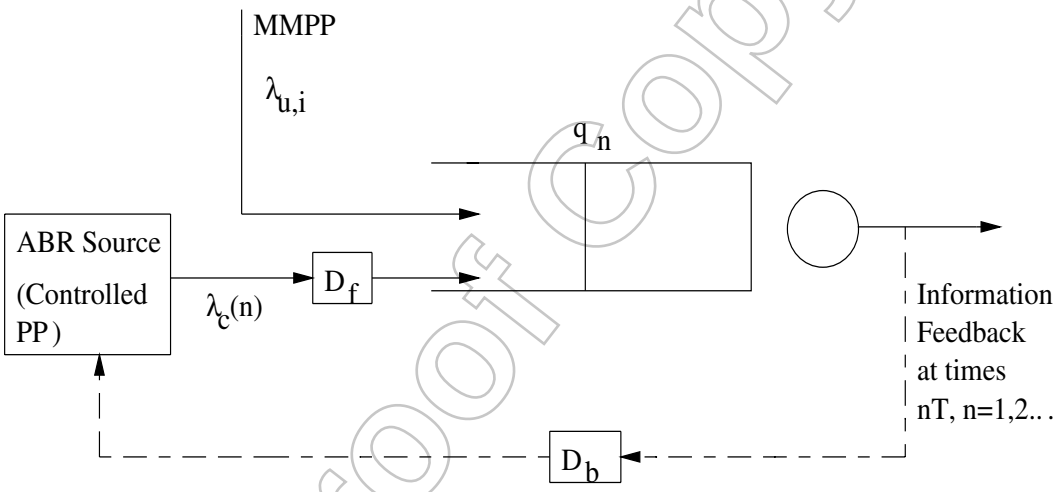

Figure 8. The available bit rate (ABR) model

(including the 10% from the uncontrolled stream) is used. SF and C-SF do not show as good performance except for in the variance of queue length measure, which has the least value in C-SF. However, the latter could also be because of the low stationary mean queue lengths $E[q]$ in both algorithms SF and C-SF.

6. Conclusions

We proposed the use of a chaotic iterative sequence for averaging. We considered the one-simulation smoothed functional algorithm and developed its two-timescale variant with an extra averaging required in high-dimensional settings. We used the chaotic iterative sequence for generating the N i.i.d., Gaussian random variables at each instant in the smoothed functional algorithm. We also used the chaotic sequence for generating the N i.i.d., Bernoulli-distributed perturbations at each instant in the two-timescale SPSA algorithm of Bhatnagar et al. [9]. We showed numerical experiments on the chaotic sequence and a good pseudo-random generator. We found that the chaotic sequence with less computation shows equivalent performance as the pseudo-random generator. We also showed experiments on a network of *M/G/1* queues with feedback using all the algorithms with both low- and high-dimensional parameters. We observed that in most cases, algorithms that use the chaotic iterative sequence for averaging show better performance than those that use the pseudo-random generator. Moreover, C-SPSA shows the best performance in both settings. Finally, we also applied

this approach to computing closed-loop feedback optimal (within a prespecified class) policies in ABR service in ATM networks. We observed that C-SPSA shows the best performance among all algorithms in this setting as well.

Recently, in Bhatnagar et al. [25], other variants of SPSA-type algorithms that use certain lexicographic and Hadamard matrix-based deterministic perturbation sequences have been developed. These were shown to improve performance over randomized perturbation SPSA under the settings considered therein. Also, in Spall [26], a one-simulation form of SPSA has been proposed with its two-timescale version developed in Bhatnagar et al. [25]. As future work, it would be interesting to see performance comparisons of these with the algorithms developed in this article, particularly C-SPSA and C-SF algorithms that use the chaotic iterative sequence, proposed here for random-number generation. Moreover, it would be interesting to explore the possibility of using chaotic iterative sequences in place of pseudo-random numbers in other simulation-based schemes such as *importance sampling* [27] and *neurodynamic programming* [28].

7. Acknowledgments

The first author thanks Professor G. Koole for inviting him to the Free University, Amsterdam, on a postdoctoral fellowship during 2000-2001. The second author thanks Professor G. Koole for his hospitality at Free University, Amsterdam, during a summer visit in the course of which this work was done.

8. References

[1] Chong, E. K. P., and P. J. Ramadge. 1993. Optimization of queues using an infinitesimal perturbation analysis-based stochastic algorithm with general update times. *SIAM Journal on Control and Optimization* 31 (3): 698-732.

[2] Chong, E. K. P., and P. J. Ramadge. 1994. Stochastic optimization of regenerative systems using infinitesimal perturbation analysis. *IEEE Transactions on Automatic Control* 39 (7): 1400-10.

[3] Ho, Y.-C., and X.-R. Cao. 1991. *Perturbation analysis of discrete event dynamical systems*. Boston: Kluwer.

[4] Glasserman, P., and Y.-C. Ho. 1990. *Gradient estimation via perturbation analysis*. Boston: Kluwer.

[5] Cassandras, C. G. 1993. *Discrete event systems: Modeling and performance analysis*. Boston: Aksen Associates and IRWIN.

[6] Bhatnagar, S., and V. S. Borkar. 1997. Multiscale stochastic approximation for parametric optimization of hidden Markov models. *Probability in the Engineering and Informational Sciences* 11:509-22.

[7] Bhatnagar, S., and V. S. Borkar. 1998. A two time scale stochastic approximation scheme for simulation based parametric optimization. *Probability in the Engineering and Informational Sciences* 12:519-31.

[8] Spall, J. C. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control* 37 (3): 332-41.

[9] Bhatnagar, S., M. C. Fu, S. I. Marcus, and S. Bhatnagar. 2001. Two timescale algorithms for simulation optimization of hidden Markov models. *IIE Transactions* 33 (3): 245-58.

[10] Niederreiter, N. 1995. New developments in uniform pseudorandom number and vector generation. In *Monte Carlo and quasi-Monte Carlo methods in scientific computing*, edited by H. Niederreiter and P. J.-S. Shiue. New York: Springer.

[11] L'Ecuyer, P. 1998. Random number generators and empirical tests. In **BOOK TITLE?**, edited by H. Niederreiter, P. Hellekalek, G. Larcher, and P. Zinthof. New York: Springer.

[12] Boyarski, A., and P. Gora. 1997. *Laws of chaos: Invariant measures and dynamical systems in one dimension*. Boston: Birkhauser.

[13] Katkovnik, V. Y. A., and Y. U. Kulchitsky. 1972. Convergence of a class of random search algorithms. *Automation Remote Control* 8:1321-6.

[14] Rubinstein, R. Y. 1981. *Simulation and the Monte Carlo method*. New York: John Wiley.

[15] Bharath, B., and V. S. Borkar. 1998. Robust parameter optimization of hidden Markov models. *Journal of the Indian Institute of Science* 78:119-30.

[16] Park, S. K., and K. W. Miller. 1988. Random number generators: Good ones are hard to find. *Communications of the ACM* 31:1192-1201.

[17] Bhatnagar, S., M. C. Fu, S. I. Marcus, and P. J. Fard. 2001. Optimal structured feedback policies for ABR flow control using two-timescale SPSA. *IEEE/ACM Transactions on Networking* 9 (4): 479-91.

[18] Schweitzer, P. J. 1968. Perturbation theory and finite Markov chains. *Journal of Applied Probability* 5:401-13.

[19] Vazquez-Abad, F. J., and H. J. Kushner. 1992. Estimation of the derivative of a stationary measure with respect to a control parameter. *Journal of Applied Probability* 29:343-52.

[20] Box, G. E. P., and M. E. Muller. 1958. A note on the generation of random normal deviates. *Annals of Mathemathical Statistics* 29:610-1.

[21] Hirsch, M. W. 1989. Convergent activation dynamics in continuous time networks. *Neural Networks* 2:331-49.

[22] Benveniste, A., M. Metivier, and P. Priouret. 1990. *Adaptive algorithms and stochastic approximations*. Berlin: Springer Verlag.

[23] Borkar, V. S. 1997. Stochastic approximation with two time scales. *Systems and Control Letters* 29:291-4.

[24] Ripley, B. D. 1987. *Stochastic simulation*. New York: John Wiley.

[25] Bhatnagar, S., M. C. Fu, S. I. Marcus, and I.-J. Wang. 2003. Two timescale simultaneous perturbation stochastic approximations using deterministic perturbation sequences. *ACM Transactions on Modeling and Computer Simulation* 13 (2): 1-30.

[26] Spall, J. C. 1997. A one-measurement form of simultaneous perturbation stochastic approximation. *Automatica* 33:109-12.

[27] Srinivasan, R. 2002. *Importance sampling*. New York: Springer Verlag.

[28] Bertsekas, D. P., and J. N. Tsitsiklis. 1996. *Neuro-dynamic programming.* Belmont, MA: Athena Scientific.

Shalabh Bhatnagar *is* ***(POSITION?)*** *in the Department of Computer Science and Automation at the Indian Institute of Science, Bangalore, India.*

Vivek S. Borkar *is* ***(POSITION?)*** *in the School of Technology and Computer Science at the Tata Institute of Fundamental Research, Mumbai, India.*