Journal of Computational Mathematics, Vol.8, No.1, 1990, 45–54.

# CONVERGENCE SPEED AND ASYMPTOTIC DISTRIBUTION OF A PARALLEL ROBBINS–MONRO METHOD*

Zhu Yun-min[1)]
(*Institute of Mathematical Sciences, Chengdu Branch, Academia Sinica, Chengdu, China*)
Yin Gang[2)]
(*Department of Mathematics, Wayne State University, Detroit, USA*)

**Abstract**

Very recently, there is a growing interest in studying parallel and distributed stochastic approximation algorithms. Previously, we suggested such an algorithm to find zeros or locate maximum values of a regression function with large state space dimension in [1], and derived the strong consistency property for that algorithm. In the present work, we concern ourselves with the problem of asymptotic properties of such an algorithm. We will study the limit behavior of the algorithm and obtain the rate of convergence and asymptotic normality results.

## §1. Introduction

Very recently, there is a growing interest in studying parallel and distributed stochastic approximation algorithms. [1]–[5] proposed several such schemes. The purposes of the studies are to exploit the opportunities provided by parallel processing methods and take advantage of the asynchronous communication.

Motivated by [5], we suggested a parallel stochastic approximation algorithm in [1] to locate seros or maximum values of a regression function with large state space dimension. The methods of random truncations were employed in order to obtain the boundedness of the algorithm and to enssure the convergence. By the truncation techniques, we were able to treat a rather broad class of regression functions. The strong consistency property for the aforementioned algorithm was proved under rather weak conditions.

To study any kind of recursive algorithm, there are basically two questions that one wants to answer. First, one wants to see if the algorithm works (convergence); next, if the algorithm converges, one would like to find the convergence speed. In this paper, we will concern ourselves with the second problem, namely rate of convergence.

The algorithm suggested in [1] is a generalization of the relaxation method. Such an idea was originally given in [5]. There are two distinct features for the parallel RM algorithm proposed in [1]. First, there is no iteration number which is a common index to all the processors. Second, the computation intervals are random. Intuitively, we would expect that similar 'rate' of convergence result for the classical algorithm still holds for the parallel algorithms. However, because of the asynchronization and additional randomness (random computation times) coming in, the analysis is not straightforward. Since we must take account of available information for all the processors, the notations as well as the analysis are pretty complicated. One of the key points here is to overcome the difficulties of different

* Received May 25, 1987.
1) Research of this author was supported in part by Science Foundation of Academia Sinica.
2) Research of this author was supported in part by Wayne State University under the Wayne State University Research Award.

computation times for different processors. When evaluating the limit, we thus need to work on each processor separately.

The paper is organized as follows. In Section 2, the basic formulation and some conditions are stated. In Section 3, the result of $n^\delta x_n \to 0$ for some $\delta$, with $0 < \delta < \frac{1}{2}$, is obtained. Finally, in Section 4, some discussion on the asymptotic normality is derived.

## §2. The Algorithm

Let $x$ be an $r$-dimensional vector, $x = (x^1, \cdots, x^r)'$. Let there be $r$ processors, each controlling one component of the state vector. For each $i \le r$, let processor $i$ takes $y_j^i$ units of time to complete the $j$th iteration, where $\{y_j^i\}$ is a sequence of positive integer valued random variables which may depend on state and noise. Define $\tau_n^i$ by

$$\tau_0^i = 0, \quad \tau_n^i = \sum_{j=1}^{n} y_j^i.$$

$\{\tau_n^i\}$ is the random computation time. It is quite similar to the conventional renewal process. Let $\xi_{\tau_n^i}^i$ be the noise incurred in the $n$th iteration for $x^i$. Let $x_1 = (x_1^1, \cdots, x_1^r)'$ be the initial value and $x_{\tau_n^i}^i$ be the value of the $i$th component of the state at the end of the $n$th iteration (or $n$th processing.time). For $n \in [\tau_j^i, \tau_{j+1}^i)$, put

$$x_n^i = x_{\tau_j^i}^i,$$
$$\xi_n^i = \xi_{\tau_j^i}^i,$$
$$x_{\tau_n^i} = (x_{\tau_n^i}^1, \cdots, x_{\tau_n^i}^r)'.$$

Let $b(\cdot) : R^r \longmapsto R^r$ be a continuous function, $b(\cdot) = (b^1(\cdot), \cdots, b^r(\cdot))'$ and $\varepsilon_n = \frac{1}{n}$; the basic algorithm to be considered is

$$x_{\tau_{n+1}^i}^i = x_{\tau_n^i}^i + \varepsilon_{\tau_n^i}^i (b^i(x_{\tau_n^i}) + \xi_{\tau_n^i}^i), \quad i \le r. \tag{2.1}$$

**Remark.** Comparing with the classical RM algorithm here we emphasize parallel aspects of our algorithm. Starting with initial value, new values of $x^i$ are computed based on the most recently determined values of $x^j$, for $j \le r$. The newly computed values are passed to all other components of $x$. This is a generalization of the relaxation methods.

Since each processor takes a random time to complete each iteration, and this random time in general is different from processor to processor, there is no "iteration number" which is a common index for all the processors. This causes difficulties in both notation and analysis, and we have to use elapsed processing time (real time) as the time indicator.

To assume each processor to control only one component of the state vector is really no loss in generality. In fact, we could consider the case that each processor controls several components of the state vector. The notation would be more complicated, but the analysis essentially remains the same.

To proceed, we also need the following definitions.

$$\begin{aligned} N_i(n) &= \sup\{k; \tau_k^i \le n\}, \\ \Delta_n^i &= n - \tau_{N_i(n)}^i, \\ I_n^i &= I_{\{\Delta_n^i = 0\}}. \end{aligned} \tag{2.2}$$