A Fast and Low-Storage Algorithm for Finding Quantiles

by

Lap-Cheung Lau and Dennis D. Boos
Department of Statistics, North Carolina State University
Raleigh, NC 27695-8203

Institute of Statistics Mimeo Series No. 2268

September 1994

A Fast and Low-Storage Algorithm for Finding Quantiles

Lap-Cheung Lau and Dennis D. Boos
Department of Statistics, North Carolina State University
Raleigh, NC 27695-8203

September 1994

Abstract

When a data set is too large to be stored in a computer's primary memory, direct sorting methods for finding the sample quantiles cannot be used. In this paper, a memory-saving method is proposed that can reduce the number of comparison operations so that CPU time is dramatically reduced. Moreover, in contrast to other memory-saving methods which give approximations to the sample quantiles, this one produces the exact sample quantile.

Key Words and Phrases: Critical value; confidence interval; quantile; sorting.

1 Introduction

In many cases, the sample quantiles of a large data set are desired. For example, when investigating the power of a test statistic without any clear knowledge of its distribution, the first step of a Monte Carlo study is often to generate a large sample of such test statistics from the null population. The $(1-\alpha)$th sample quantile is then used as the critical value for an α level test in the rest of the study. In nonparametric statistics, sample quantiles of large sets of data are often required for standard inference. For example, the Hodges-Lehmann estimator for the difference in the locations of two populations is the median of all pairwise differences (see, for example, Randles and Wolfe, 1979, p.208-209). For samples of size n_1 and n_2, there will be $O(n_1 n_2)$ such differences. When sample sizes are large, ordinary quick sorting methods for computing the Hodges-Lehmann estimator may require a prohibitive amount of storage. In this particular situation, there are fast sorting algorithms available(see Monahan, 1984). Rousseeuw and Bassett (1990) give other applications where quantiles of large data sets are required, ranging from ERG curves in opthamology to crystallography.

It is thus quite common to face extremely large data sets. However, in any computing environment, the number of observations that can be stored in primary memory is finite. Moreover, many computing environments also have limits on the maximum array size, and this may be much smaller and even independent of the availability of the memory. For example, in Common Lisp (Steele, 1990), the maximum array size is implementation dependent. In one particular Lisp implementation, Macintosh Common Lisp (Apple Computer, 1992), the maximum array size is just over two million. Even that array size will not be sufficient for some applications which have more than several gigabytes of data.

In this report we are concerned with low-storage quantile estimation and the efficient use of CPU time. More specifically, suppose that $x_1, ..., x_n$ are independent observations from a continuous distribution function F with density function f. The number ξ_p such that $F(\xi_p) = p$ is called the pth quantile. We wish to estimate ξ_p with relatively low storage and low CPU time.

The sample quantile $\hat{\xi}_p$ is a solution of $F_n(t) = p$, where F_n is the empirical distribution function. It is the simplest and most intuitive estimator for the population quantile and has the smallest asymptotic variance among translation invariant estimators (Pfanzagl, 1974). It is consistent, and if $f(\xi_p) > 0$, then $\hat{\xi}_p$ has the following asymptotic distribution:

$$\sqrt{n}(\hat{\xi}_p - \xi_p) \longrightarrow N\left(0, \frac{p(1-p)}{f^2(\xi_p)}\right)$$

in distribution as $n \longrightarrow \infty$.

Many methods have been proposed for estimating ξ_p from a large data set. Pearl (1981) suggested a minimax tree for estimating an arbitrary quantile. Weide (1978), Tukey (1978), Pearl (1981), and Rousseeuw and Bassett (1990) independently proposed another tree-based method for median estimation, by using recursive medians of subsamples. Following Rousseeuw and Bassett (1990), we call it the remedian method. The method given by Tierney (1983) is based on stochastic approximation, using algebraic operations to update the quantile estimate as the data values are read. Hurley and Modarres (1994) proposed another method based on the sample histogram. We will introduce all these methods in Section 2.

Our method of quantile estimation is based on a probabilistic algorithm introduced by Dunn (1991). His algorithm dramatically reduced the required amount of storage from $O(n)$ to $O(n^{1/2})$, and is much faster than ordinary quick sorting methods. However, it needs a position shift in the storage whenever a new observation falls into the stored confidence set. In Section 3, we propose an improved probabilistic algorithm which still needs only $O(n^{1/2})$ of storage, but does not require a position shift as often. We then demonstrate the substantial gains possible with our algorithm in a small simulation study in Section 4.

2 Methods for Quantile Estimation

2.1 Stochastic Approximation

Tierney (1983) introduced the following stochastic approximation estimator for the pth population quantile ξ_p. A preliminary estimate $\hat{\xi}_{p,m}$ is obtained by using the first m observations. Then as the next observations are read in, the estimate is updated by the following formula:

$$\hat{\xi}_{p,i+1} = \hat{\xi}_{p,i} - d_i[I(x_{i+1} \leq \hat{\xi}_{p,i}) - p],$$

$i = m+1, ..., n$, where $I()$ is the indicator function, and the d_i are chosen so that the final estimator has minimum asymptotic properties. Tierney (1983) has shown that it has the same large sample behaviour as the sample quantile.

At first, the formula above looks very simple since it requires only memory for the first m observations. In fact, however, the values d_i involve an estimate of the density at the population quantile which requires intense computations.

2.2 Minimax Trees

Pearl (1981) proposed an estimator for quantiles based on minimax trees. A minimax tree is a uniform d-ary tree whose terminal nodes are independent observations from F, and whose non-terminal nodes at odd levels of the tree are the minimum values of its d children while even level nodes are the maximum values of its d children. (By convention, the root is at level 0.) The root node is our estimate for a specific quantile of the distribution function F, where the quantile estimated is related to d. This estimator converges in probability to the specific quantile for strictly increasing F, but the asymptotic distribution is non-normal (Pearl, 1981). The breakdown point is $n^{-1/2}$ (Hurley and Modarres, 1994).

2.3 The Remedian

This method appeared in Weide (1978), Tukey (1978), Pearl (1981), and Rousseeuw and Bassett (1990). Its mechanism is basically the same as the Minimax Tree. However, instead of searching the minimum or the maximum at the alternative levels of the tree, it always finds the median at each level.

Let k be the number of the levels in the tree. For any fixed d, the number of children for each node, the estimator, which is the root of the tree, is consistent for the median, converging to a non-normal distribution as $k \rightarrow \infty$.

For $d \geq 3$, the remedian has breakdown point $\epsilon_{remedian} = ([d/2]/d)^k \geq 1/2^k$, where $[x]$ is the smallest integer greater than or equal to x. Therefore, a little algebra reveals that

$$\epsilon_{remedian} \geq n^{-1/2} \geq \epsilon_{minimax}.$$

When $d = 2$, the remedian is the same as the sample mean. In that case, the breakdown point is $\epsilon_{remedian} = 1/n < \epsilon_{minimax}$.

2.4 Histogram Quantile Estimates

Hurley and Modarres (1994) proposed a method based on sample histograms. Suppose that l and r are fixed and known numbers which contain the specific sample quantile with a very high confidence. Divide $(l,r]$ into m bins, each of width w/m where $w = l - r$. Then count the number falling in each bin $(l + iw/m, l + (i+1)w/m]$, $i = 0, 1, ..., m-1$, and the number falling at or below l and beyond r. The histogram estimator of ξ_p is

$$\hat{\xi}_p = a\frac{F_n(b) - 1/2}{F_n(b) - F_n(a)} + b\frac{1/2 - F_n(a)}{F_n(b) - F_n(a)},$$

where $F_n(x)$ is the empirical distribution function and the bin $(a, b]$ contains the sample quantile so that

$$F_n(a) < p \leq F_n(b).$$

Note that the estimator $\hat{\xi}_p$ is simply obtained by linear interpolation of bin end points.

It can be seen that

$$\hat{\xi}_p = \hat{\xi}_{p,s} + O(1/m).$$

Hence, $\hat{\xi}_p$ has the same asymptotic distribution as the sample median when $n = o(m^2)$. However, it is not necessarily a consistent estimator of ξ_p if m is of smaller order.

3 An Algorithm (P-Algorithm) for Finding the Exact Sample Quantile

The methods described in Section 2 are commonly suggested methods to estimate population quantiles in large samples. However, either their large sample behaviour is not as good as the sample quantile, or they need intense computations (see Hurley and Modarres, 1994). If ordinary sorting methods are used to find the sample quantile, limited storage might be a problem when the data set is very large.

Dunn (1991) introduced a probabilistic method to locate the sample quantile which dramatically reduced the required storage. However, his method needs a position shift in the storage

whenever the new read-in observation falls in the confidence set. Here, we propose an improved probabilistic algorithm (P-algorithm say) which will reduce the number of position shifts by imposing a two-dimensional array.

Suppose that we wish to find the pth sample quantile $\hat{\xi}_p$ from a data set, $x_1, x_2, ..., x_n$. Our P-algorithm is as follows:

1. Randomly select m observations, $x_1, x_2, ..., x_m$, and find an approximate $100(1-\alpha)\%$ confidence interval $[x_{(l)}, x_{(u)})$ for ξ_p where l and u (Juritz, Juritz and Stephens, 1983) are obtained by

$$l = \left[mp - z_{\alpha/2}\sqrt{mp(1-p)}\right]$$

and

$$u = \left[mp + z_{\alpha/2}\sqrt{mp(1-p)}\right] + 1;$$

2. Divide the confidence interval into $v = u - l$ subintervals with endpoints $[a_i, b_i)$, $i = 1, 2, .., v$, where $a_i = x_{(i+l-1)}$ and $b_i = x_{(i+l)}$. On each interval, build a stack of length k, where k is obtained by $k = [2z_{\alpha/2}\sqrt{np(1-p)}/v + 1]$, so that it guarantees the total amount of storage in the stacks is greater than the number needed to have a $100(1-\alpha)\%$ confidence set for ξ_p. The stacks are used to store the values of x_i which fall into the corresponding subintervals;

3. Initialize the numbers $n_{below} = l - 1$, $n_{above} = m - v + 1$, and $n_i = 1, i = 1, 2, ..., v$ where n_{below} is the number of x_i less than a_l, n_{above} is the number of x_i greater than or equal to b_v, and n_i are the numbers of x_i falling into the interval $[a_i, b_i)$;

4. The remaining $(n-m)$ observations are read in one by one. If x has value in between a_1 and b_v, find the subinterval, $[a_{i_0}, b_{i_0})$ say, in which it actually falls and store it in the associated stack, and update the number n_{i_0} by one increment;

5. If the associated stack is full, a stack at the side, either the subinterval $[a_1, b_1)$ or the subinterval $[a_v, b_v)$, will be removed, according to the following criteria: if $n_{below} + n_1 \leq [n_{current}p - z_{\alpha/2}\sqrt{n_{current}p(1-p)}]$, where $n_{current}$ is the number of observations already read in, then remove the first subinterval; otherwise, the last subinterval will be thrown away. After that, the saturated stack will be split into two stacks, and the index of the

subintervals and the associated stacks will be shifted according to which side stack was removed. For example, if the i_0th stack is full and the first stack is removed, then move the data in the second stack to the first, those in the third to the second, and so on until those in the (i_0-1)th stack has been moved to the the (i_0-2)th one, update $n_i = n_{i+1}$, $i = 1, 2, ..., (i_0-2)$, sort the observations in the i_0th stack, place the smallest $[(k+1)/2]$ of them to the (i_0-1)th stack and leave the remaining in the i_0th stack, and finally, update $n_{i_0-1} = [(k+1)/2]$ and $n_{i_0} = (k+1) - [(k+1)/2]$;

6. Continue the steps (4) and (5) until all n observations are read in;

7. The desired sample quantile $\hat{\xi}_p = x_{[np]}$ can be found by sorting the observations in the jth stack where j satisfies

$$n_{below} + \sum_{i=1}^{j-1} n_i < [np] \leq n_{below} + \sum_{i=1}^{j} n_i, \tag{1}$$

and $\hat{\xi}_p = x_{j,(b)}$ where $b = [np] - n_{below} + \sum_{i=1}^{j-1} n_i$ and $x_{j,(b)}$ is the bth smallest observations in the jth stack. If there is no j satisfying the equation (1), a quick complete sorting algorithm, such as shell sort, will be used.

Note that when $k = 1$, this basically reduces to Dunn's (1991) algorithm. But Dunn's algorithm has to perform the position shift whenever a new read-in observation falls in the confidence set, and our algorithm will do it only when both a new read-in observation falls in the confidence set and the associated stack is full. This case does not happen very often. In the simulation described in the next section, our algorithm needs the shifting mechanism about 100 times for finding the .95th sample quantile from a data set of size 500,000, while Dunn's algorithm needs about 7,000.

4 Simulation Study

In this section, we will report a simple simulation result. All simulations were run on the SUN SPARC station 5, and SAS 6.09 was used for SAS runs.

First, we generate 10 sets of 500,000 independent observations from a U(0,1) distribution. Then for each set, we use four methods to find the .95th sample quantile: the shell sort method,

Table 1: Comparison of average CPU time (in seconds) for shell sort method, SAS PROC SORT, Dunn's algorithm and P-algorithm.

Methods	Total CPU Time	Read-In Time	Actual Searching Time
Shell Sort	60.09	31.43	28.66
SAS PROC SORT	47.90	22.23	25.67
Dunn's Algorithm	41.91	31.43	10.48
P-algorithm	33.19	31.43	1.76

PROC SORT in SAS, Dunn's algorithm and our P-algorithm. For our P-algorithm, we set our parameters as follows: $\alpha = .001$, $m = 200$ so that $v \approx 30$ and $k \approx 40$. Therefore, $30 \times 40 = 1200$ total units of storage are needed. Obviously, the shell method needs $500,000$ units of storage while PROC SORT in SAS uses approximately two times the $500,000$ units.

In Table 1 we can see the average CPU time needed by each method. Our P-algorithm dominates all other methods. Regardless of the read-in time, the average CPU time used by our P-algorithm is just about one sixth of Dunn's algorithm, one fourteenth of PROC SORT in SAS, and one sixteenth of the shell sort method.

5 Conclusion

Our P-algorithm, unlike the methods described in Section 2, can find exact sample quantiles with very high probability. That probability can be increased by making α smaller although the average computing time would then be increased. Compared to other algorithms for finding the exact sample quantiles, our P-algorithm requires relatively low storage ($O(n^{1/2})$), and saves CPU time (only one sixth of Dunn's efficient algorithm). These improvements will reduce the chance of insufficient memory or lengthy searches.

REFERENCES

Apple Computer Inc. (1992), *Macintosh Common Lisp Reference.*

Dunn, C. L. (1991), "Precise Simulated Percentiles in a Pinch," *The American Statistician*, 45, 207-211.

Hurley, C., and Modarres, R. (1994), "Low-Storage Quantile Estimation," Technical Report.

Juritz, J. M., Juritz, J. W. F., and Stephens, M. A. (1983), "On the Accuracy of Simulated Percentage Points," *Journal of the American Statistical Association*, 83, 441-444.

Monahan, J. F. (1984), "Fast Computation of the Hodges-Lehmann Location Estimator," *ACM Transactions on Mathematical Software*, 10, 265-270.

Pearl, J. (1981), "A Space-Efficient On-Line Method of Computing Quantile Estimates," *Journal of Algorithms*, 2, 164-177.

Pfanzagl, J. (1974), "Investigating the Quantile of an Unknown Distribution," *Contributions to Applied Statistics*, W. J. Ziegler, ed. Birkhauser Verlag, Basel, 111-126.

Randles, R. H., and Wolfe, D. A. (1979), *Introduction to the Theory of Nonparametric Statistics*, John Wiley, New York.

Rousseeuw, P. J., and Bassett, G. W. (1990), "The Remedian: A Robust Averaging Method for Large Datasets," *Journal of the American Statistical Association*, 85, 97-104.

Steel, G. L. (1990), *Common Lisp, The Language*, 2nd ed. Digital Press, Bedford Mass.

Tierney, L. (1983), "A Space-Efficient Recursive Procedure for Estimating a Quantile of an Unknown Distribution," *SIAM Journal on Scientific and Statistical Computing*, 4(4), 706-711.

Tukey, J. W. (1978), "The Ninther: A Technique for Low-Effort Robust (Resistant) Location on Large Samples," *Contributions to Survey Sampling and Applied Statistics in honor of H. O. Hartley*, H. A. David(editor), Academic Press, New York, 251-257.

Weide, B. W. (1978), "Space-Efficient On-Line Selection Algorithm," *Computer Science and Statistics: Proceedings of the 11th Symposium on the Interface*, 308-311.