The Pennsylvania State University

The Graduate School

ESTIMATING MEANS USING EMPIRICAL LIKELIHOOD IN LARGE DATASETS

A Thesis in

Statistics

by

Maitraya Ghatak

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

August 2021

The thesis of Maitraya Ghatak was reviewed and approved by the following:

Nicole Lazar
Professor of Statistics
Thesis Advisor

G. Jogesh Babu
Distinguished Professor of Statistics, and Astronomy and Astrophysics

Ephraim Hanks
Associate Professor of Statistics
Chair of Graduate Studies

ABSTRACT

Empirical Likelihood does not suffer from model misspecification and provides robust estimates of means. However, computational inefficiency arises in large datasets, where multiple points must be tested on a grid at minute intervals. I identify how these computational inefficiencies can affect mean estimates and show that partitioning, subsampling, and certain other algorithmic methods can be used to obtain reliable estimates of the mean. I build three algorithms that rely on a two-part approach, which involves narrowing down the space of possible mean estimates in subsequent steps. I find that using this approach while also taking advantage of parallel computing can provide reliable estimates which converge very quickly. I discuss further modifications that can be used to extend these algorithms to other contexts.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I am very thankful to Dr. Nicole Lazar, my master's advisor, for her constant support and help throughout this process. She has been always very kind and helpful to me. I am also grateful to Dr. G. Jogesh Babu for his unwavering support since my entry into the graduate program in Statistics at PSU. I want to acknowledge to my friends, Mushan, Dario, Elle, Qi, and Shuo Shuo, among many others for their camaraderie and scholastic support. Finally, I thank my mother, without whose constant support I could not have finished my masters.

Chapter 1

Introduction to Empirical Likelihood

Maximum likelihood estimation plays a central role in statistical inference. Although the modern formulation was created by Ronald Fisher (1922), the fundamental ideas behind maximum likelihood have existed for decades before its formal introduction (Hald, 1999). Yet regardless of its long history and widespread usage, maximum likelihood suffers from the major shortcoming of model misspecification. Specifying an incorrect distribution can severely bias parameter estimates. To tackle such issues, various alternate likelihood functions have been developed over the years, such as, composite likelihood (Lindsay, 1988), bootstrap-based approaches (Efron and Tibshirani, 1994), etc. to name a few. One such non-parametric variant that has been very influential in statistical literature is empirical likelihood. Even though empirical likelihood benefits from its robustness arising from its non-parametric formulation, it suffers from computational inefficiencies. This thesis explores methods which can be used to more efficiently estimate univariate means using empirical likelihood.

Basic Theory

The theory of empirical likelihood was developed by Owen (1988, 1990) and expanded to more contexts through the use of estimating equations by Qin and Lawless (1994). As a non-parametric method, empirical likelihood does not involve any assumptions about data. Below I very briefly summarize the empirical likelihood derivation for the univariate mean (Lazar, 2021). Let $X_1, X_2, \ldots, X_n$ be independent and identically distributed observations from some distribution F_0. Note the empirical distribution function of these observations is given by

$$F_n(x) = \frac{1}{n}\sum_{i=1}^{n} 1(X_i \leq x),$$

for $-\infty < x < \infty$. The empirical likelihood function is given by

$$L(F_0) = \Pi_{i=1}^{n} dF_0(X_i) = \Pi_{i=1}^{n}(F_0(X_i) - F_0(X_i -)) = \Pi_{i=1}^{n} p_i,$$

where $F_0(X_i -) = P(X_i < x_i)$.

The empirical cumulative distribution function maximizes the likelihood function: $L(F_n) = \Pi_{i=1}^{n} \frac{1}{n}$. So, the empirical likelihood ratio is given by $R(F_0) = \frac{L(F_0)}{L(F_n)} = \Pi_{i=1}^{n} np_i$. Now, the interest is in estimating a parameter θ. To that end, the profile empirical likelihood is defined as

$$R_E(\theta) = sup(R(F_0)|\mathrm{T(F)} = \theta, F \in G),$$

where T(F) is a function of distributions, and F is a member of the set of distributions G. Then, the profile empirical ratio for the univariate mean is

$$R_E(\mu) = sup(\Pi_{i=1}^{n} np_i | p_i \geq 0, \sum_{i=1}^{n} p_i = 1, \sum_{i=1}^{n} p_i X_i = \mu),$$

where, a unique solution exists if μ is in the convex hull of the data. Otherwise, $R_E(\theta) = 0$ (Owen, 1988). To solve the optimization problem, Lagrange multipliers can be used. Subject to the given constraints, the maximum value of $R_E(\mu)$ can be attained when

$$p_i = \frac{1}{n(1+\lambda(\mathrm{X_i}-\mu))},$$

where λ is the solution to $\sum_{i=1}^{n} \frac{X_i - \mu}{1+\lambda(X_i-\mu)} = 0$. So, the maximum empirical likelihood estimate (MELE) is $\widehat{\theta_E} = max_{\theta} R_E(\theta)$.

Identifying the Issues Affecting Computational Efficiency

Empirical likelihood is solved numerically by evaluating the empirical likelihood function on a grid of values. The corresponding negative log likelihoods of the values in this grid are found, and the value which attains the lowest negative log likelihood ratio is said to be the empirical likelihood estimate. There are no underlying assumptions regarding a distribution or its corresponding support, and the parameter must be inside the convex hull. Hence, this grid must consist of values ranging from the minimum of the dataset to the maximum of the dataset when trying to compute the univariate mean, as the convex hull corresponds to the range of the dataset in one-dimensional space.

Two factors lead to computational inefficiency. The first issue arises due to the sheer size of a dataset. Empirical likelihood estimates for the mean in a dataset consisting of a hundred points will be computed much faster that one consisting of ten thousand such points. The second factor that leads to

increased computational times is the size of the grid of values which are tested. As a grid becomes finer, the estimates become more precise. Suppose data is drawn from a normal distribution with mean 1.15 and standard deviation of one. Also assume that the dataset ranges in values from -2 to 2. If only eleven points are tested at equal intervals of 0.4, {-2, -1.6, -1.2, -0.8, -0.4, 0, 0.4, 0.8, 1.2, 1.6, 2}, the true mean will never be tested, and the empirical likelihood estimate will be quite poor. In fact, at least 81 points need to be tested at equal intervals of 0.05 from the minimum to the maximum of the dataset for the point 1.15 to be tested. Hence, in many cases to capture the true mean, more points need to be tested, and the grid needs to be finer.

In large datasets, these two aforementioned factors are of major concern. As the dataset becomes larger, the expected number of values that exceed a certain range increases as well. For example, suppose data is generated from a normal distribution with mean zero and standard deviation of one. The expected number of values that will deviate from the mean by four or more standard deviations in a dataset of 10000 points is 0.64, but this expected number becomes 64 when considering a dataset of 1000000 points. Hence, as the dataset becomes larger, the number of points tested, or the grid, also becomes larger. To get precise estimates, this grid also has to be finer. Hence, both of the aforementioned problems become more severe in larger datasets.

Previous Work on Improving Computational Efficiency

Some of the ideas in this thesis are in part influenced by the ideas of Jaeger (2015), whose dissertation revolves around the creation of hybrid likelihood functions, which exhibit the favorable properties of different likelihood functions. Jaeger borrows from the ideas of composite likelihood (Lindsay, 1988) and empirical likelihood, and builds a hybrid likelihood function, which retains the favorable properties of empirical likelihood but does not sacrifice computational efficiency. In essence, Jaeger shows that by making splits along the parameter space, his likelihood function makes computation more tractable. I do not discuss the derivation of solutions or the asymptotic properties Jaeger's version but only briefly mention its basic formulation. In the following chapters, I do not use his likelihood function or its properties, but mostly draw inspiration from his ideas in my work.

Lindsay (1988) developed the theory of composite likelihood. Composite likelihood considers the weighted product of marginal or conditional likelihoods. Let $X_1, X_2, \dots, X_n$ be independent and identically distributed observations from some distribution F_0. For likelihood functions $L_j(\theta)$, where j $\in \{ 1, 2, \dots, J \}$, the composite likelihood is defined as $L_C(\theta) = \Pi_{j=1}^{J} L_j^{w_j}(\theta)$, where w_j are fixed weights corresponding to each $L_j(\theta)$, and $\widehat{\theta_C} = max_{\theta} L_C(\theta)$. Building on the machinery of both empirical and composite likelihood functions, Jaeger introduces the idea of a composite empirical likelihood function. Below I summarize his formulation for the bivariate case.

Let $X, Y \in R$ be random variables with joint distribution F_{xy}, and F_x, F_y exist. Let x have a sample size of n_x, and let y have a sample size of n_y. Assume (x_i, y_i) are independent and identically distributed, then the composite marginal empirical likelihood function is given by

$$L_C(F_{xy}) = L(F_x)L(F_y) = \Pi_i dF_X(x_i)\Pi_i dF_Y(y_i) = \Pi_i P(X = x_i)\Pi_i P(Y = y_i) = \Pi_i u_i \Pi_i v_i,$$

which is maximized when $u_i = 1/n_x, v_i = 1/n_y \; \forall i$. Using these weights, the composite empirical cumulative distribution function is given by

$$F_{n_x,n_y} = (\tfrac{1}{n_x}\textstyle\sum_{i=1}^{n_x} I(x_i < x))(\tfrac{1}{n_y}\textstyle\sum_{i=1}^{n_y} I(y_i < y)).$$

Just as estimating equations were required in the derivation of the univariate mean using empirical likelihood, we impose the estimating equations: $g(x_i, \mu_1) = x_i - \mu_1, g(y_i, \mu_2) = y_i - \mu_2$. Then, the bivariate composite empirical likelihood function is given by

$$L_{CE}(\mu) = sup\big(\Pi_i^{n_x} u_i \mid \textstyle\sum_{i=1}^{n_x} u_i g(x_i, \mu_1) = 0\,, u_i \geq 0, \sum_{i=1}^{n_x} u_i = 1\big)\; sup\big(\Pi_i^{n_y} u_i \mid \sum_{i=1}^{n_y} u_i g(y_i, \mu_2) =$$

$$0\,, u_i \geq 0, \textstyle\sum_{i=1}^{n_y} u_i = 1\big).$$

Jaeger follows Owen (1988, 1990) and Qin and Lawless (1994) to maximize the above equation and sets $\widehat{\theta_{CE}} = max_{\theta} L_{CE}(\theta)$.

In his dissertation, Jaeger finds that by splitting the likelihood function across the parameter space, and by creating this hybrid likelihood function, he is able to then recombine his estimates to find an overall estimate. On the other hand, I show that data can be split into subsets, and the corresponding empirical

likelihood estimates can be found. Then, these estimates can be combined to find the overall parameter estimate.

Chapter 2

Addressing Issues with Computation

I introduce some methods that can be used to address the two major issues affecting computational times, as mentioned in the last chapter. I partition the data into subsets using parallel computing, subsample from the partitions, and also detail a technique that can be used to narrow down the number of grid values that are tested.

Partitioning

I begin by addressing the first concern, which is to reduce computation time by limiting the number of points in the dataset. I begin by randomly separating the dataset into multiple equally sized partitions, where each core in a computer performs computations on one partition through the use of parallel computing. Then, for that partition, I compute the corresponding empirical likelihood estimate for the mean. Finally, I average all the empirical likelihood estimates computed in each partition to find the overall empirical likelihood estimate for the mean. This method is in part related to divide and conquer algorithms used in big data and machine learning (Wang et al., 2016); however, unlike in divide and conquer algorithms, I do not define any recurrence relations. I only divide the data into a certain number of parts and do not continue this subdivision indefinitely, as would be the case in recurrence relations in big data contexts. Furthermore, unlike in recurrence relations, no formal principles of induction are used in my proposed methods. The partitions do not need to be of equal size, and having equally-sized partitions does not provide any particular benefits. In case of partitions of unequal size, the weighted average of the estimates can be found, where the weight is proportional to the size of the partition. However, for simplicity and consistency throughout the thesis, I use partitions of equal size and find the unweighted average.

I compare my results to the maximum likelihood estimates. All computations in this thesis are done using the "emplik" package in R. Figure **1-1** presents these computations for the Poisson distribution. Here, I sample 50000 points from the Poisson(1) distribution, and I separate the data points into 50 partitions of 1000 points each. Then I compute the corresponding empirical likelihood estimates and find the unweighted

average of all these estimates to obtain an overall estimate for the mean. One hundred such 50000-point samples were generated, and their means were estimated in the aforementioned fashion. For each partition, the grid consisted of equidistant points spanning the range of the dataset at intervals of 0.05. In Figure **1-1**, each point corresponds to one such estimate from a 50000 point sample from the Poisson(1) distribution. For each one of these samples, I compute the corresponding maximum likelihood estimates for comparison.

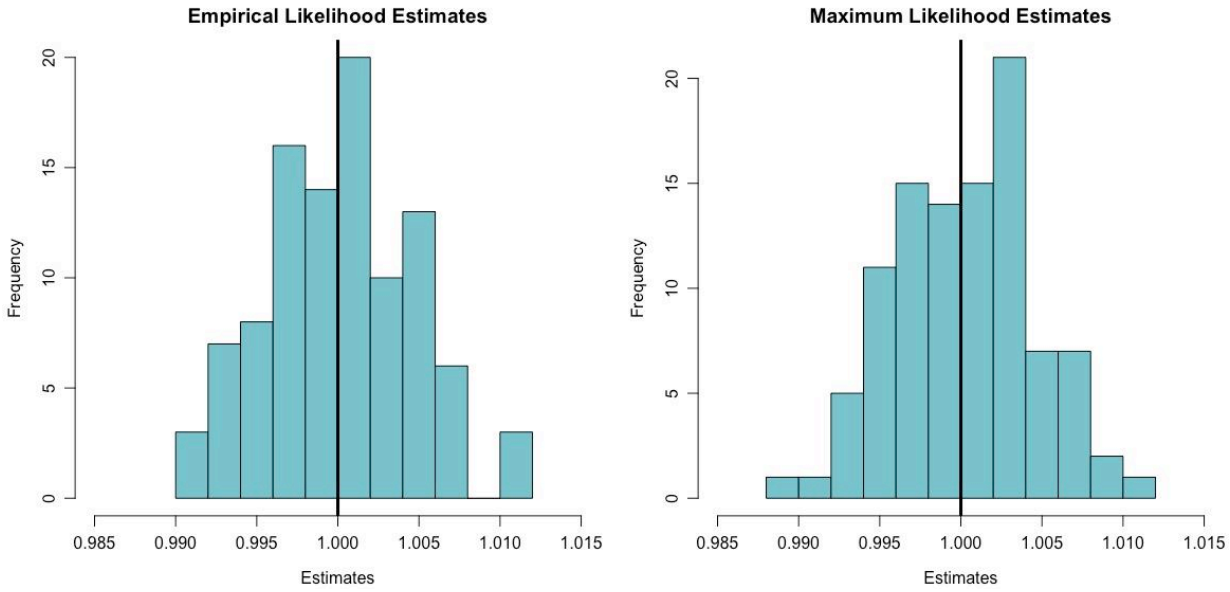

Figure **1-1**: ELE vs. MLE Estimates from Poisson (1)

I perform similar calculations for distributions not belonging to the exponential family as well. The maximum likelihood estimate for the location parameter of the Laplace distribution is its median. In Figure **2-1**, I have sampled from the Laplace(1, 1) distribution. 100 such 50000-point samples were generated, and the corresponding parameter estimates are provided in Figure **2-1**. The grid consisted of equidistant points spanning the range of the dataset at intervals of 0.05.

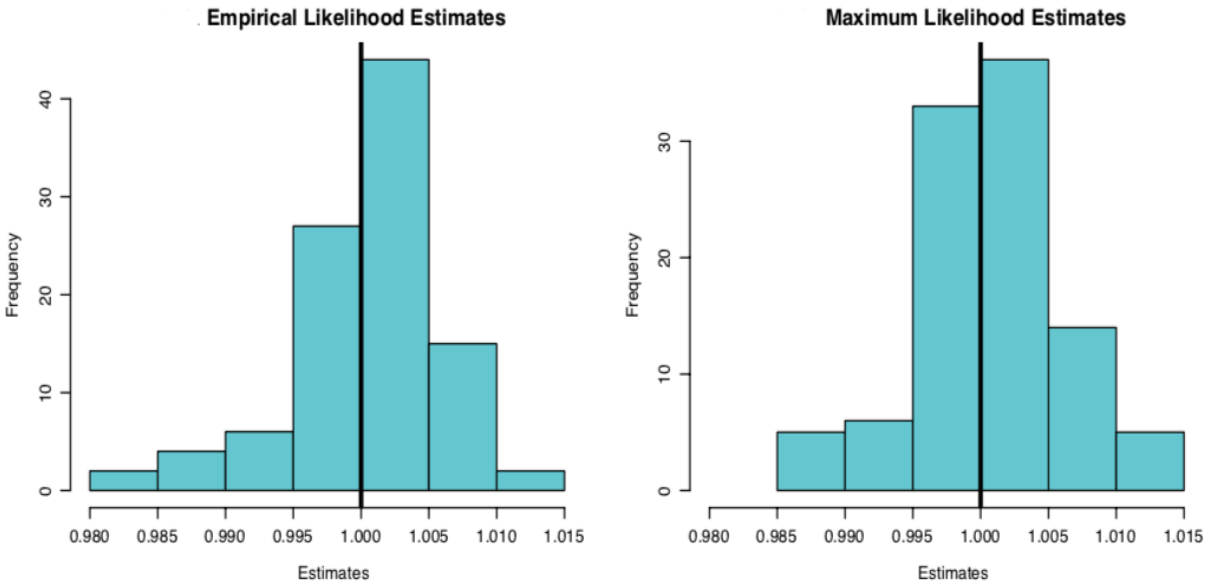

Figure **2-1**: ELE vs. MLE Estimates from Laplace (1,1)

Figures **1-1** and **2-1** demonstrate that the empirical likelihood estimates found by partitioning and recombining are quite close to the true parameter and are spread out similarly as the maximum likelihood estimates in Figure **1-1** and **2-1**. Most importantly, I find that if I separate a dataset into partitions and find the corresponding empirical likelihood estimates, then finding the average of all these empirical likelihood estimates yields valid estimates of the true mean, regardless of the central tendency of the distribution. This can be especially noted from Figure **2-1**, where finding the unweighted average of the empirical likelihood estimates provides good estimates of the location parameter, even though the maximum likelihood estimate for the location parameter is the median, not the mean. Although distributional assumptions do not guide empirical likelihood, I observe that partitioning and recombining empirical likelihood estimates provides estimates comparable to maximum likelihood estimates that rely on assumptions about a particular distribution.

Subsampling

Although delegating to multiple cores saves computation time, there may still be enough data points in each partition to drastically slow down computation. Hence, to address this issue, after I partition the dataset, I randomly sample 200 points from each partition and only perform computations on those 200

points. Each point in Figure **3-1** corresponds to a 50000-point sample from a standard normal distribution, which was further divided into 50 partitions of 1000 points each. However, this time, I only randomly sampled 200 points out of the 1000 points. Then, I found the corresponding empirical likelihood estimate of those 200 points, and then found the unweighted average of the empirical likelihood estimates corresponding to the 50 partitions. 100 such 50000-point samples were generated from the standard normal distribution, and the estimates computed using subsampling are presented in Figure **3-1**. For each partition, a grid interval of 0.05 was used.

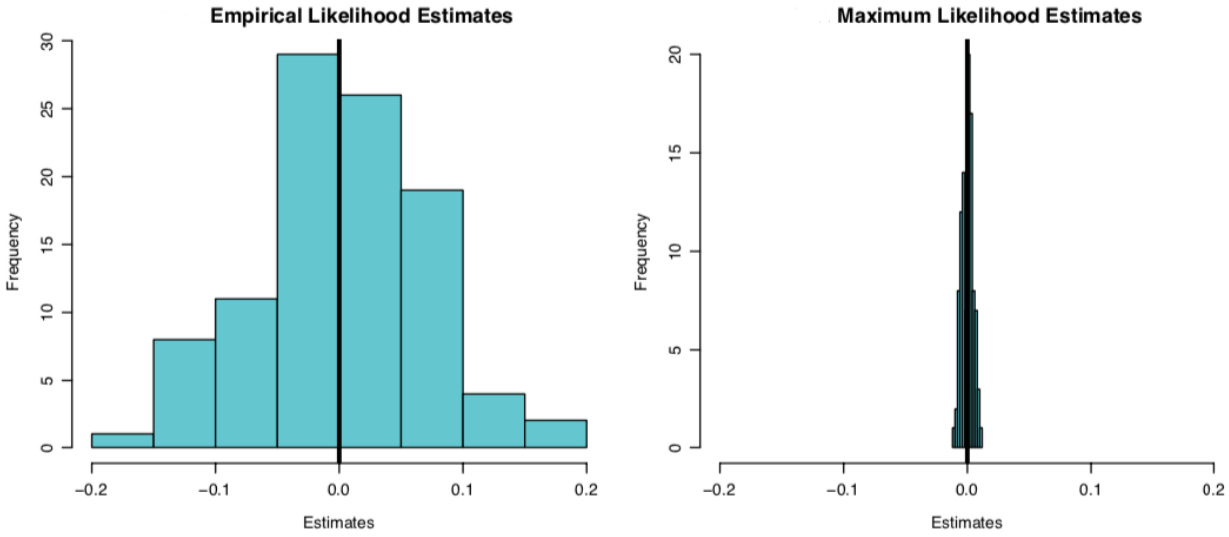

Figure **3-1**: ELE vs. MLE from Standard Normal Using Subsampling

Although the empirical likelihood estimates are close to the true mean of the distribution in Figure **3-1**, one major issue that is present in the figure is the larger variability in the empirical likelihood estimates compared to the maximum likelihood estimates. Hence, this figure indicates that there is trade-off between computational efficiency and precision. This issue is addressed in more detail in Chapter 4. However, for the most part, all the figures suggest that using partitioning and subsampling still provide valid estimates of the parameter of interest.

Managing the Grid of Points to be Tested

Now, I address the issue involving the grid of points for which the negative log-likelihood ratios need to be computed. Chapter 2 briefly describes how the empirical likelihood estimates become more

precise as the number of points in the grid increases. This phenomenon is graphically depicted in Figure **4-1**. As the number of points tested increases, the mean estimates become more accurate. Figure **4-1** includes four graphs, each depicting empirical likelihood mean computations on 1000 points generated from the exponential(3) distribution. However, in each respective image, the empirical likelihood estimate is computed by testing on an increasingly finer grid. For example, in the first image, the grid includes all points from the minimum of the 1000 points to the maximum of the 1000 points in intervals of 0.5. Hence, this is a very coarse grid, and there are very few points that are tested. 100 such 1000 point samples were generated from the exponential(3) distribution, and for each sample the empirical likelihood estimates were computed corresponding to grids with intervals 0.5, 0.1, 0.05, and 0.01 respectively. The horizontal line is situated at $\frac{1}{3}$, the true mean. For the image corresponding to intervals of 0.5, the empirical likelihood estimate is quite far from the true mean. As the grid becomes finer, the estimates approach the true mean. In fact, the image which corresponds to a grid consisting of points at intervals of 0.01 finds estimates closest to the true mean.

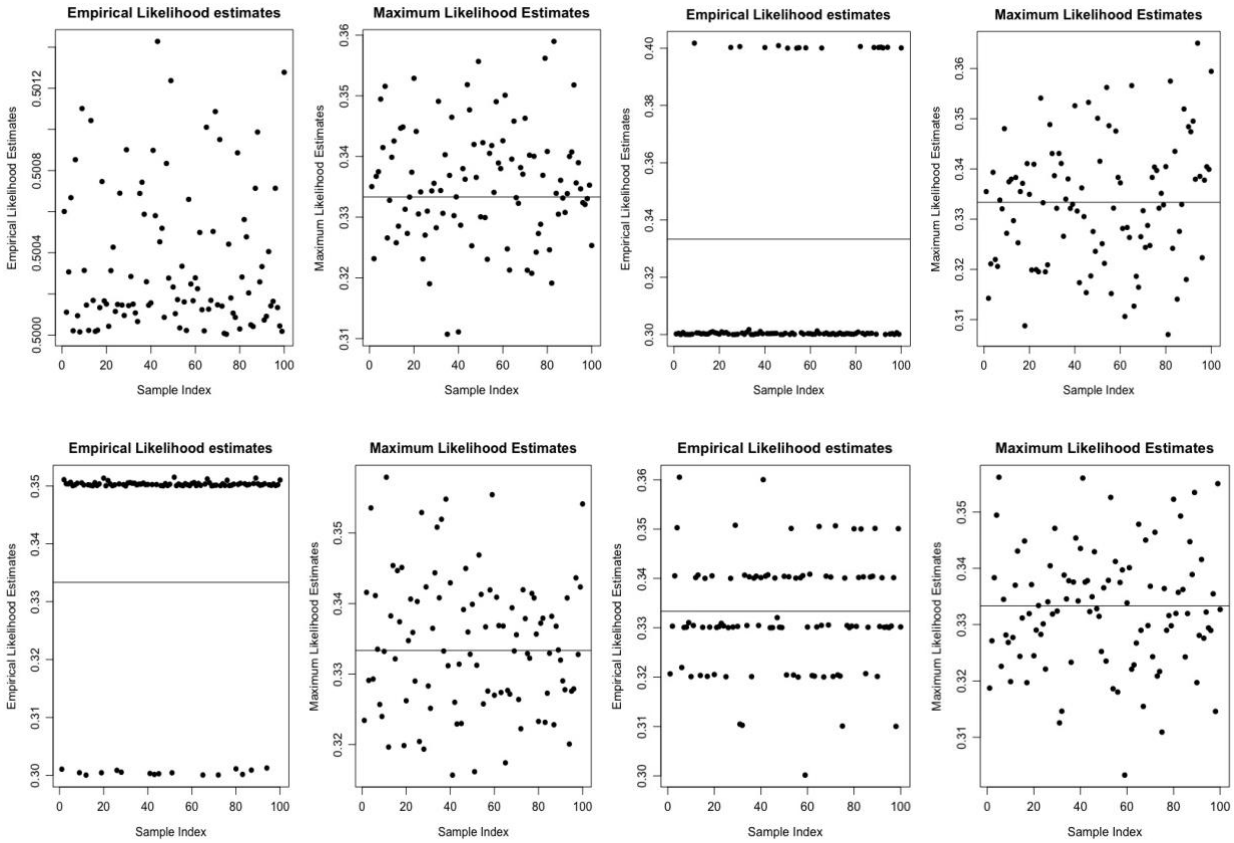

Figure **4-1**: ELEs for Exponential (3)-Intervals of 0.5 (Top-Left), 0.1 (Top-Right), 0.05 (Bottom-Left), and 0.01 (Bottom-Right)

Two-Part-Approach

As can be seen from the Figure **4-1**, the more points are tested, the more precise the estimates become. However, testing at minute intervals is very computationally intensive. For large datasets with a large range, it will be very inconvenient to test at intervals of 0.01 or less. To address the issue arising from the need to test at multiple points, I suggest a two-part approach. To begin, I find the range of a dataset X: $max(X) - min(X)$. Then, I divide this range by 200, which yields intervals of $\tau = \frac{max(X)-min(X)}{200}$. Then, I test at a total of 201 equidistant points in this range at intervals of τ. In this grid of 201 points (lets call this set Y), I pick only 20 points which yield the lowest negative log likelihood ratios. The second step of this two-part approach involves considering the range of these 20 points: $max(Y) - min(Y)$. Then, I divide

this range by 200, which yields intervals of $\rho = \frac{max(X)-min(X)}{200}$. Then, I test at a total of 201 equidistant points within this range at intervals of ρ. In this grid of 201 points, I pick the value yielding the lowest negative log likelihood ratio. This point is my estimate for the mean. Figure **5-1** shows empirical likelihood mean estimates computed for a standard normal distribution in such fashion, where 50000 points were generated, and the two-part approach was followed with 201 points tested in each round. 100 such 50000-point samples were generated, and their mean estimates are compared with the MLEs corresponding to these samples. Figure **5-1** shows that the estimates using the two-part approach are quite close to the true mean and have approximately the same spread as the maximum likelihood estimates.

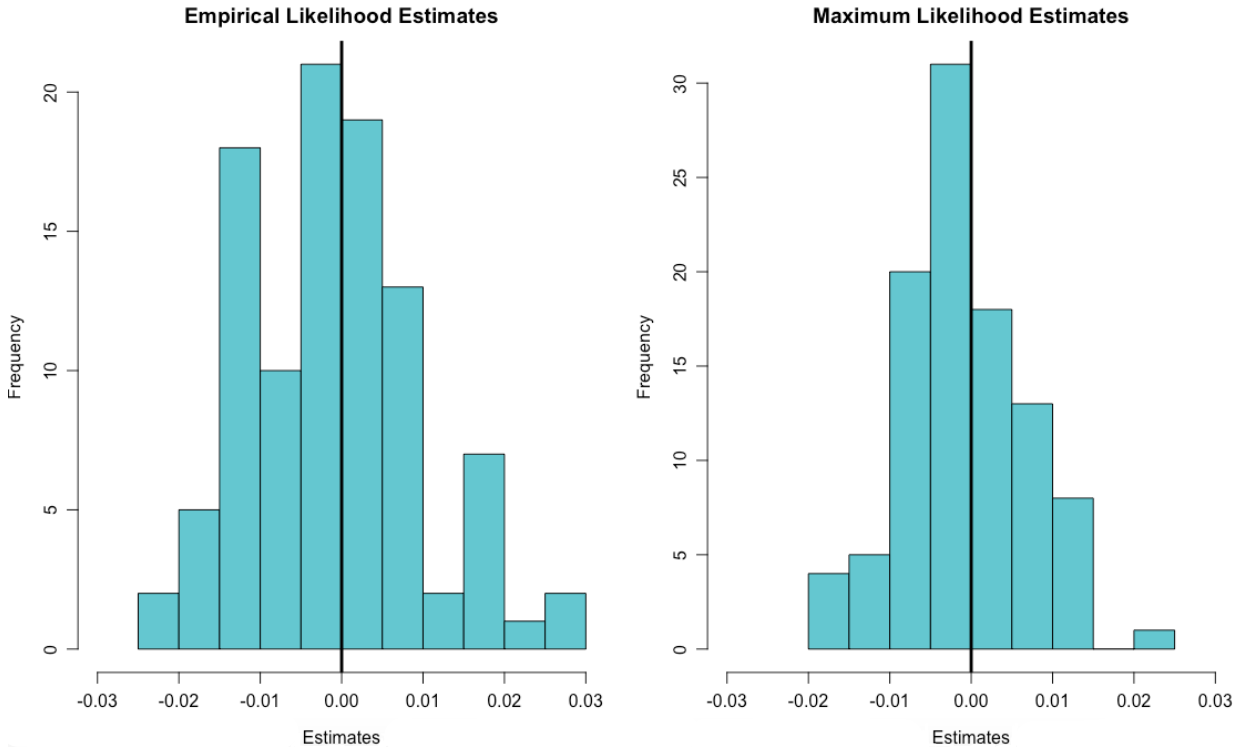

Figure **5-1**: Two-Part Approach on the Standard Normal Distribution

Chapter 3

Algorithms

Using the ideas developed in Chapter 2, I build three algorithms which can be used to mitigate the issues adversely affecting the computational speed. When comparing the two issues which affect computational times, I find that the problem arising from having to test at a large number of points in a grid affects the computational time more adversely than having a large number of data points in the dataset itself. Moreover, given that some computers may not be capable of parallel computing, I build one algorithm which does not use parallel computing and solely relies on limiting the grid of values which are tested. Hence, I suggest three approaches: the first two using parallel computing, and the third one only using the two-part approach without incorporating parallel computing.

Algorithm 1

Algorithm 1 incorporates partitioning, subsampling, and the two-part approach to address both the issues discussed in Chapter 2. However, the first part of the two-part approach is applied before any partitioning. In this way, fewer computations need to done in each core, which may be more computationally efficient.

Parallel Computing I**:**

i) Sample 500 points from the entire dataset and denote this set of points by X.

ii) Test X at intervals of $\frac{max(X)-min(X)}{200}$ from min(X) to max(X). Hence, I test X at 201 equally spaced-out points ranging in values from the minimum to the maximum of X.

iii) Pick 20 such points from the points tested, which yield the smallest negative log likelihood ratios. Denote this set of points by T.

iv) Note down all points from the minimum of T to the maximum of T by intervals of $\frac{max(T)-min(T)}{200}$. Call this set of 201 points Y.

v) Split the dataset into n partitions, X_i, where i takes values from 1 to n.

vi) Delegate one partition to each core.

vii) Sample 200 points from each partition X_i.

viii) Test the partition X_i at all the points of Y.

ix) The value in Y which has the lowest negative log likelihood is the empirical likelihood estimate corresponding to the partition i.

x) Average all the empirical likelihood estimates corresponding to all the partitions to find the overall empirical likelihood estimate for the mean.

Algorithm 2

Algorithm 2 is very similar to algorithm 1, except that both parts of the two-part algorithm are applied after partitioning. Hence, each core is more stressed in this algorithm.

Parallel Computing II:

i) Split the dataset into n partitions, X_i, where i takes values from 1 to n.

ii) Delegate one partition to each core.

iii) Sample 500 points from each partition X_i. Denote this set/ subsample of 500 points by Y_i.

iv) Test Y_i at intervals of $\frac{max(Y_i)-min(Y_i)}{200}$ from min(Y_i) to max(Y_i). That is, I test the subsample of 500 points, Y_i, at 201 equally spaced-out points from the minimum to the maximum of the subsample Y_i.

v) Pick 20 such points tested which yield the smallest negative log likelihood ratio. Denote this set of points by Z_i.

vi) Separate the set Z_i by intervals of $\frac{max(Z_i)-min(Z_i)}{200}$ from the min(Z_i) to max(Z_i). That is, I separate the set of 20 points, Z_i, in 201 equally spaced-out points from the minimum to the maximum of Z_i. Call this set W_i.

vii) Test Y_i at all the points of W_i.

viii) The element of the set W_i which has the lowest corresponding negative log likelihood is the empirical likelihood estimate corresponding to the partition i.

ix) Average all the empirical likelihood estimates corresponding to all the partitions to find the overall empirical likelihood estimate for the mean.

Algorithm 3

No Parallel Computing:

As mentioned earlier in this chapter, the primary reason for computational inefficiency is due to a large number of points in the grid. Furthermore, for computers that cannot parallel compute, it is important to develop a method to improve computational efficiency. To compensate for the lack of partitioning, I increase the value of the parameters in this algorithm. I initially sample 1000 points and test at 401-point intervals.

i) Sample 1000 points from dataset, and denote this set by X.

ii) Test at intervals $\frac{max(X)-min(X)}{400}$ from min(X) to max(X).

iii) Pick 20 such points which yield the smallest negative log likelihood ratios. Call this set Y.

iv) Separate Y into 401 points at intervals of $\frac{max(Y)-min(Y)}{400}$. Call this set Z.

v)Test X at every point of Z.

vi) The value of Z which provides the lowest negative log likelihood is the empirical likelihood estimate of the dataset.

Comparison of the Algorithms

These three algorithms are applied to 100 replicates of sample size 50000, generated from a chi square distribution with one degree of freedom. I find the mean estimates using 15 partitions in Algorithms 1 and 2, and the results are presented in Figure **6-1**. I also compare how much time is taken to run these three different algorithms on one 10,000,000-point set generated from the chi square distribution with one degree of freedom, as compared to normally computing the empirical likelihood using the default method. These time comparisons are presented in Table **1-1** (Algorithms 1 and 2 use 15 partitions). I especially choose one as the degree of freedom to aid in the depiction of how the three methods compare to the default empirical likelihood method. This is because chi-square distribution with only one degree of freedom has

less variance than a chi square distribution with any other degrees of freedom greater than one. If I were to choose any degree of freedom other than one, the data generated would have greater variance, and hence, the grid of points which need to be tested at a given interval increases, because the range of the dataset will likely increase with greater variance. This will slow down the default empirical likelihood method even more. So, the case I portray is the best-case scenario for the default empirical likelihood method when generating data from a chi-square distribution. In other words, the times I show for the three algorithms are the worst they could be when compared to the default empirical likelihood estimating method for the mean.

Figure **6-1** shows that Algorithm 1 yields mean estimates which have more variability than the mean estimates using Algorithm 2. Algorithm 3 does worse than the other two algorithms, as there is more variability in the mean estimates. In Table **1-1**, I see that Algorithm 1 takes the least time, followed by Algorithms 2 and 3. In contrast, the default empirical likelihood estimate never converges.

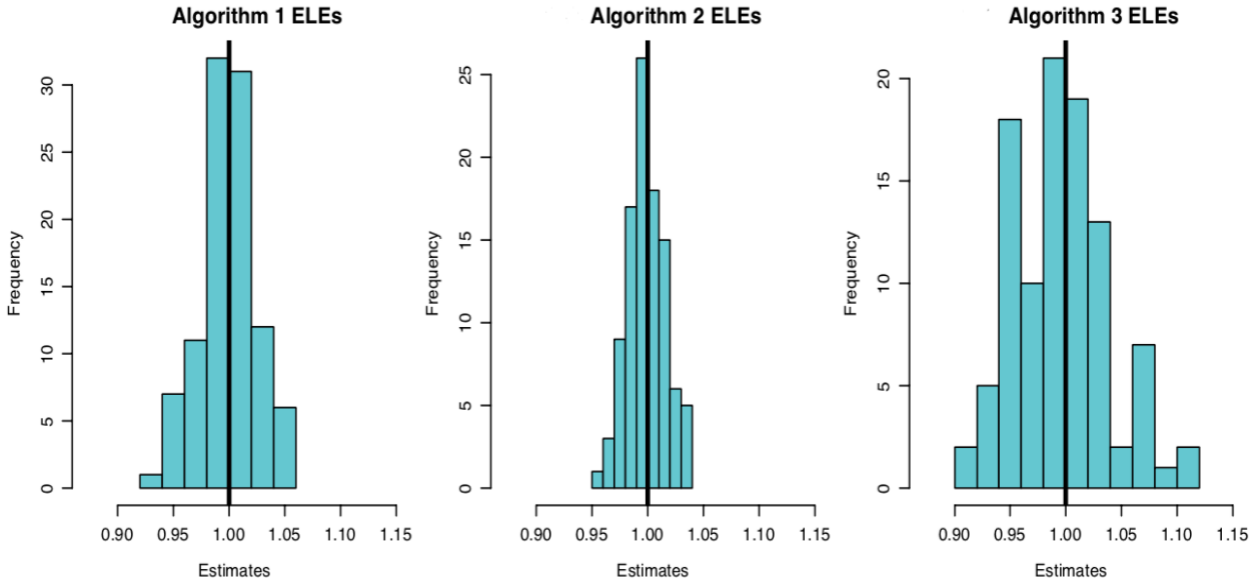

Figure **6-1**: Comparisons on a χ_1^2 Distribution

The results from Figure **6-1** and Table **1-1** intuitively make sense. In Algorithm 3, no parallel computing is used. Hence the estimates are more imprecise, because fewer data points are used in the computation of the empirical likelihood estimates due to the lack of partitioning in Algorithm 3. Furthermore, to account for the lack of partitioning, I sampled more points at the start of the algorithm, so

the computational times were slower as well, as can be seen in Table **1-1**. Algorithm 2 is more precise than Algorithm 1. This is because in Algorithm 1, the first part of the two-part approach is done on 500 points chosen from the entire dataset. In contrast, 500 points are randomly picked in each partition in Algorithm 2, and the first part of the two-part approach is applied to those 500 points. Hence, Algorithm 2 is tested 15*500=7500 points in the first part of the two-part approach, when Algorithm 1 only tests 500 points. This explains why Algorithm 2 is more precise, as can be seen from Figure **6-1**. However, each core is stressed more, as the 500 points are tested in the first part of the two-part approach in each partition in Algorithm 2. This explains why Algorithms 2 requires more time for convergence. In Algorithm 3, only 1000 points are sampled from the entire dataset. In the other two algorithms, an equal number of points corresponding to each partition is sampled. Hence, the number of data points increases proportionally to the number of cores in Algorithms 1 and 2. As a result, the estimates of Algorithm 3 are more imprecise when compared to the other two algorithms, where 16 cores have been used. Algorithm 3 relies on one core to test 1000 points; in contrast, although more points are sampled in Algorithms 1 and 2, each core handles 200 points in Algorithm 1 and 500 points in Algorithm 2 after the partitioning. Hence, this larger number of computations required of a single core leads to a slower convergence time for Algorithm 3, as presented in Table **1-1**.

Table **1-1**: Speed Comparisons Using a 16-Core Computer

Method/Data	10,000,000 points from χ_1^2
Default empirical likelihood method to compute means tested at intervals of 0.1	>5 min Never Finished
Algorithm 1 with 15 partitions	1.1s
Algorithm 2 with 15 partitions	2.1s
Algorithm 3	2.7s

Chapter 4

Discussion

When comparing the methods in described in Chapter 3, and also from Figures **2-1**, **3-1**, and **4-1**, I notice that there is a trade-off between precision and computational efficiency. This point is more evident when comparing Algorithms 1 and 2 using Figure **6-1** and Table **1-1**. Using the algorithms definitely improves computational efficiency, but the methods of Chapter 2 and the algorithms of Chapter 3 can also adversely affect the precision of the estimates. Regardless, these methods present a much better alternative than having to use the default empirical likelihood method to test at every point in a very fine grid.

Figure **4-1** highlights the need for the algorithms I present. One seemingly feasible alternative to any of the algorithms I present would be to simply draw a histogram of the data and to only test a grid of points within a small interval where one may suspect the mean lies. However, Figure **4-1** shows that using this method may still provide biased estimates, depending on what grid interval is used. Figure **4-1** shows the empirical likelihood estimates of the mean of an exponential(3) distribution, where a grid of equidistant points ranging from the minimum to the maximum of the dataset are tested at intervals of 0.5, 0.1, 0.05, and 0.01 respectively. For intervals of 0.1 and 0.05, the estimates never fall on or near the true mean and are clustered on either side of the true mean of 1/3. A large majority of the estimates falls on one side of the true mean, indicating that the empirical likelihood may be biased. This happens because the support of the exponential(3) distribution is $[0, \infty)$, and the majority of the probability mass of the distribution lies near 0.

In Figure **4-1**, I generate 1000 points from exponential(3), where the true mean is 1/3. The probability that a point lies in (0,0.001) is about 0.003. So, I can expect there to be about 3 points in the interval from (0,0.001) among the 1000 points. So, if I test at intervals of much greater than 0.001, from the minimum to the maximum at regular steps, the points I will test at across different samples will be very similar. This is because most of the minima in the samples are all within plus or minus 0.001 of each other. Hence, based on the size of the grid interval, the estimates may be biased. For example, for interval 0.05,

there is a cluster of mean estimates which are anywhere from 0.35 to 0.351, and a smaller cluster is around 0.3 to 0.301. This is because the minimum is almost always between 0 to 0.001, and I usually test once anywhere from (0.3, 0.301) and then again anywhere from (0.35, 0.351) when testing at intervals of 0.05. Since the true mean of 1/3 is closer to (0.35,0.351), there is a bigger cluster in that range.

The exponential(3) distribution provides an example of a case where simply testing a limited grid of values in an interval where one may expect the true parameter to lie will not be useful. Depending on the size of grid interval, the estimate may still be biased. I find in Figure **4-1** that if the true distribution of a dataset has a support which is a proper subset of R, and the majority of the probability mass lies on an end bounded by a finite point, as in exponential(3) distribution, then testing at equidistant points over a limited range may still yield biased estimates, depending on what grid interval is used.

Limitations

The algorithms presented suffer from the limitation that more time is required to improve precision. However, there are certain methods to improve precision without vastly affecting the time needed to finish computations. These methods involve modifying the aforementioned algorithms to a certain extent. To improve precision, I suggest updating the parameters in my algorithms. For example, testing at 400 points at each step of the two-step process in Algorithm 2 will provide more precise estimates, albeit taking more time to converge. I also suggest incorporating three or four-part approaches to narrow down the grid of points which are tested in a multi-modal distribution with a major mode. In a multi-modal distribution with a primary mode, using the two-part approach may not narrow down the range of the grid at all. Narrowing down the range of the grid by testing 201 points and narrowing down to 20 points multiple times may be beneficial in these instances. Finally, the number of partitions may be increased to find more precise estimates. Increasing the number of partitions may require more processing power in a computer, but this will allow more points to be tested simultaneously and help make computations more precise.

Conclusion

I find that partitioning and the two-step testing approach are both necessary for a fast estimation of the mean. Neither of the problems in Chapter 1 can be properly addressed without addressing the other problem. Algorithm 3 shows that only using the two-part approach without partitioning adversely affects both precision and convergence time. On the other hand, the example in Chapter 1 shows that a large dataset is likely to have some extreme values. When partitioning and subsampling, one such extreme value may be included in a grid, hence necessitating testing at a very large number of values, in effect slowing down computation time. Hence, both parallel computing and the two-part (or more refined) approach need to be used to find estimates of the mean efficiently. Overall, I find that the methods I suggest and the algorithms I develop help in mitigating the computational issues encountered in the estimation of univariate means using empirical likelihood. My thesis considered the estimation of means in the univariate setting. However, the same approach can be extended to the estimation of other statistical functionals as well. In those cases, the relevant statistical functional needs to be estimated using empirical likelihood in each partition, and then the corresponding estimates can be averaged to find an estimate for the entire dataset. Subsampling, partitioning, and the two-part approach can be used in the same manner as they are used in the estimation of means. Similarly, the extension from the univariate case to the multivariate case should also be straightforward. The ideas developed in this thesis can be easily adapted to the multivariate setting, where instead of the scalar operations, vector operations will need to be used. Again, the use of subsampling, partitioning, and the two-part approach will remain the same in the multivariate setting.

Bibliography

Efron, Bradley, and Robert J. Tibshirani. *An Introduction to the Bootstrap.* CRC Press, 1994.

Fisher, Ronald A. "On the mathematical foundations of theoretical statistics." *Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character* 222.594-604 (1922): 309-368.

Hald, Anders. "On the history of maximum likelihood in relation to inverse probability and least squares." *Statistical Science* 14.2 (1999): 214-222.

Jaeger, Adam Paul. *Composite Empirical Likelihood: A Derivation of Multiple Non-parametric Likelihoods*. Diss. University of Georgia, 2015.

Lazar, Nicole A. "A Review of Empirical Likelihood." *Annual Review of Statistics and its Application* 8 (2021): 329-344.

Lindsay, Bruce G. "Composite likelihood methods." *Contemporary Mathematics* 80.1 (1988): 221-239.

Owen, Art B. "Empirical likelihood ratio confidence intervals for a single functional." *Biometrika* 75.2 (1988): 237-249.

Owen, Art. "Empirical likelihood ratio confidence regions." *The Annals of Statistics* 18.1 (1990): 90-120.

Qin, Jin, and Jerry Lawless. "Empirical likelihood and general estimating equations." *The Annals of Statistics* (1994): 300-325.

Wang, Chun, Ming-Hui Chen, Elizabeth Schifano, Jing Wu, and Jun Yan. “Statistical methods and computing for big data.” *Statistics and Its Interface* 9.4 (2016): 399-414.

Appendix

Code

```
#R code for the three algorithms
library("emplik")
library("ExtDist")
library("tictoc")
library("doParallel")
############################
############################
##Algorithm 1
number=100
empirical_likelihood_estimates=integer(number)
maximum_likelihood_estimates=integer(number)
#Below is the code for empirical likelihood estimate for each sample
for (i in 1:number) {
  tic()
  #n is the number of points in each sample
  n=50000
  original=rchisq(n, 1) #the distribution generated from

  x=sample(original, 500, replace = FALSE)
  originaltestinterval=(max(x)-min(x))/200

  #y will give the neg log likelihood ratios
  y=rep(0,ceiling((1/originaltestinterval)*(max(x)-min(x)) + 1))
  y[1]=el.test(x,min(x))$'-2LLR'
  for (t in 1:(length(y)-1)) {
    y[t+1]=el.test(x,min(x)+(t*originaltestinterval))$'-2LLR' }

  l=order(y)
  newsetoftwenty=rep(0,20)
  for (z in 1:20) {
    newsetoftwenty[z]=min(x)+originaltestinterval*(l[z]-1)
  }

  #lists the set of 101 new testing points
  newtestingpoints=seq(from=min(newsetoftwenty), to=max(newsetoftwenty),
by=((max(newsetoftwenty)-min(newsetoftwenty))/200))

  #k= number of partitions of the sample=number of cores
  k=15

  number_in_each_partition=n/k
  #parallelization and next part of testing
  registerDoParallel(k)
  emplikest<-foreach (a=1:k, .combine = c) %dopar% {
```

```
   newx=sample(original[(1+(number_in_each_partition*(a-1))):(number_in_each_partition*a)], 200,
replace = FALSE)

   #negloglik is a vector that consists of negative log likelihoods associated with testing certain values
   negloglik=rep(0,length(newtestingpoints))
   for (p in 1:length(newtestingpoints)) {
     testpoint=newtestingpoints[p]
     negloglik[p]=el.test(newx, testpoint)$'-2LLR'
   }
  newtestingpoints[which(negloglik==min(negloglik))]
   }
  #overall empirical likelihood of the sample found by averaging all empirical likelihood estimates from
all the partitions
  empirical_likelihood_estimates[i]=mean(emplikest)
  maximum_likelihood_estimates[i]=mean(original)
  toc()
  cat("We are on sample ", i,"      ")
}

############################
############################
##Algorithm 2
number=100
empirical_likelihood_estimates=integer(number)
maximum_likelihood_estimates=integer(number)
#Below is the code for empirical likelihood estimate for each sample
for (i in 1:number) {
  tic()
#n is the number of points in each sample
n=50000
original=rchisq(n, 1) #the distribution generated from
#k= number of partitions of the sample
k=15

number_in_each_partition=n/k

registerDoParallel(k)  # use multicore, set to the number of our cores
emplikest<-foreach (a=1:k, .combine = c) %dopar% {

   x=sample(original[(1+(number_in_each_partition*(a-1))):(number_in_each_partition*a)],500,replace =
FALSE)
   #y is a vector that consists of negative log likelihoods associated with testing certain values

   #The values tested at go from the minimum value in the data to the largest value in increments of the
"testinterval" below
   testinterval=(max(x)-min(x))/200

   y=rep(0,ceiling((1/testinterval)*(max(x)-min(x)) + 1))
   y[1]=el.test(x,min(x))$'-2LLR'

        for (t in 1:(length(y)-1)) {
```

```
        y[t+1]=el.test(x,min(x)+(t*testinterval))$'-2LLR'}

l=order(y)
newtestingpoints=rep(0,20)
for (t in 1:20) {
  newtestingpoints[t]=min(x)+testinterval*(l[t]-1)
}


newtestingpoints=seq(from=min(newtestingpoints), to=max(newtestingpoints),
by=((max(newtestingpoints)-min(newtestingpoints))/200))

negloglik=rep(0,length(newtestingpoints))
for (p in 1:length(newtestingpoints)) {
  testpoint=newtestingpoints[p]
  negloglik[p]=el.test(x, testpoint)$'-2LLR'
}

newtestingpoints[which(negloglik==min(negloglik))]
  }

#overall empirical likelihood of the sample found by averaging all empirical likelihood estimates from all
the partitions
empirical_likelihood_estimates[i]=mean(emplikest)
maximum_likelihood_estimates[i]=mean(original)
toc()
cat("We are on sample ", i,"      ")
}


############################
############################
##Algorithm 3

number=100
empirical_likelihood_estimates=integer(number)
maximum_likelihood_estimates=integer(number)
#Below is the code for empirical likelihood estimate for each sample: very similar to Algorithms 1 and 2
for (i in 1:number) {
  tic()
  #n is the number of points in each sample
  n=50000
  original=rchisq(n, 1) #the distribution generated from

    x=sample(original,1000,replace = FALSE)
    #y is a vector that consists of negative log likelihoods associated with testing certain values

    #The values tested at go from the minimum value in the data to the largest value in increments of the
"testinterval" below
    testinterval=(max(x)-min(x))/400
```

```
    y=rep(0,ceiling((1/testinterval)*(max(x)-min(x)) + 1))
    y[1]=el.test(x,min(x))$'-2LLR'

    for (t in 1:(length(y)-1)) {
      y[t+1]=el.test(x,min(x)+(t*testinterval))$'-2LLR'}

    l=order(y)
    newtestingpoints=rep(0,20)
    for (t in 1:20) {
      newtestingpoints[t]=min(x)+testinterval*(l[t]-1)
    }


    newtestingpoints=seq(from=min(newtestingpoints), to=max(newtestingpoints),
by=((max(newtestingpoints)-min(newtestingpoints))/400))

    negloglik=rep(0,length(newtestingpoints))
    for (p in 1:length(newtestingpoints)) {
      testpoint=newtestingpoints[p]
      negloglik[p]=el.test(x, testpoint)$'-2LLR'
    }
  #Note that there are no partitions here
  empirical_likelihood_estimates[i]=newtestingpoints[which(negloglik==min(negloglik))]
  maximum_likelihood_estimates[i]=mean(original)
  toc()
  cat("We are on sample ", i,"      ")
}
```