Parameter Estimation for the Dirichlet-Multinomial Distribution

Amanda Peterson
Department of Mathematics and Statistics, University of Maryland, Baltimore County

May 20, 2011

Abstract. In the 1998 paper entitled *Large Cluster Results for Two Parametric Multinomial Extra Variation Models*, Nagaraj K. Neerchal and Jorge G. Morel developed an approximation to the Fisher information matrix used in the Fisher Scoring algorithm for finding the maximum likelihood estimates of the parameters of the Dirichlet-multinomial distribution. They performed simulation studies comparing the results of the approximation to the results of the usual Fisher Scoring algorithm, for varying dimensions of the parameter vector. In this study, parallel computing in R is utilized to extend the previous simulation studies to larger dimensions. Additionally, the Fisher Scoring algorithm and the direct numerical maximization of the maximum likelihood are compared.

Key words. Parallel Computing, Fisher Scoring Algorithm, Maximum Likelihood Estimation, Dirichlet-Multinomial

1 Introduction

Consider the Dirichlet-multinomial distribution which is defined as

$$P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho) = \frac{m!}{t_1!t_2!\cdots t_k!}\frac{\Gamma(c)}{\Gamma(m+c)}\frac{\prod_{i=1}^{k}\Gamma(t_i+c\pi_i)}{\prod_{i=1}^{k}\Gamma(c\pi_i)}, \tag{1.1}$$

where $c = \rho^{-2}(1-\rho^2)$, $0 < \rho < 1$, and Γ is the gamma function. Here $\mathbf{t}$ is a vector in $\mathbb{R}^k$ and represents counts for k categories where $t_k = m - \sum_{i=1}^{k-1} t_i$. Additionally, $\boldsymbol{\pi}$ is a vector in $\mathbb{R}^k$ such that $\pi_k = 1 - \sum_{i=1}^{k-1}\pi_i$ and π_i represents the probability associated with the i^{th} category.

As an example of its application, say that a question with k possible responses is given to a group of m people. Then T_i represents the number of people that chose the i^{th} response (t_i is the realization of T_i). If their responses are independent, then their response vector $\mathbf{T}$ is distributed multinomial. However, when the responses are *not* independent, then extra variation is introduced and multinomial is no longer valid. The Dirichlet-multinomial is widely used in the case of extra variation. For a discussion of the Dirichlet-multinomial distribution, refer to [3].

Now suppose that a set of data is given which is known to have Dirichlet-multinomial distribution, but the parameters $\boldsymbol{\pi}$ and ρ are not known. We can estimate these parameters using the given data and a number of different estimation techniques. Two such techniques are direct numerical maximization of the likelihood equations and the Fisher scoring algorithm (FSA), both of which we investigate in this report. The FSA requires the calculation of the Fisher information matrix, which in some cases can be cumbersome to compute. It turns out, however, that it is not too complicated to calculate for Dirichlet-multinomial. Additionally, we'll investigate the Approximate FSA proposed in [1], which is computationally more efficient than the FSA and works well for moderately large cluster size m.

In Section 2 we discuss the Fisher Scoring Algorithm using the Fisher information matrix and an approximate Fisher information matrix. Then we discuss the estimation technique of direct numerical maximization of the likelihood equations. In section 3, we describe a simulation that we ran for large k extending results in [1], which compares the use of the approximate Fisher information matrix to the exact Fisher Information matrix. Additionally, we discuss the parallelization of the simulation, which enabled the feasibility of the simulation for large k. In section 4, we provide the results of the simulation along with speedup and efficiency of our parallel implementation, showing that the parallel implementation is efficient. Additionally, we perform a timing comparison of the direct numerical maximization, Fisher Scoring algorithm, and the Fisher Scoring algorithm using an approximate Fisher information matrix.

2 Problem

2.1 Fisher Scoring Algorithm

Using the Fisher Scoring Algorithm (FSA), the parameter estimates at the $(i+1)$st iteration are

$$\boldsymbol{\theta}^{(i+1)} = \boldsymbol{\theta}^{(i)} + [\mathcal{I}(\boldsymbol{\theta}^{(i)})]^{-1} u(\boldsymbol{\theta}^{(i)}) \tag{2.1}$$

where $\boldsymbol{\theta}$ is a parameter vector in $\mathbb{R}^k$, $\mathcal{I}(\boldsymbol{\theta})$ is the Fisher information matrix, and $u(\boldsymbol{\theta})$ is the score function which is equivalent to the gradient of the log-likelihood function. In the case of the Dirichlet-multinomial, $\boldsymbol{\theta} = \begin{pmatrix} \boldsymbol{\pi} \\ \rho \end{pmatrix} \in \mathbb{R}^{k+1}$.

2.2 Fisher Information Matrix

In general, the Fisher information matrix is defined as

$$\mathcal{I}(\boldsymbol{\theta}) = E\left[\left(\frac{\partial}{\partial \boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(x)\right)\left(\frac{\partial}{\partial \boldsymbol{\theta}} \log f_{\boldsymbol{\theta}}(x)\right)^T\right] = -E\left[\frac{\partial^2}{\partial \boldsymbol{\theta}^2} \log f_{\boldsymbol{\theta}}(x)\right]$$

where $f_{\boldsymbol{\theta}}(x)$ is the likelihood function associated with the random sample $x_1, x_2, ..., x_n$.

The entries of the Fisher information matrix are calculated as follows for the Dirichlet-multinomial.

$$\begin{aligned}
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i^2}\right) &= c^2[E_{T_i}\{\Psi'_{T_i}(c\pi_i)\} + E_{T_k}\{\Psi'_{T_k}(c\pi_k)\}], i = 1,2,...,k-1 \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i \partial \pi_{i'}}\right) &= c^2 E_{T_k}\{\Psi'_{T_k}(c\pi_k)\}, i, i' = 1,2,...,k-1, i \neq i' \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i \partial \rho}\right) &= c[\pi_k E_{T_k}\{\Psi'_{T_k}(c\pi_k)\} - \pi_i E_{T_i}\{\Psi'_{T_i}(c\pi_i)\}]\left(\frac{2}{\rho^3}\right), i = 1,2,...,k-1 \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \rho^2}\right) &= \left[\sum_{i=1}^{k} \pi_i^2 E\{\Psi'_{T_i}(c\pi_i)\} - \Psi'_m(c)\right]\left(\frac{4}{\rho^6}\right), i = 1,2,...,k-1
\end{aligned} \tag{2.2}$$

Here $\Psi(x)$ and $\Psi'(x)$ are the digamma and trigamma functions which are defined as $\Psi(x) = (\partial/\partial x)\log\Gamma(x)$ and $\Psi'(x) = (\partial/\partial x)\Psi(x)$. Also, for any nonnegative integer t and real number $x > 0$, we can write $\Psi_t(x) = \Psi(x) - \Psi(t+x)$ and $\Psi'_t(x) = \Psi'(x) - \Psi'(t+x)$.

Each of the expectations in (2.2) are calculated using a summation, which adds to the complexity of calculating the Fisher information matrix. For example, consider the second equation in (2.2). We can re-write the expected value as

$$E_{T_k}\{\Psi'_{T_k}(c\pi_k)\} = \sum_{t_k=0}^{m}\{\Psi'(t_k + c\pi_k) - \Psi'(c\pi_k)\}P_{DM}(t_k;\boldsymbol{\pi},\rho).$$

2.3 Approximate Fisher Information Matrix

In [1] Neerchal and Morel proposed the following approximation to the elements of the Fisher information matrix for the Dirichlet-multinomial as $m \to \infty$.

$$\begin{aligned}
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i^2}\right) &\to c^2\{\Psi'(c\pi_i) + \Psi'(c\pi_k)\}, i = 1,2,...,k-1 \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i \partial \pi_{i'}}\right) &\to c^2\Psi'(c\pi_k), i, i' = 1,2,...,k-1, i \neq i' \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \pi_i \partial \rho}\right) &\to c\{\pi_k\Psi'(c\pi_k) - \pi_i\Psi'(c\pi_i)\}\left(\frac{2}{\rho^3}\right), i = 1,2,...,k-1 \\
E_T\left(-\frac{\partial^2 \log P_{DM}(\mathbf{t};\boldsymbol{\pi},\rho)}{\partial \rho^2}\right) &\to \{\sum_{i=1}^{k}\pi_i^2\Psi'(c\pi_i) - \Psi'(c)\}\left(\frac{4}{\rho^6}\right), i = 1,2,...,k-1
\end{aligned} \tag{2.3}$$

The Fisher information matrix needs to be calculated, whether by exact calculation or approximation, for each iteration of the FSA given in (2.1). If we use the approximate Fisher information matrix, then we refer to (2.1) as the Approximate Fisher Scoring Algorithm (AFSA).

Additionally, Neerchal and Morel performed a simulation study and provided results as shown in Tables 1 and 2, which demonstrate the error between FSA using the exact Fisher information matrix and AFSA using the approximated Fisher information matrix. The entries of Table 1 are the maximum over 5000 replications of

$$\frac{1}{k-1}\sum_{i=1}^{k-1}\frac{|\hat{\pi}_i - \tilde{\pi}_i|}{\hat{\pi}_i} \tag{2.4}$$

		Value of $\boldsymbol{\pi}$							
ρ	m	(.1)	(.3)	(.5)	$\begin{pmatrix}.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.5\end{pmatrix}$	$\begin{pmatrix}.3\\.5\end{pmatrix}$	$\begin{pmatrix}.1\\.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.2\\.3\end{pmatrix}$
.3	10	1.25	.844	.567	1.58	1.30	.700	1.58	1.30
	40	.443	.285	.182	.591	.534	.277	.644	.551
.7	10	6.05	.579	.318	3.82	4.08	.637	3.44	3.13
	40	1.62	.476	.207	2.91	2.87	.558	2.87	2.60

Table 1: Table as published in [1]. Monte Carlo Properties of the Error in Approximation for π_i's. Numbers in table are in terms of 10^{-4}.

		Value of $\boldsymbol{\pi}$							
ρ	m	(.1)	(.3)	(.5)	$\begin{pmatrix}.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.5\end{pmatrix}$	$\begin{pmatrix}.3\\.5\end{pmatrix}$	$\begin{pmatrix}.1\\.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.2\\.3\end{pmatrix}$
.3	10	139	12.9	11.9	18.4	9.41	7.92	8.17	6.35
	40	1.81	1.24	1.28	1.29	1.31	1.16	1.27	1.20
.7	10	6.92	.800	.675	.986	.932	0.917	1.12	1.06
	40	1.14	.410	.319	.536	.508	0.482	.627	.589

Table 2: Table as published in [1]. Monte Carlo Properties of the Error in Approximation for ρ. Numbers in table are in terms of 10^{-4}.

and the entries of Table 2 are the maximum over 5000 replications of

$$\frac{|\hat{\rho} - \tilde{\rho}|}{\hat{\rho}} \tag{2.5}$$

where $\hat{\pi}_i$ and $\hat{\rho}$ are the estimates from FSA and $\tilde{\pi}_i$ and $\tilde{\rho}$ are the estimates from AFSA. The maximum over 5000 replications gives us an idea about the worst case scenario.

Because of computing limitations, Neerchal and Morel were not able to test $k > 4$ as shown in Tables 1 and 2. In this study, we use a parallel computing architecture to extend Tables 1 and 2 and include values of k up to 15. We also study the performance of our parallel implementation.

2.4 Direct Numerical Maximization of the Likelihood

An alternative to FSA for finding the maximum likelihood estimates of the parameters is the direct numerical maximization of the likelihood equation. In general, the likelihood function of θ given n samples is

$$L(x;\theta) = \prod_{i=1}^{n} f(x;\theta) \tag{2.6}$$

where $f(x;\theta)$ is the distribution of x.

The goal is to maximize (2.6) with respect to θ. It can be difficult to maximize a product, so alternatively we consider the log-likelihood function which is defined as

$$l(x;\theta) = \sum_{i=1}^{n} \log(f(x;\theta)). \tag{2.7}$$

Since the logarithm function is a monotone increasing function, maximizing (2.7) is equivalent to maximizing (2.6).

3 Numerical Method

With the goal of replicating Tables 1 and 2 and extending them for larger values of k, we ran simulations using $n = 250$, where n is the sample size, and 5000 replications as was done in [1]. For each replication of the simulation, we randomly generated a new set of 250 data points generated from the Dirichlet-multinomial distribution. To do this, as described in [1], we perform the following steps:

1. Define the parameters n, m, $\boldsymbol{\pi}$, and ρ
2. Generate a probability vector from the Dirichlet distribution using `rdirichlet` from the R `MCMCpack` package and the parameters defined in step 1 as input values.
3. Draw the vector $\boldsymbol{t}$ from the multinomial distribution using the R function `rmultinom` given the probability vector generated in step 2.

In each replication we estimate the parameters of the randomly generated data by performing FSA and AFSA and then compute the difference using (2.4) and (2.5). At the end of the simulation we find the maximum of each over the 5000 replications. The stopping rule that we used for FSA and AFSA was

$$\|\boldsymbol{\theta}^{(i)} - \boldsymbol{\theta}^{(i-1)}\| < 10^{-6} \tag{3.1}$$

where $\boldsymbol{\theta}^{(i)}$ is the parameter estimate at the i^{th} iteration of the FSA.

In order to extend Tables 1 and 2 for larger k, it was important to parallelize the code. As part of the process of determining the best method for parallelization, we investigated the relationship between k and the number of iterations that the FSA requires to obtain the convergence defined in (3.1). As shown in Figure 1, the number of iterations that are required for FSA to converge suggest a trend that increases as k increases. The noise in the plot is due to the fact that the number of iterations is dependent on the random sample of data and the fact that we are only considering one FSA experiment for each k. This plot was generated using $n = 250$, $\rho = 0.7$, and $m = 40$. Note the FSA iterations themselves cannot be split among multiple processes since each iteration of the algorithm depends on the previous. The strategy that we chose for parallelization, was to split the replications of the simulation among the multiple processes.

The High Performance Computing Facility (HPCF, `www.umbc.edu/hpcf`) at UMBC houses an 86 node cluster, each node having 24 GB of memory and 8 cores split among 2 quad core Intel Nehalem X5550 processors (2.66 GHz, 8192 kB cache per core). We implemented the algorithm on the HPCF cluster using the statistical software R in conjunction with the R package snow. "snow" is an acronym for Simple Network Of Workstations, which is used to implement R code in a parallel environment. There are other R packages available for parallel computing, but an advantage of snow is that it provides a layer of abstraction so that the user does not need to define the communication

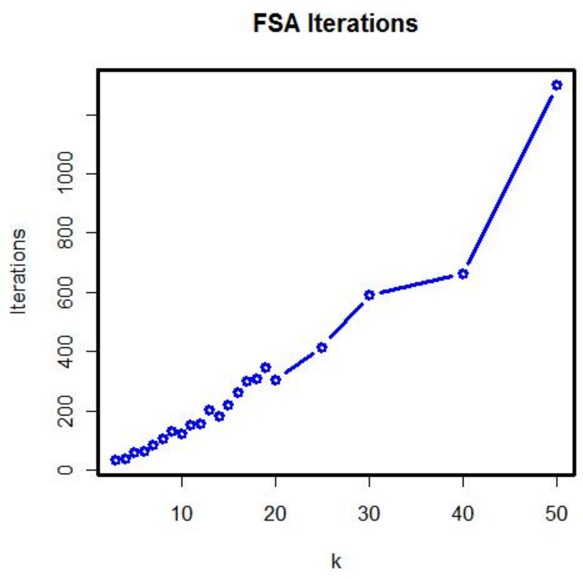

Figure 1: Relationship between k and the number of iterations required for FSA to obtain convergence

details. See `http://cran.r-project.org/web/views/HighPerformanceComputing.html` for the most recent information about parallel computing with R.

snow functions that were used in our parallel code are:

- `makeMPIcluster`: Make a cluster composed of p processes, where p was obtained by a call to the Rmpi function `mpi.universe.size`.
- `clusterExport`: Used to export all global variables and functions to all processes in the cluster.
- `clusterEvalQ`: Used to load necessary libraries on all processes in the cluster.
- `clusterSplit`: Used to split 5000 simulation replications among the p processes. If p does not evenly divide 5000, then the split is made to be as close to equal as possible.
- `clusterApply`: Used to run the simulation on each process.
- `stopCluster`: Shuts down the cluster and does cleanup.

Refer to [2] for a nice explanation of available snow functions and examples of their use.

4 Numerical Results

The results of our simulations are shown in Tables 3 and 4. The first five columns for $\boldsymbol{\pi}$ shown in the tables, which are associated with $k = 3$ and $k = 4$, can be compared to the corresponding

		Value of $\boldsymbol{\pi}$							
ρ	m	$\begin{pmatrix}.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.5\end{pmatrix}$	$\begin{pmatrix}.3\\.5\end{pmatrix}$	$\begin{pmatrix}.1\\.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.2\\.3\end{pmatrix}$	$k = 5$	$k = 10$	$k = 15$
0.3	10	40.1	.0329	.2310	.0294	.0125	.1030	.6300	.1270
	40	288	.0484	.0127	.0491	.0552	.0191	.2840	.0112
0.7	10	.0819	.0195	.0131	.0148	.0145	.0128	.1360	.0144
	40	.1590	.0136	.0082	.0128	.0127	.0116	.0080	.0089

Table 3: Monte Carlo Properties of the Error in Approximation for π_i's. Obtained via parallel implementation. Numbers in table are in terms of 10^{-4}.

		Value of $\boldsymbol{\pi}$							
ρ	m	$\begin{pmatrix}.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.5\end{pmatrix}$	$\begin{pmatrix}.3\\.5\end{pmatrix}$	$\begin{pmatrix}.1\\.1\\.3\end{pmatrix}$	$\begin{pmatrix}.1\\.2\\.3\end{pmatrix}$	$k = 5$	$k = 10$	$k = 15$
0.3	10	34.2	.2580	.4610	.2590	.3620	.5430	.6020	.2000
	40	368	.0504	.0158	.0296	.0374	.0142	.0570	.0007
0.7	10	.0181	.0039	.0040	.0041	.0027	.0023	.0119	.0010
	40	.0167	.0016	.0024	.0017	.0013	.0006	.0004	.0001

Table 4: Monte Carlo Properties of the Error in Approximation for ρ. Obtained via parallel implementation. Numbers in table are in terms of 10^{-4}.

columns in Tables 1 and 2. Additionally we have provided results for $\boldsymbol{\pi}$ vectors having length $k = 5$, $k = 10$, and $k = 15$.

The following values of $\boldsymbol{\pi}$ were used for $k \in \{5, 10, 15\}$:

- $k = 5$: $\boldsymbol{\pi}$ =(.1, .15 , .2, .25, .3)
- $k = 10$: $\boldsymbol{\pi}$ =(.05, .1, .05, .1, .25, .15, .05, .1, .05, .1)
- $k = 15$: $\boldsymbol{\pi}$ =(1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3)/30

Comparing the results for small k to Tables 1 and 2, many of the numbers are of different magnitudes. Differences may *possibly* be attributed to different FSA stopping rules (we did not verify the stopping rule used in [1]). Additionally, we are using a numerical approximation to the score function in (2.1) which differs from the exact calculation of the score function used in [1]. Regardless, however, these results show that the differences between the two methods are small even for k up to and including 15.

Table 5 shows the wall clock time in seconds, the observed speedup, and the observed efficiency for three values of k for the 5000 replication simulation. As recommended in [4], 8 processes per node were used for $p > 8$, and for $p \leq 8$ all processes were run on one node. Notice that the simulation for $k = 5$ using one process took 14.75 hours (53131.9 seconds). This was cut down to 7.8 minutes (468.2 seconds) using 256 processes.

(a) Walltime in seconds = $T_p(k)$									
k	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$
3	19652.4	9217.0	4591.5	3037.5	1796.5	937.5	454.4	271.9	127.1
4	34962.3	15828.0	8909.1	5708.5	2717.2	1487.1	764.0	489.8	170.8
5	53131.9	25672.2	13966.0	8980.6	6722.4	3620.2	1376.4	684.5	274.2
(b) Speedup: $S_p(k) = T_1(k)/T_p(k)$									
k	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$
3	1.0	2.1	4.3	6.5	10.9	21.0	43.2	72.3	154.6
4	1.0	2.2	3.9	6.1	12.9	23.5	45.8	71.4	204.7
5	1.0	2.1	3.8	5.9	7.9	14.7	38.6	77.6	193.8
(c) Efficiency: $E_p(k) = S_p(k)/p$									
k	$p = 1$	$p = 2$	$p = 4$	$p = 8$	$p = 16$	$p = 32$	$p = 64$	$p = 128$	$p = 256$
3	1.00	1.07	1.07	0.81	0.68	0.66	0.68	0.56	0.60
4	1.00	1.10	0.98	0.77	0.80	0.73	0.72	0.56	0.80
5	1.00	1.03	0.95	0.74	0.49	0.46	0.60	0.61	0.76

Table 5: (a) Wall clock time (seconds), (b) speedup, and (c) efficiency for 5000 replications varying k. 8 processes per node were used for $p > 8$, and for $p \leq 8$ all processes were run on one node.

Figure 2 shows the speedup and efficiency associated with Table 5. Figure 2a shows the observed speedup for $k \in \{3,4,5\}$ where speedup is defined as $S_p(k) = T_1(k)/T_p(k)$ and $T_p(k)$ is the wall clock time for k using p processes. Figure 2b shows the observed effiency for each value of k where efficiency is defined as $E_p(k) = S_p(k)/p$.

Finally, for comparison, we ran one simulation of 5000 replications using direction numerical estimation of the likelihood equation to estimate the parameters. We also ran a simulation using FSA only, and a simulation using AFSA only. For each experiment (i.e., simulation) we used $k = 3$. Direct numerical estimation was performed by passing the log-likelihood function, defined in (2.7), at each replication to the R function `optim`[1], which is an optimization function. We ran these experiments in serial mode (i.e., $p = 1$) and recorded the run times which are displayed in Figure 3. We can see in this figure that the direct numerical maximization was slightly faster than both FSA and AFSA, and AFSA was slightly faster than FSA. It is noteworthy, also that the estimation using the direct numerical maximization required the least amount of development time.

5 Acknowledgments

The author would like to thank Dr. Nagaraj Neerchal for providing the Dirichlet-multinomial problem and for answering numerous questions. Many thanks to Andrew M. Raim for valuable suggestions and seed code for the direct numerical maximization. Also, thanks to Dr. Matthias Gobbert for providing the instruction of parallel computing essentials.

The hardware used in the computational studies is part of the UMBC High Performance Computing Facility (HPCF). The facility is supported by the U.S. National Science Foundation through the MRI program (grant no. CNS–0821258) and the SCREMS program (grant no. DMS–0821311),

[1]The L-BFGS-B option was used, which is a limited-memory modification of the BFGS quasi-Newton method and allows for box constraints.

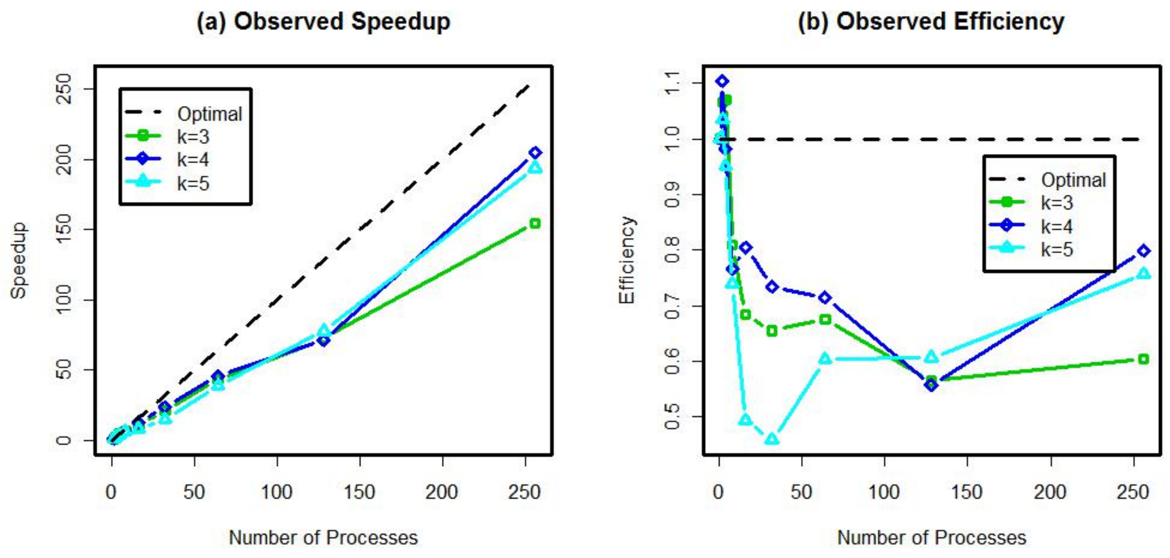

Figure 2: (a) Speedup and (b) Efficiency plots for various k

with additional substantial support from the University of Maryland, Baltimore County (UMBC). See `www.umbc.edu/hpcf` for more information on HPCF and the projects using its resources.

References

[1] Nagaraj K. Neerchal and Jorge G. Morel, *Large Cluster Results for Two Parametric Multinomial Extra Variation Models*, Journal of the Americal Statistical Association, Vol. 93, No. 443, pages 1078-1087, 1998.

[2] Sigal Blay, *snow Simplified*, `www.sfu.ca/~sblay/R/snow.html` .

[3] Norman L. Johnson, Samuel Kotz, and N.Balakrishnan, *Discrete Multivariate Distributions*, John Wiley & Sons, 1997.

[4] Matthias K. Gobbert, *Parallel Performance Studies for an Elliptic Test Problem.* Technical Report HPCF-2008-1, UMBC High Performance Computing Facility, University of Maryland, Baltimore County, 2008.

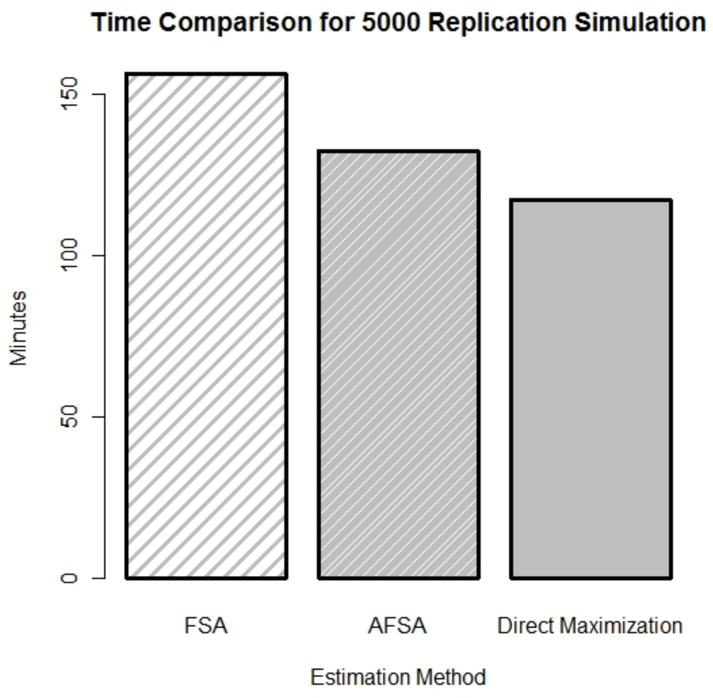

Figure 3: Time comparison of 3 simulations using different estimation methods