

T-4436

COMPUTER BASED BOOTSTRAP SIMULATION

ARTHUR LAKES LIBRARY
COLORADO SCHOOL OF MINES
GOLDEN, CO 80401

by

Bradley A. Warner

ProQuest Number: 10783926

All rights reserved

INFORMATION TO ALL USERS

The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if material had to be removed, a note will indicate the deletion.



ProQuest 10783926

Published by ProQuest LLC (2018). Copyright of the Dissertation is held by the Author.

All rights reserved.

This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

T-4436

A thesis submitted to the Faculty and the Board of Trustees of the Colorado School of Mines in partial fulfillment of the requirements for the degree of Master of Science (Mathematical and Computer Sciences).

Golden, Colorado

Date October 22, 1993

Signed: Bradley A. Warner
Bradley A. Warner

Approved: William Astle
William Astle
Thesis Advisor

Golden, Colorado

Date October 22, 1993

Ardel Boes
Dr. Ardel Boes
Professor and Head,
Department of Mathematical
and Computer Sciences

ABSTRACT

This paper discusses the development of a computer program that simulates the bootstrap. The computer based bootstrap is used to understand the power and limitations of the bootstrap.

The bootstrap is a tool that is used to estimate statistical error, such as standard error or confidence intervals. The advantage of the bootstrap is it makes no assumptions about the population distribution. The bootstrap resamples from the original sample to obtain an estimate of the statistic's distribution. Measures of statistical error are obtained from this estimated distribution.

The computer based bootstrap simulation was applied to two common statistics, the sample mean and sample variance. These two statistics have well-known properties and provide excellent reference points for understanding the bootstrap. The results demonstrate the accuracy of the bootstrap depends on the original sample size, the population distribution, and the statistic of interest. These three factors are also interrelated in their effects on the bootstrap.

The power of the bootstrap is demonstrated by applying it to a lesser known statistic, the process control ratio. Methods for obtaining measures of statistical error for the process control ratio are based on strict assumptions about the underlying population distribution. The bootstrap avoided this assumption and performed well.

TABLE OF CONTENTS

	Page
ABSTRACT.....	iii
LIST OF FIGURES.....	vi
LIST OF TABLES.....	vii
ACKNOWLEDGMENTS.....	viii
Chapter 1. INTRODUCTION.....	1
Chapter 2. THE BOOTSTRAP.....	3
2.1 The Bootstrap.....	3
2.2 Example Using Sample Mean.....	4
2.3 Bootstrap Simulation.....	5
2.4 Bootstrap Confidence Intervals.....	7
Chapter 3 COMPUTER PROGRAM.....	13
3.1 Computer Program.....	13
3.2 Balanced Bootstrap.....	19
3.3 How Many Replications?.....	21
3.4 How to Add a New Statistic.....	22
Chapter 4 BOOTSTRAP SIMULATION OF MEAN AND VARIANCE.....	24
4.1 Introduction.....	24
4.2 Sample Mean for a Normal Population.....	25
4.3 Sample Mean for an Exponential Population.....	27
4.4 Sample Variance for a Normal Population.....	31
4.5 Sample Variance for an Exponential Population..	34
4.6 Summary.....	43
Chapter 5 APPLICATION OF THE BOOTSTRAP TO THE PROCESS CONTROL RATIO.....	45
5.1 Introduction.....	45
5.2 Bootstrap Standard Error for C_{pk} (Normal Population).....	46

5.3 Bootstrap Confidence Intervals for C_{pk}	48
5.4 Bootstrap of C_{pk} for Log-Normal Population.....	52
5.5 Summary.....	54
Chapter 6 CONCLUSION.....	55
REFERENCES CITED.....	57
ADDITIONAL READINGS.....	59
APPENDIX COMPUTER SOURCE CODE.....	60
Computer Diskette.....	Pocket

LIST OF FIGURES

	Page
Figure 2.1 Transformed and Standard Normal Distributions	9
Figure 3.1 Program Menus.....	14
Figure 3.2 Main Menu Screen.....	16
Figure 3.3 Histogram from Program.....	18
Figure 4.1 Bootstrap Distribution versus Actual Distribution for Sample Mean (Sample Size of 10 from a Standard Normal Population).....	28
Figure 4.2 Comparison of Exponential Distribution with its Bootstrap Estimate (n=10, variance=1).....	37
Figure 4.3 Bootstrap Results for the Standard Error of Variance (Exponential Distribution n=10, variance=1).....	38
Figure 4.4 Histogram for the Standard Error of the Sample Variance (Sample Size of 10 from Standard Normal Population).....	40
Figure 4.5 Histogram for Standard Error of the Sample Variance (Sample Size of 10 from an Exponential Distribution).....	41
Figure 4.6 Histogram for the Standard Error of the Sample Variance (Sample Size of 100 from an Exponential Population).....	42
Figure 5.1 Histogram for the Bootstrap Standard Error for C_{pk} (Sample Sizes of 25 from a Standard Normal Population).....	47
Figure 5.2 Histogram for Bootstrap Standard Error for C_{pk} (Sample Sizes of 25 from a Standard Normal Population).....	49

LIST OF TABLES

	Page
Table 4.1 Summary Statistics for Test Samples.....	24
Table 4.2 Standard Error for Sample Mean (Random Sample from a Standard Normal Population).....	26
Table 4.3 Bootstrap 95% Confidence Intervals for Mean (Random Sample from a Standard Normal Population).....	27
Table 4.4 Standard Error for Sample Mean (Random Sample from an Exponential Population).....	29
Table 4.5 Bootstrap 95% Confidence Intervals for Mean (Random Sample from an Exponential Population).....	31
Table 4.6 Standard Error for Sample Variance (Random Sample from a Standard Normal Population).....	32
Table 4.7 Bootstrap 95% Confidence Intervals for Variance (Random Sample from a Standard Normal Population).....	33
Table 4.8 Standard Error for Sample Variance (Random Sample from an Exponential Population).....	35
Table 5.1 Summary Statistics for Test Samples.....	50
Table 5.2 95% Lower Confidence Limit for C_{pk} (Standard Normal Population).....	51
Table 5.3 Bootstrap and Simulation Results for C_{pk} from a Log-Normal Population.....	53

ACKNOWLEDGMENTS

I would like to thank Professor William Astle for his invaluable insights and assistance, especially with the proofs. His commitment and dedication was incredible. I would also like to thank Dr. Robin Murphy for her assistance in educating a poor programmer.

CHAPTER 1

INTRODUCTION

Often in scientific and engineering applications statistics help describe and provide information about a process. The sample mean is an example of a statistic commonly used for these purposes. Statistics are only estimates of the respective population parameters and many times the researcher requires a measure of the error in the statistic. This is often done through the use of standard error and confidence intervals. Traditionally, the researcher had to either know the distribution of the statistic of interest, or use a large sample so that the distribution could be approximated. Under these conditions, the researcher could then determine a standard error or confidence interval.

The bootstrap is a tool proposed by Bradley Efron of Stanford University (Efron 1982) that provides a method for obtaining estimates of statistical error without any assumptions about the population. This is a significant advantage over traditional methods of statistical error estimation. For some statistics a closed form solution exists for the bootstrap estimate of statistical error; however, for most statistics this is not the case. When an empirical bootstrap estimate does not exist, a bootstrap estimate can be obtained using a computer based simulation. The balanced bootstrap algorithm (Davison, Hinkley, and Schechtman 1986) is a modified version of the bootstrap used in computer simulations. It is designed to reduce the

simulation error that is inherent in computer simulations. The balanced bootstrap is discussed in detail in section 3.2.

The purpose of the work described in this paper was to develop a computer based simulation of the bootstrap. The simulation could then be used to better understand the power and limitations of the bootstrap.

CHAPTER 2

THE BOOTSTRAP

2.1 The Bootstrap

The bootstrap was introduced as a nonparametric tool for estimation of statistical error, such as bias and standard error (Efron 1982). The idea behind it is straight forward. Consider a sample of size n , denoted by $X_1, X_2, X_3, \dots, X_n$, where the observations are independent and identically distributed. Independence means that any one sample observation does not depend on any of the other sample observations and identically distributed means that the sample observations come from the same distribution, denoted by F . The bootstrap estimates the population distribution F by using the empirical distribution of the sample. Each observation in the sample is assigned a probability of $1/n$ and this discrete probability distribution, denoted as \hat{F} , is used to estimate the original population distribution. This estimate of the population distribution is a nonparametric maximum likelihood estimate (Efron 1982). Suppose $\hat{\theta}(F)$ is the statistic of interest, this notation signifies that the statistic is a function of the underlying population distribution F , the bootstrap estimates $\hat{\theta}$ by substituting \hat{F} for F . The bootstrap is estimating the distribution of $\hat{\theta}$ based on \hat{F} . From the estimated distribution of $\hat{\theta}$, estimates of statistical error such as standard error and confidence intervals are obtained.

2.2 Example Using Sample Mean

As an example, consider finding the standard error of the sample mean. The sample mean is a commonly used descriptive statistic defined as

$$\bar{X} = \sum_{i=1}^n X_i / n \quad , \text{ where } n \text{ is the sample size.}$$

The standard error of the sample mean is

$$se = \sqrt{\sigma^2 / n} \quad ,$$

where σ^2 is the variance of the original population. The variance of the population is often unknown so it is commonly estimated by the sample variance

$$\hat{\sigma}^2 = s^2 = \sum_{i=1}^n (X_i - \bar{X})^2 / (n - 1) \quad .$$

With this definition, the sample variance is an unbiased estimate of the population variance. An estimate of standard error is simply the square root of the sample variance divided by the sample size

$$se_0 = \sqrt{s^2 / n} \quad .$$

The bootstrap estimate of the standard error of the sample mean starts with a sample of size n from \hat{F} , denoted as $X^*_1, X^*_2, \dots, X^*_n$. The sample is drawn by random sampling with replacement from \hat{F} . Therefore the bootstrap estimate of the standard error of the sample mean is

$$(\text{variance } \bar{X}^*)^{1/2} \quad , \text{ where } \bar{X}^* \text{ is the bootstrap sample mean.}$$

The variance of the bootstrap sample mean is defined as

$$\text{var } \bar{X}^* = E\{[\bar{X}^*_i - E(\bar{X}^*)]^2\} \quad ,$$

where $E(X)$ is the expected value of the random variable X .

But

$$\text{var } \bar{X}^* = \text{var}\left(\sum_{i=1}^n X_i^* / n\right) = 1/n^2 \sum_{i=1}^n \text{var}(X_i^*),$$

because the random variables are independent. The variance of the bootstrap random variable is

$$\text{var}(X_i^*) = E\{[X_i^* - E(X_i^*)]^2\}.$$

The expected value of an observation in the bootstrap sample is

$$E(X_i^*) = \sum x_i^* f(x_i^*) = \sum x_i / n = \bar{X} ,$$

because

$$f(x_i^*) = \text{Probability}(X_i^* = x_i) = 1/n$$

and X_i^* can only be the values $x_1, x_2, x_3, \dots, x_n$. Using similar arguments, the variance of the bootstrap random variable reduces to

$$\text{var}(X_i^*) = E[(X_i^* - \bar{X})^2] = (1/n) \sum_{i=1}^n (X_i - \bar{X})^2,$$

so

$$\therefore \text{var}(\bar{X}^*) = (1/n^2) \sum_{i=1}^n (X_i - \bar{X})^2. \quad (2.1)$$

The bootstrap estimate of the standard error is the square root of equation 2.1. Notice that

$$se_{\text{bootstrap}} = \sqrt{(n-1)/n} se_0 ,$$

the bootstrap estimate of the standard error of the sample mean is smaller than se_0 , the common estimate of standard error.

2.3 Bootstrap Simulation

For many statistics it is not possible to write an equation for the bootstrap estimate of statistical error. The advantage of the bootstrap is that its general form can

be implemented using a computer simulation. The basic algorithm for this simulation is:

- 1) Randomly draw a bootstrap sample of size n with replacement from the n original sample values.
- 2) Calculate the point estimate of the statistic from the bootstrap sample.
- 3) Repeat the steps 1) and 2) B times.

This algorithm uses the sample distribution as an estimate of the original population distribution and samples from it to obtain an estimate of the distribution of the statistic, referred to as the bootstrap distribution. The bootstrap distribution will have B data points that do not have to be unique.

To illustrate the bootstrap simulation, consider a sample of fifteen observations from which a point estimate of the population variance has been calculated. An estimate of the standard error of this point estimate is desired. First, obtain a bootstrap sample by randomly selecting fifteen values from the original sample, the original observations can be selected as many times as desired, but each observation must have the same probability of being selected. Next, calculate the variance of this bootstrap sample and save it as the first point in the bootstrap distribution of the sample variance. Keep taking bootstrap samples until there are B points in the bootstrap distribution. To find an estimate of the standard error of the point estimate of the variance, calculate the standard deviation of the bootstrap distribution. As B approaches infinity the simulation estimate approaches the bootstrap estimate. How large should B be? This question will be addressed in the Chapter 3.

2.4 Bootstrap Confidence Intervals

Confidence intervals are a powerful tool for measuring statistical error because they place bounds on an estimate. A confidence interval has a confidence level, a lower confidence limit, and an upper confidence limit. They are usually reported in the following form, a 95% confidence interval estimate of the population mean is $1.234 < \text{mean} < 3.702$. The interpretation of the confidence interval is that with repeated sampling of the same sample size from the population, the confidence intervals calculated for each sample would include the true population mean 95% of the time. As the confidence level increases for a given sample size so does the width of the interval.

To generate confidence intervals, traditionally, the user had to know a relationship between the population parameter and the statistic of interest that would yield a known population distribution. If the relationship yielded a commonly tabulated distribution, such as a normal or chi-squared, then upper and lower values can be found such that the area enclosed in these bounds equal the confidence level. For example, the sample mean, where the sample is drawn from a population with a normal distribution, has a normal distribution. Since the sample mean is normally distributed, we know that

$$(\bar{X} - \mu) / \sigma_{\bar{x}}$$

is distributed as a standard normal, a normal distribution with mean zero and variance one. The $(1-\alpha)100\%$ confidence interval is

$$\bar{X} - z_{(\alpha/2)} \sigma_{\bar{x}} < \text{mean} < \bar{X} + z_{(\alpha/2)} \sigma_{\bar{x}}$$

where $z_{(\alpha)}$ is the 100α percentage point of the standard normal distribution. In situations where the population is not normally distributed, the distribution of the sample mean is asymptotically normal under fairly general conditions (Freund 1992). As the sample size approaches infinity the standardized sample mean usually approaches a standard normal distribution.

A procedure for generating bootstrap confidence intervals has been proposed (Efron 1987) that assumes a monotone increasing transformation exists such that $\phi=g(\theta)$ and $\hat{\phi}=g(\hat{\theta})$. Again, $\hat{\theta}$ is the statistic from the sample that estimates the population parameter θ and $\hat{\phi}$ is the transformation of the statistic. The transformation has the relationship

$$(\hat{\phi}-\phi)/\tau \sim N(-z_0, 1),$$

where τ is the standard error of the transformed statistic. This means that the transformation has a normal distribution with variance one and mean $-z_0$. The $-z_0$ is a bias term. This distribution is displayed in Figure 2.1. To transform this distribution into a standard normal distribution it must be shifted by the bias z_0 . Therefore

$$(\hat{\phi}-\phi)/\tau + z_0 \sim N(0, 1),$$

meaning this quantity is distributed as a standard normal. The confidence interval is found from

$$\text{Prob}[|(\hat{\phi} - \phi) / \tau + z_0| < z_{\alpha/2}] = 1 - \alpha,$$

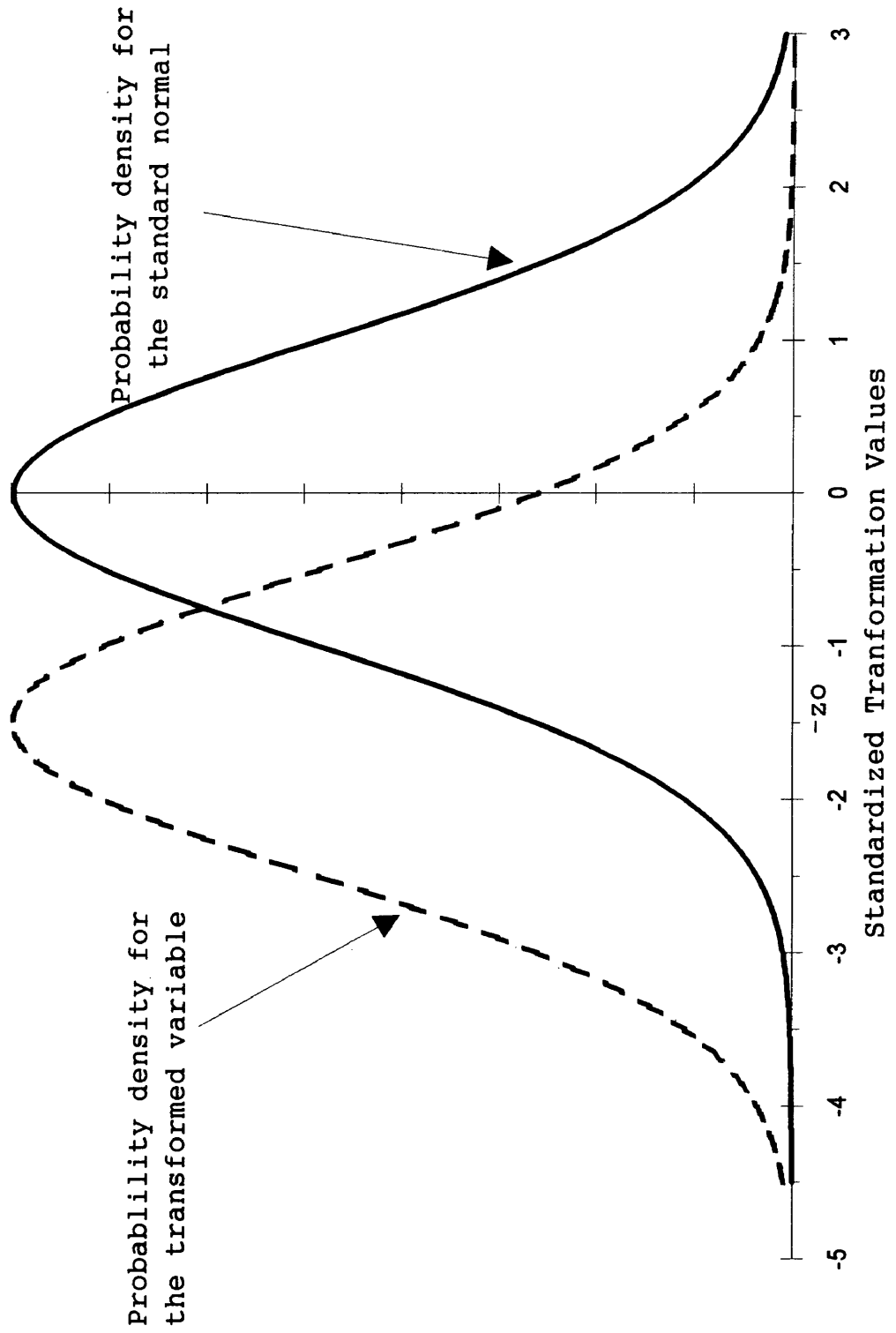
which yields

$$\hat{\phi} + z_0\tau - z_{(\alpha/2)}\tau < \phi < \hat{\phi} + z_0\tau + z_{(\alpha/2)}\tau. \quad (2.2)$$

The confidence interval for θ is obtained by taking the inverse transformation.

The bootstrap provides a way to obtain the confidence intervals without knowing the transformation g . All that is

Figure 2.1
Transformed and Standard Normal Distributions



needed is an estimate of z_0 and the bootstrap distribution of $\hat{\theta}$. The bootstrap estimate of $\hat{\theta}$ is θ^* . The transform of θ^* is $g(\theta^*) = \varphi^*$. Sampling under the bootstrap distribution, the transformation relationship $\varphi^* - \hat{\varphi}$ is an estimate of $\hat{\varphi} - \varphi$. This means that

$$(\varphi^* - \hat{\varphi})/\tau \sim N(-z_0, 1).$$

The probability of φ^* being less than a value d is defined as

$$\text{Prob}_*(\varphi^* < d) = \hat{G}(d),$$

where $\hat{G}(d)$ is the estimate of the cumulative distribution function of the transformed distribution. The notation $\text{Prob}_*(\varphi^* < d)$ means that probability is calculated by adding the number of transformed bootstrap samples less than d and dividing by the total number of bootstrap samples. Likewise

$$\text{Prob}_*(\theta^* < c) = \hat{F}(c),$$

where $\hat{F}(c)$ is the estimate of the cumulative distribution function of the bootstrap distribution. The following relationship holds

$$\hat{G}[g(t)] = \text{Prob}[\varphi^* < g(t)] = \hat{F}(t) = \text{Prob}(\theta^* < t),$$

because g is a monotone increasing function. To estimate z_0 , it is necessary to make use of the fact that the median of the bootstrap distribution is median unbiased (Efron 1982). This means that the expected value of the median of the bootstrap distribution is equal to the parameter that is being estimated. Since the parameter is unknown, $\hat{\varphi}$ is used as an estimate. So

$$\text{Prob}(\varphi^* < \hat{\varphi}) = \hat{G}(\hat{\varphi}) = \hat{G}(g(\hat{\theta})) = \hat{F}(\hat{\theta}) = \Phi(z_0),$$

where Φ is the cumulative distribution of the standard normal distribution. Solving for z_0

$$z_0 = \Phi^{-1}[\hat{F}(\hat{\theta})].$$

This is interpreted as finding the number of values from the bootstrap distribution that are less than the statistic from the original sample and dividing by the total number of bootstrap samples. This estimates the median bias of the bootstrap distribution. Any difference from 0.5 is due to bias and will be accounted for in the confidence interval. Then to find z_0 take the inverse standard normal cumulative density function of the median bias estimate.

To develop a confidence interval from equation 2.2 an expression for $\hat{\phi} + z_0 \tau \pm z_{\alpha/2} \tau$ is needed. It is known that

$$\text{Prob}(\phi^* < \hat{\phi} + z_0 \tau \pm z_{(\alpha/2)} \tau) = \hat{G}(\hat{\phi} + z_0 \tau \pm z_{(\alpha/2)} \tau) \quad (2.3)$$

and

$$(\phi^* - \hat{\phi}) / \tau + z_0 \sim N(0, 1).$$

Rearranging equation 2.3

$$\text{Prob}[(\phi^* - \hat{\phi}) / \tau + z_0 < 2z_0 \pm z_{(\alpha/2)}] = \Phi(2z_0 \pm z_{(\alpha/2)}). \quad (2.4)$$

Combining equations 2.3 and 2.4

$$\hat{\phi} + z_0 \tau \pm z_{(\alpha/2)} \tau = \hat{G}^{-1}\{\Phi[2z_0 \pm z_{(\alpha/2)}]\}.$$

Transforming equation 2.2 back to θ

$$g^{-1}(\hat{\phi} + z_0 \tau \pm z_{(\alpha/2)} \tau) = g^{-1}(\hat{G}^{-1}\{\Phi[2z_0 \pm z_{(\alpha/2)}]\}),$$

which reduces to

$$\hat{F}^{-1}\{\Phi[2z_0 \pm z_{(\alpha/2)}]\}.$$

Therefore the $(1-\alpha)100\%$ confidence interval for θ is

$$\hat{F}^{-1}\{\Phi[2z_0 - z_{(\alpha/2)}]\} < \theta < \hat{F}^{-1}\{\Phi[2z_0 + z_{(\alpha/2)}]\}.$$

Since Φ will return a percentage, \hat{F}^{-1} is found by multiplying this percentage by the number of bootstrap samples, taking the greatest integer value of this result, and then picking that value from the numerically sorted bootstrap distribution.

The method of generating a confidence interval assumes that a monotone increasing function for transforming the distribution exists and that z_0 is a constant. There are

situations where these may not be valid assumptions. Therefore, the user must be careful when using bootstrap confidence intervals. Note also that there was the assumption that the transformations produce a normal distribution. The true requirement is that the transformation produce a symmetric distribution. The normal distribution was selected because it is common and well documented. A monotone decreasing function could also be assumed without affecting the results.

CHAPTER 3

COMPUTER PROGRAM

As previously discussed, the bootstrap estimate of statistical error does not have a simple equation for many statistics. However, the bootstrap lends itself well to a computer based simulation. This chapter discusses the development and operation of a bootstrap computer package.

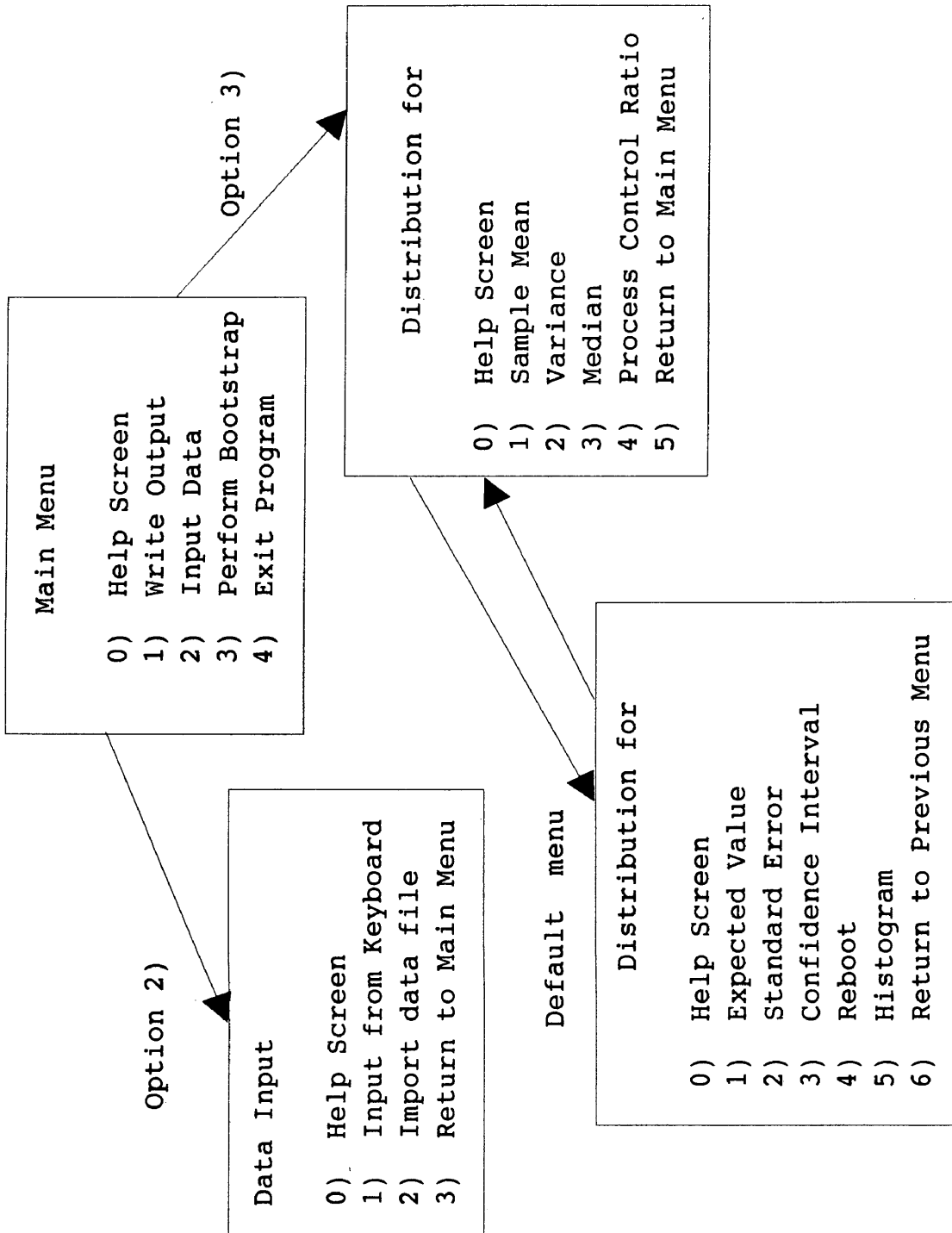
In writing the computer program, there were two primary goals. First, the program had to be easy to use and understand. This was accomplished by the use of a menu driven interface and on-line help system. Second, the program must allow new features to be added without having to re-write much of the existing code. This is important if it is necessary to add a statistic of particular interest. A modular program design incorporating four source files was used to accomplish the goal.

3.1 Computer Program

The computer program was written in C for an IBM compatible personal computer. The compiler was Borland's Turbo C, the DOS version. To execute the program, go to the directory that contains the file boot.exe and type boot. To use the help menus, the four files with the .hlp extension must also be in the directory from which you executed the program.

Figure 3.1 shows the menu hierarchy of the computer program. Notice that a help menu is available at any of the

Figure 3.1
Program Menus



menus. This help system provides a brief description of the options available in each menu.

The main menu allows the user to input data, write results to an output file, or perform a bootstrap. Figure 3.2 is a reproduction of the actual main menu as it would be seen on the screen. The other menus have a similar format. When the program is first executed it skips the main menu and puts the user at the input menu because the program must have data to operate.

The user has two options for data input, direct input from the keyboard or input from a data file. If the data is input from a file, it must be in ASCII format and each data point must be separated by a space or carriage return. The program asks the user for the file name. If the file is in a different directory than the program, the directory must be specified. The user is given the options of viewing the data and of viewing a histogram of the data. Both of these tools allow the user to verify the data. If the data was entered from the keyboard, an option to correct the values is also available.

The write output option of the menu allows the user to output the data to an ASCII file. The program asks for the file name. If the file already exists, the program prompts the user that the old file will be lost if the program is allowed to continue. The user is given the chance to pick a different file name. The program offers the options of writing the input data, the bootstrap data, histogram of the data sets, and a summary, which may include the expected value, standard error, and confidence interval, to the output file.

The bootstrap menu has a choice of statistics. Currently included on the menu are mean, variance, median,

Figure 3.2
Main Menu Screen

BOOTSTRAP SIMULATION

Main Menu

- 0: Help Screen
- 1: Input data
- 2: Write to file
- 3: Perform Bootstrap
- 4: Exit Program

Input your choice: [0...4]:

and process control ratio. The first three were added to explore the bootstrap as will be discussed in Chapter 4; however, median will not be discussed. The program is written in modular code so that any statistic of interest can be added. This will require computer code and a new executable code. This process is discussed in section 3.4.

Once a statistic is selected, the user must enter the number of bootstrap replications. The program generates the bootstrap distribution for the selected statistic. The number of replications determines the number of observations in the bootstrap distribution. The larger the number of replications there are, the better the estimate of the statistic's distribution; but the longer the computational time. A counter at the bottom displays the percentage completion to keep the user informed of the progress of generating the bootstrap distribution.

At the completion of the bootstrap the user is taken into a menu that allows him to explore the bootstrap distribution. This menu estimates parameters of the statistic's distribution from the bootstrap distribution. The expected value is estimated by the mean of the bootstrap replications. Standard error is estimated by the standard deviation of the bootstrap distribution. Confidence intervals are calculated using the bias corrected method discussed in Chapter 2. The reboot procedure calculates the standard error of the standard error of the statistic. For repeated sampling schemes, the reboot provides a method to determine how many replications are necessary. A histogram option allows the user to observe a histogram of the bootstrap distribution. Figure 3.3 is an actual screen display of a representative histogram.

Figure 3.3
Histogram from Program

Frequency	7	30	75	107	102	96	47	26	8	2

Each * equals 3 points.										
108				*						
105				*						
102				*	*					
99				*	*					
96				*	*	*				
93				*	*	*				
90				*	*	*				
87				*	*	*				
84				*	*	*				
81				*	*	*				
78				*	*	*				
75			*	*	*	*				
72			*	*	*	*				
69			*	*	*	*				
66			*	*	*	*				
63			*	*	*	*				
60			*	*	*	*				
57			*	*	*	*				
54			*	*	*	*				
51			*	*	*	*				
48			*	*	*	*	*			
45			*	*	*	*	*			
42			*	*	*	*	*			
39			*	*	*	*	*			
36			*	*	*	*	*			
33			*	*	*	*	*			
30		*	*	*	*	*	*			
27		*	*	*	*	*	*	*		
24		*	*	*	*	*	*	*		
21		*	*	*	*	*	*	*		
18		*	*	*	*	*	*	*		
15		*	*	*	*	*	*	*		
12		*	*	*	*	*	*	*		
9		*	*	*	*	*	*	*	*	
6	*	*	*	*	*	*	*	*	*	
3	*	*	*	*	*	*	*	*	*	*

Interval	0.250	1.250	2.250	3.250	4.250					
Mid-points	0.750	1.750	2.750	3.750	4.750					
The printed values must be multiplied by 10** ⁻¹ .										

3.2 Balanced Bootstrap

The program's main algorithm is the bootstrap. The bootstrap algorithm used in the program is based on a balanced bootstrap (Davison, Hinkley, and Schechtman 1986). Simulating the bootstrap yields simulation error. This occurs because the number of bootstrap replications ideally should be infinite, but simulation uses a finite number of replications. The bootstrap uses an empirical distribution where each of the n sample points has a probability of $1/n$ of being selected. The problem is that for a finite number of bootstrap replications the ratio of the number of times a particular sample appears is not $1/n$. This will lead to small errors due only to simulation. The balanced bootstrap reduces this error by ensuring that each of the original samples is used only in the ratio of $1/n$. This is accomplished by generating a population that has the bootstrap number of replications for each of the original observations. For example, suppose the original sample had 10 observations and the user wanted a bootstrap distribution that has 1000 values, the number of bootstrap replications is 1000. The balanced bootstrap population is generated by placing 1000 copies of each of the original observations in the population. The population has 10,000 values. The bootstrap distribution is generated by random sampling without replacement from the population of 10,000. Therefore each of the original values will appear in the bootstrap distribution $1/n$ times. In the example each of the original 10 sample values appears $1000/10,000$ or $1/10$ times in the bootstrap distribution.

As a simple example of how the simulation error is reduced by using the balanced bootstrap consider the

bootstrap estimate of the bias of the sample mean. Bias is defined as the difference between the expected value of a statistic and the parameter that the statistic is estimating. In notation form the bias for the sample mean is

$$\text{BIAS} = E(\bar{X}) - \mu,$$

but

$$E(\bar{X}) = E\left(\sum_{i=1}^n X_i / n\right) = (1/n)E\left(\sum_{i=1}^n X_i\right) = \mu.$$

The bias of the sample mean is zero. The bootstrap estimate of bias uses the mean of the bootstrap distribution minus the statistic from the original sample

$$\text{BIAS}^* = \bar{\bar{X}}^* - \bar{X}. \quad (3.1)$$

Now

$$\bar{\bar{X}}^* = \sum_{b=1}^B \bar{X}^{*b} / B = \sum_{b=1}^B \left(\sum_{i=1}^n X_i^{*b} / n \right) / B = \sum_{b=1}^B \sum_{i=1}^n X_i^{*b} / (nB).$$

The balanced bootstrap uses each sample B times, so further reducing yields

$$(BX_1)/(nB) + (BX_2)/(nB) + \dots + (BX_n)/(nB)$$

but this is

$$\sum_{i=1}^n (X_i / n) = \bar{X}.$$

Substituting back into equation 3.1,

$$\text{BIAS}^* = \bar{\bar{X}}^* - \bar{X} = \bar{X} - \bar{X} = 0.$$

This agrees with the true bias of the sample mean. If the bootstrap had been used instead of the balanced bootstrap, then the expected value of the bootstrap sample might not have been the sample mean. This occurs because it is unlikely that each sample will be selected B times. So in the case of the bootstrap estimate of the bias of the sample

mean, a non-zero value would have resulted. This bias would be due purely to simulation error.

The actual balanced bootstrap algorithm is modeled after the one proposed by Gleason (1988). Algorithms used in other portions of the program are referenced in the selected bibliography (see Craig and Griffiths and Hill).

3.3 How Many Replications?

When the bootstrap is simulated, one of the primary questions is how many replications are necessary? Remember that the bootstrap simulation provides a measure of statistical error for a statistic. Obviously the greater the number of replications there are, the better the estimate of statistical error. Better in the sense that the estimate will have less variability due to simulation and therefore will be more stable. The cost is computational overhead; a large number of replications may, depending on the computer, take a long time. The reboot option provides a method to determine how many replications are sufficient.

The reboot calculates the standard error of the standard error for a statistic. This gives a measure of the stability of the standard error estimate. The reboot works by generating a standard error estimate using the prescribed number of bootstrap replications. This process is repeated for each of the reboot replications. This is an extremely long procedure. As an example of the use of the reboot procedure, suppose the standard error of a statistic with 500 replications was 0.234 and the reboot indicated that the standard error of the 0.234 estimate was 0.010. The user must make a subjective decision whether this is acceptable. If the user considers this level of variability acceptable,

then 500 replications should be used. Otherwise, more than 500 replications would be required.

Since the reboot procedure is computationally intensive, it is best used when there is repeated use of the bootstrap on similar sample data. For example, suppose a company wants to establish the process capability for a machining process. This will require many samples to be taken over a period of time and process control ratios calculated for each sample. In this case the reboot procedure is used to determine the number of bootstrap replications. If the bootstrap is a one time shot, the user is better off just using a large number of replications.

3.4 How to Add a New Statistic

One of the goals of the program is to have the ability to adapt the program by adding different statistics. This is accomplished through a modular program design. The program is broken into four source files, or modules. The boot.c file has all the math computations including the bootstrap algorithm. The input.c file controls all data input and output as well as the histogram algorithm. The menu.c file contains the structure and displays for the menus and also the help displays. The main menu controls the three other files. It determines which menu to call, when to call for input or output, and when to call the bootstrap.

To illustrate how to add a statistic, consider the situation where we want to get the standard error of the trimmed mean. Trimmed mean is similar to sample mean except a percentage of the sample points from both ends of the ordered sample are ignored. The percentage ignored is

decided by the user. First, trimmed mean must be added to the statistic menu. The easiest method to accomplish this is by replacing an existing statistic. If the statistic is added without replacing an existing menu item, the menus tend to get too long and more changes must be made to the code. For the example, replace process control ratio with trimmed mean. In the menu.c file replace "Process Control Ratio" with "Trimmed Mean." In main.c, changes to the section that controls the program when trimmed mean is selected must be made. This section is the one in the main code that controls the statistic menu. In the example; change the statements asking for specification limits to one asking for the percentage to be trimmed. In boot.c, write a function for calculating the trimmed mean. This would be easy because a sorting algorithm already exists in boot.c. Then in the bootstrap function change the call to C_{pk} to a call to our new trimmed mean function. To finish, compile and link the program. This entire process requires knowledge of the C programming language, copies of the source files, and a Borland Turbo C compiler.

CHAPTER 4
BOOTSTRAP SIMULATION OF MEAN AND VARIANCE

4.1 Introduction

This chapter will explore the bootstrap by using two common and well-understood statistics, the sample mean and sample variance. The effects of sample size, number of bootstrap replications, and population distribution on standard error and confidence intervals of these two statistics are examined. Samples of sizes 10, 100, 1000 from a normal distribution with mean zero and variance 1 and three samples of the same sizes from an exponential distribution with mean and variance 1 are used. A sample size of 10 represents a small sample, 100 is a large sample, and 1000 is essentially an infinite sample. Table 4.1 summarizes the statistics for these samples.

Table 4.1
Summary Statistics for Test Samples

Sample Size	Normal Distribution ($\mu=0, \sigma^2=1$)		Exponential Distribution ($\mu=1, \sigma^2=1$)	
	Sample Mean	Sample Variance	Sample Mean	Sample Variance
10	.020	1.3479	.633	.2777
100	.2181	.7240	1.0291	.6233
1000	.0136	.9795	.9424	.9308

4.2 Sample Mean for a Normal Population

The distribution of the sample mean, when the population has a normal distribution, is normal with mean μ , the population mean, and variance σ^2/n , where σ^2 is the population variance. The standard error of the sample mean is

$$\sigma / \sqrt{n},$$

which for a population variance of 1 is simply the reciprocal of the square root of the sample size. Often the population variance is not known, so it is a common practice to estimate the population variance by the sample variance. Remember that sample variance is defined as

$$s^2 = \sum_{i=1}^n (X_i - \bar{X})^2 / (n - 1).$$

For the bootstrap simulation, replication sizes of 500, 2000, and 10000 were used. Table 4.2 summarizes the results. Notice that the bootstrap estimate of standard error closely matches the commonly used estimate of the square root of sample variance divided by sample size. This occurs because, as discussed in section 2.2, the two estimates are proportional.

A reboot procedure was also performed to obtain the standard error of the standard error. When the sample size was ten and there were 500 bootstrap replications, the standard error of the standard error was .0106. When the replications went up to 2000, the value was halved to .0052. When the sample size increased to 100 and there were 500 bootstrap replications, the standard error of the standard error was .0025. This demonstrates that by increasing the

Table 4.2
Standard Error for Sample Mean
(Random Sample from a Standard Normal Population)

Sample Size	Bootstrap Estimate of Standard Error B = Number of Bootstrap Replications			Common Estimate	Actual Standard Error
	B=500	B=2000	B=10000	s / \sqrt{n}	σ / \sqrt{n}
n=10	.3588	.3585	.3506	.3671	.3162
n=100	.0851	.0844	.0846	.0851	.1
n=1000	.0307	.0308	.0311	.0313	.0316

number of replications, the bootstrap estimate of standard error becomes more stable. The increase in sample size did not reduce the magnitude of the standard error of the standard error. For the sample size of 10, the standard error of the standard error divided by the standard error is $.0106/.3588$ or 3%, and for the sample of size 100, the result is $.0025/.0851$ which is also 3%.

Next bootstrap confidence intervals were generated for the sample mean. The $100(1-\alpha)\%$ confidence interval that is commonly used for the sample mean is

$$\bar{X} - t_{(1-\alpha/2),v} s / \sqrt{n} < \mu < \bar{X} + t_{(1-\alpha/2),v} s / \sqrt{n},$$

where $t_{(1-\alpha/2),v}$ is the $\alpha/2$ percentage point from a t-distribution with v degrees of freedom. This confidence interval is used as a reference for the bootstrap confidence interval. The bootstrap distribution is based on 1000 replications for each of the three sample sizes. Table 4.3

Table 4.3
 Bootstrap 95% Confidence Intervals for Mean
 (Random Sample from a Standard Normal Population)

Sample Size	95% Confidence Interval	Reference Confidence Interval
n=10	$-.5953 < \mu < .7668$	$-.8105 < \mu < .8505$
n=100	$.0525 < \mu < .3841$	$.0493 < \mu < .3869$
n=10000	$-.0469 < \mu < .0733$	$-.0373 < \mu < .0645$

summarizes the results. The bootstrap confidence interval produces results that closely match the one derived from the t-distribution.

The bootstrap performed well for the sample mean from a population with a normal distribution. Even with a small sample size and small number of replications, the bootstrap estimate of the distribution of the sample mean closely matches the actual distribution. This can be seen in Figure 4.1. With an accurate estimate of the distribution of the sample mean, the bootstrap estimates of standard error and confidence intervals will be accurate. This is confirmed by the data.

4.3 Sample Mean for an Exponential Population

The same experiment described in Section 4.2 was repeated for a population with an exponential distribution. The distribution had a mean and a variance of 1. The exponential distribution was selected because it is asymmetrical.

Figure 4.1
Bootstrap Distribution versus Actual Distribution for
Sample Mean
Sample Size of 10 from a Standard Normal Populaiton

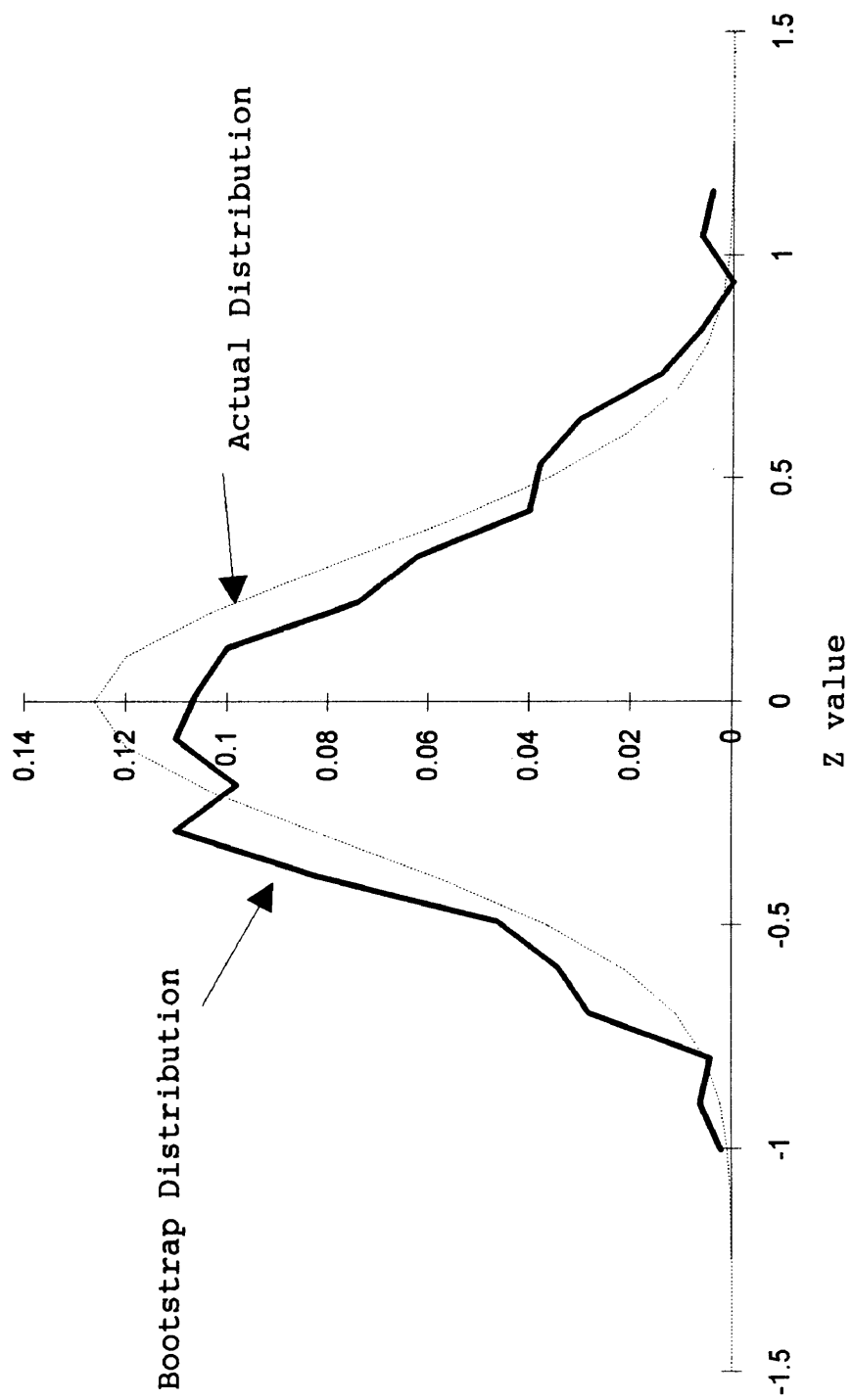


Table 4.4 summarizes the results for the standard error of the sample mean. Two estimates of the standard error are provided. The naive estimate is where the square root of the sample variance is substituted as an estimate of the exponential population parameter β , where β is the mean of the exponential population. This is something someone without a lot of statistical experience might do. The second estimate uses the maximum likelihood (ML) estimate of the exponential population parameter β . Notice that the bootstrap estimate of standard error is closer to the naive estimate of standard error than it is to the actual standard error. This agrees with section 2.2 because the bootstrap standard error of the sample mean is based on sample standard deviation.

For the sample size of 10 the bootstrap estimate of standard error is much smaller than the true standard error.

Table 4.4
Standard Error for Sample Mean
(Random Sample from an Exponential Population)

Sample Size	Bootstrap Estimate of Standard Error B = Number of Bootstrap Replications			Naive Est.	ML Est.	Actual Error
	B=500	B=2000	B=10000	s / \sqrt{n}	\bar{x} / \sqrt{n}	σ / \sqrt{n}
n=10	.1609	.1586	.1574	.1667	.2516	.3162
n=100	.0795	.0781	.0779	.0790	.1014	.1
n=1000	.0278	.0306	.0297	.0305	.0307	.0316

This occurs because the variance of the sample is only .2777 compared with the actual variance of 1. The accuracy of the bootstrap estimate of standard error depends on how close the sample variance matches the population variance because for the sample mean the bootstrap estimate of standard error is the sample standard deviation divided by the square root of the sample size. As the sample size increases the bootstrap estimate of standard error gets closer to the actual standard error, an asymptotic relationship.

Bootstrap confidence intervals were generated for the test cases again using 1000 bootstrap replications. The same reference intervals described in Section 4.2 were used. However, for small sample sizes this confidence interval is not appropriate. The reference confidence interval assumes that the distribution for the sample mean is normal. When the sample size is small, the distribution of the sample mean from an exponential distribution is not normal. Conversely for large samples, greater than 30, the central limit theorem applies and the standardized sample mean is approximately normally distributed. There is no commonly used measure for calculating confidence intervals for the sample mean from an exponential distribution with a small sample size. Therefore, for comparison purposes, the same reference confidence intervals were used for the sample size of 10. Table 4.5 summarizes the confidence intervals. Notice that the bootstrap confidence intervals closely match the reference confidence intervals at the larger sample sizes. At the smaller sample size the bootstrap provides a confidence interval with a smaller width.

ARTHUR LAKES LIBRARY
COLORADO SCHOOL OF MINES
GOLDEN, CO 80401

Table 4.5
 Bootstrap 95% Confidence Intervals for Mean
 (Random Sample from an Exponential Population)

Sample Size	95% Confidence Interval	Reference Confidence Interval
n=10	.3396< μ <.9535	.2560< μ <1.0100
n=100	.8803< μ <1.1865	.8725< μ <1.1857
n=10000	.8856< μ <1.0025	.8826< μ <1.0022

4.4 Sample Variance for a Normal Population

Now the performance of the bootstrap for another common statistic, sample variance, is examined. The variance of the sample variance s^2 is (Efron, 1982)

$$\text{Var}(s^2) = \{\mu_4 - [(n-3)/(n-1)]\mu_2^2\} / n, \quad (4.1)$$

where

$$\mu_k = E\{[X - E(X)]^k\}.$$

For the normal distribution

$$\mu_4 = 3\sigma^4$$

and

$$\mu_2 = \sigma^2 \text{ (Mood, Graybill, and Boes 1974).}$$

So after substituting into equation 4.1 and reducing

$$\text{var}(s^2) = 2\sigma^4 / (n-1).$$

Taking the square root of the variance yields

$$\text{standard error of } s^2 = \sigma^2 \sqrt{2 / (n-1)}. \quad (4.2)$$

Since σ^2 is often not known, s^2 is used as an estimate of the population variance.

Bootstrap estimates of the standard error of the sample variance for the samples from the standard normal distribution were generated with replication size of 500, 2000, and 10000. Table 4.6 summarizes the results. Notice that for the sample size of 10, the bootstrap provides a better estimate of the standard error than is obtained by substituting s^2 into equation 4.2.

Bootstrap confidence intervals were generated using 1000 replications. For comparison, a reference confidence interval is needed. When the population has a normal distribution, then the ratio

$$(n - 1)s^2 / \sigma^2$$

has a chi-squared distribution with $n-1$ degrees of freedom.

Table 4.6
Standard Error for Sample Variance
(Random Sample from a Standard Normal Population)

Sample Size	Bootstrap Estimate of Standard Error B = Number of Bootstrap Replications			Estimate of Standard Error	Actual Standard Error
	B=500	B=2000	B=10000	$s^2 \sqrt{\frac{2}{(n-1)}}$	$\sigma^2 \sqrt{\frac{2}{(n-1)}}$
n=10	.4189	.4368	.4399	.6354	.4714
n=100	.1106	.1123	.1114	.1029	.1421
n=1000	.0402	.0440	.0433	.0438	.0447

So

$$\text{Prob}[\chi_{\alpha/2, n-1}^2 < (n - 1)s^2 / \sigma^2 < \chi_{1-\alpha/2, n-1}^2] = \alpha,$$

where $\chi_{\alpha, n-1}^2$ is the α percentage point from chi-squared distribution with $(n-1)$ degrees of freedom. Thus the $(1-\alpha)100\%$ confidence limit for the population variance is

$$(n - 1)s^2 / \chi_{\alpha/2, n-1}^2 < \sigma^2 < (n - 1)s^2 / \chi_{1-\alpha/2, n-1}^2.$$

For large n , greater than thirty, the chi-squared distribution is approximated by a normal distribution. In fact, if X is a random variable with a chi-square distribution and $(n-1)$ degrees of freedom, then

$$[X - (n - 1)] / \sqrt{2(n - 1)} \sim N(0, 1) \text{ (Fruend 1992).}$$

So

$$\text{Prob}(|\{[(n - 1)s^2 / \sigma^2] - (n - 1)\} / \sqrt{2(n - 1)}| < z_{\alpha/2}) = \alpha$$

and the $(1-\alpha)100\%$ confidence interval is

$$\frac{(n - 1)s^2}{(n - 1) + z_{\alpha/2}\sqrt{2(n - 1)}} < \sigma^2 < \frac{(n - 1)s^2}{(n - 1) - z_{\alpha/2}\sqrt{2(n - 1)}}.$$

Table 4.7 summarizes the results for the confidence intervals. Notice that for the small sample size the

Table 4.7

Bootstrap 95% Confidence Intervals for Variance
(Random Sample from a Standard Normal Population)

Sample Size	95% Confidence Interval	Reference Confidence Interval
n=10	.4926 < σ^2 < 2.2088	.6377 < σ^2 < 4.4931
n=100	.5330 < σ^2 < .9767	.5663 < σ^2 < 1.0036
n=10000	.8991 < σ^2 < 1.0707	.9005 < σ^2 < 1.0737

bootstrap does not simulate the long tail of the chi-square distribution. The reason is that the bootstrap distribution is multinomial. In the case of the sample size of ten, the maximum variance from the bootstrap occurs when five values of the bootstrap sample are from one endpoint of the original sample and the other five are from the other endpoint. For small samples the probability of getting an observation from the tail of the normal distribution is smaller than it is for a larger sample. The maximum bootstrap variance for small samples will often be less than it is for large samples. It is this maximum bootstrap variance that determines the upper tail of the bootstrap distribution. With larger sample sizes there are more observations from the tail of the normal population and thus it is possible to get larger variances. As the sample size increases, the bootstrap confidence interval closely matches the reference interval. However, for this particular example, the shorter width of the bootstrap confidence interval is an advantage because it includes the actual value in smaller limits. Most people would agree that it is preferable to know a value is between 1 and 2 vice 1 and 200, so shorter confidence widths are preferred.

4.5 Sample Variance for an Exponential Population

The sample variance trial was repeated using the same exponential distribution that was used for sample mean.

Using equation 4.1, an expression for the variance of the sample variance from an exponential population will be developed. The exponential distribution is defined as

for $x > 0$ $f(x) = (1 / \beta)e^{-x/\beta}$
 elsewhere $f(x) = 0$.

Now

$$\mu_2 = \beta^2,$$

and

$$\mu_4 = 9\beta^4 \text{ (Mood, Graybill, and Boes 1974).}$$

Substituting into equation 4.1 and reducing,

$$\text{var}(s^2) = \beta^4[(8n - 6) / (n^2 - n)]. \tag{4.3}$$

The standard error of the sample variance is the square root of the right hand side of equation 4.3.

Table 4.8 summarizes the bootstrap estimate of standard error compared to the theoretical standard error and an estimate of the standard error based on the theoretical standard error. Since the parameter β is often not known,

Table 4.8
 Standard Error for Sample Variance
 (Random Sample from an Exponential Population)

Sample Size	Bootstrap Estimate of Standard Error B = Number of Bootstrap Replications			Estimate of Standard Error	Actual Standard Error
	B=500	B=2000	B=10000	$\bar{x}^2 \sqrt{\frac{8n - 6}{(n^2 - n)}}$	$\beta^2 \sqrt{\frac{8n - 6}{(n^2 - n)}}$
n=10	.0867	.0893	.0892	.3633	.9190
n=100	.0821	.0773	.0785	.2999	.2832
n=1000	.0933	.0900	.0892	.0794	.0895

the maximum likelihood estimate of β , the sample mean \bar{X} , is used as an estimate of standard error. The bootstrap grossly underestimates the standard error for all except the sample size of 1000. In fact the sample size of 10 is almost an order of magnitude off.

As listed in Table 4.1, the sample variance for the sample size of 10 is only .2777 while the true variance is 1. Figure 4.2 is a plot of the actual exponential distribution, denoted by the solid line, and the discrete nonparametric estimate of the actual distribution, denoted by the triangles. In this case the estimate of the distribution is a poor match with the actual distribution. Examining the ten observations, it can be seen that the largest value is 1.5989 and the smallest value is .0171. The maximum bootstrap variance occurs when five of the values selected are the smallest original observation and the other five values are the largest observation. This leads to a variance of .695. The smallest bootstrap variance occurs when all ten values are the same original observation, a variance of zero. The bootstrap variance is between zero and .695. There are ten different ways to get a zero variance as opposed to the one way to get .695, so that the distribution of the bootstrap estimate of sample variance is skewed. This is why the bootstrap estimate of standard error is so low.

To determine the relationship between sample variance and the bootstrap estimate of the standard error of the sample variance, one hundred random samples of size 10 were taken from an exponential distribution. For each of these samples, a bootstrap estimate of standard error using 500 replications was generated. Figure 4.3 shows the results of this experiment and indicates a strong linear relationship.

Figure 4.2
Comparison of Exponential Distribution with its Bootstrap
Estimate
(n=10, variance=1)

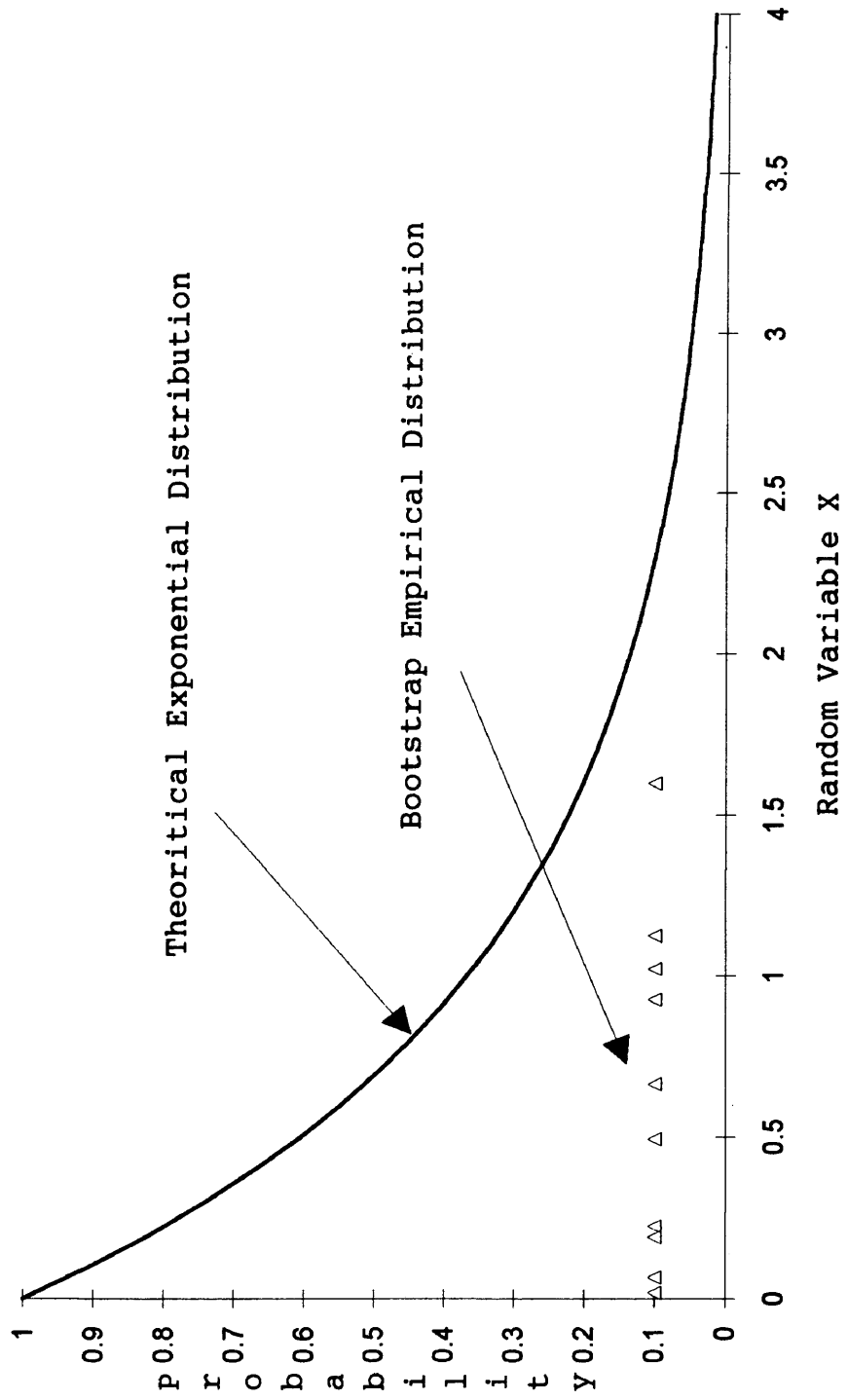
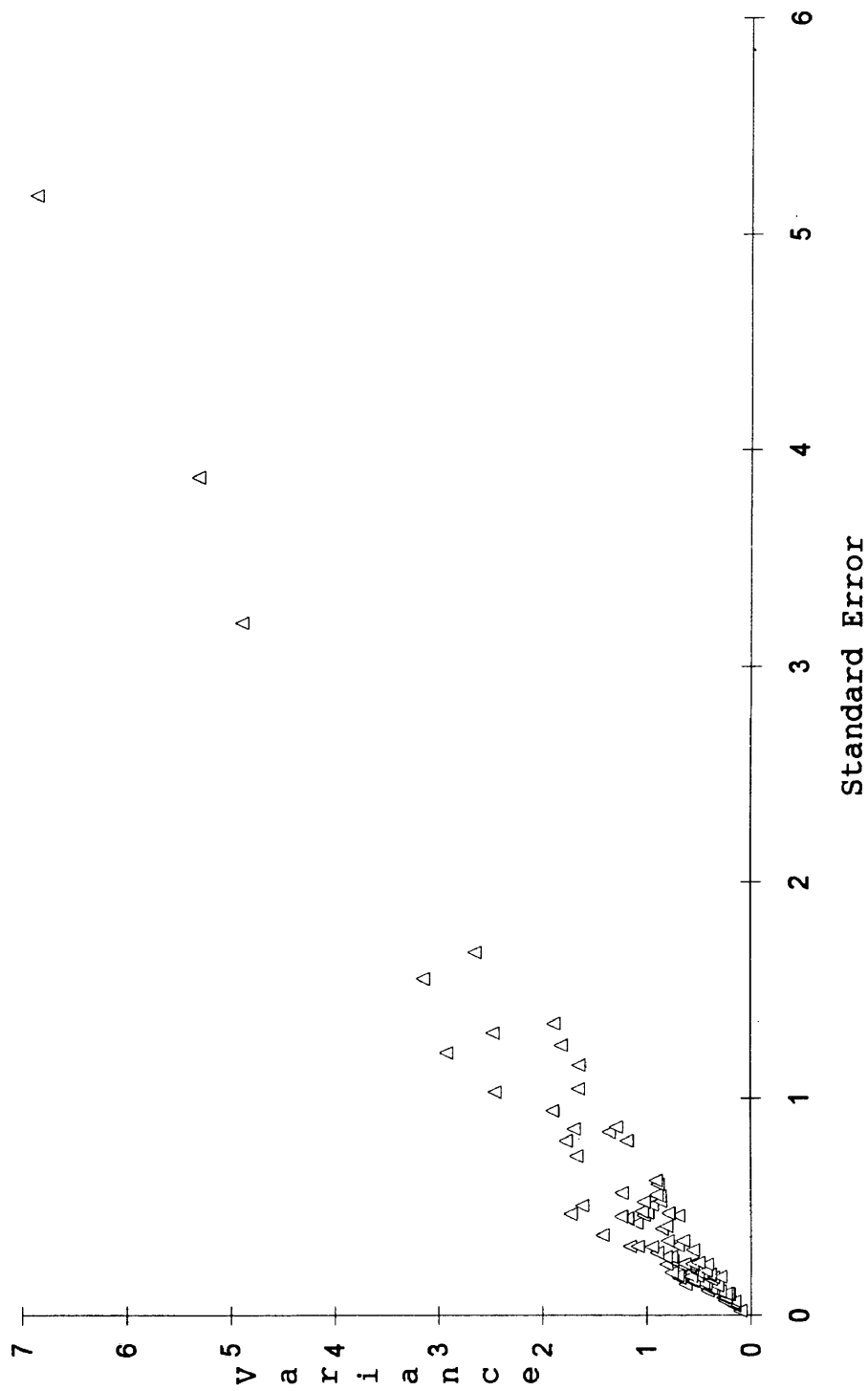


Figure 4.3
Bootstrap Results for the Standard Error of Variance
(Exponential Distribution $n=10$, variance 1)



Notice that the majority of the bootstrap estimates of standard error are less than the actual standard error of .9190. The problem appears to be due to the long tail of the exponential distribution. Eighty percent of the observations for the exponential population with mean and variance 1 are between 0 and 1.6, yet the variance of the sample variance for a sample size of 10 is .8222.

For comparison, 100 bootstrap estimates of the standard error of the sample variance were generated from a standard normal distribution. The samples were independent and of size 10. Figure 4.4 is a histogram of the results. Figure 4.5 is a histogram of the bootstrap estimates of standard error of random samples of size 10 from an exponential population, this is the data plotted on the x-axis in Figure 4.3. Remember from section 4.3 that the results for the bootstrap estimate of standard error of the sample variance for a sample size of 10 from the standard normal population were excellent. This was only one sample, which by coincidence had a sample variance that closely matched the population variance. Figure 4.4 demonstrates that the bootstrap also underestimates the standard error of the sample variance from the standard normal population. However, examining the chart you notice that there are quite a few values near the actual value of .4714. Contrasting, Figure 4.5 exhibits few values near the actual standard error of .9190. So the problem is more exaggerated for the exponential population.

To examine the effects of sample size, 100 independent samples of size 100 from an exponential distribution were generated. For each of these samples, the bootstrap estimate of standard error of the sample variance was calculated using 500 bootstrap replications. Figure 4.6

Figure 4.4
Histogram for the Standard Error of the Sample Variance
(Sample Size of 10 from Standard Normal Population)

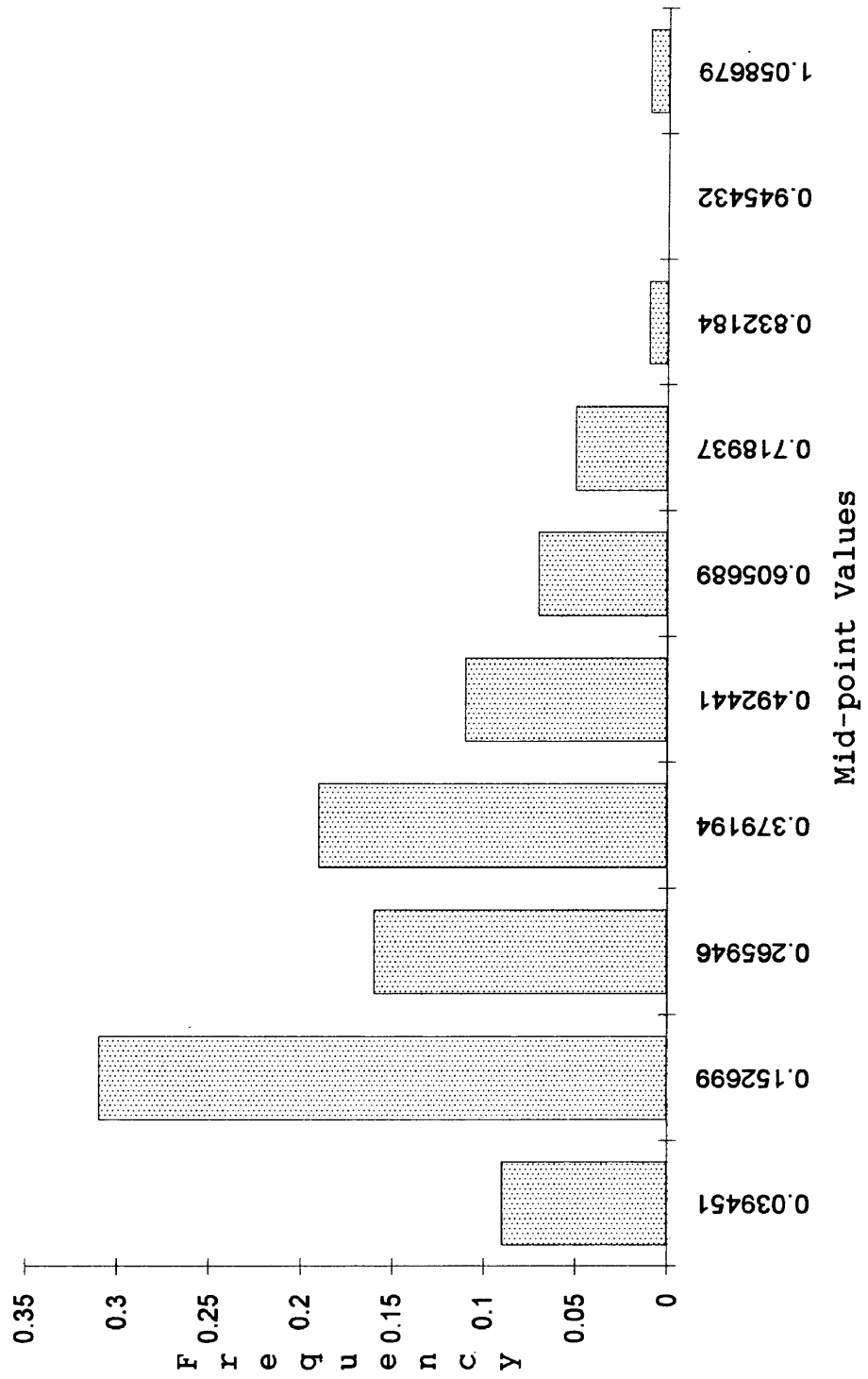


Figure 4.5
Histogram for Standard Error of the Sample Variance
(Sample Size of 10 from an Exponential Distribution)

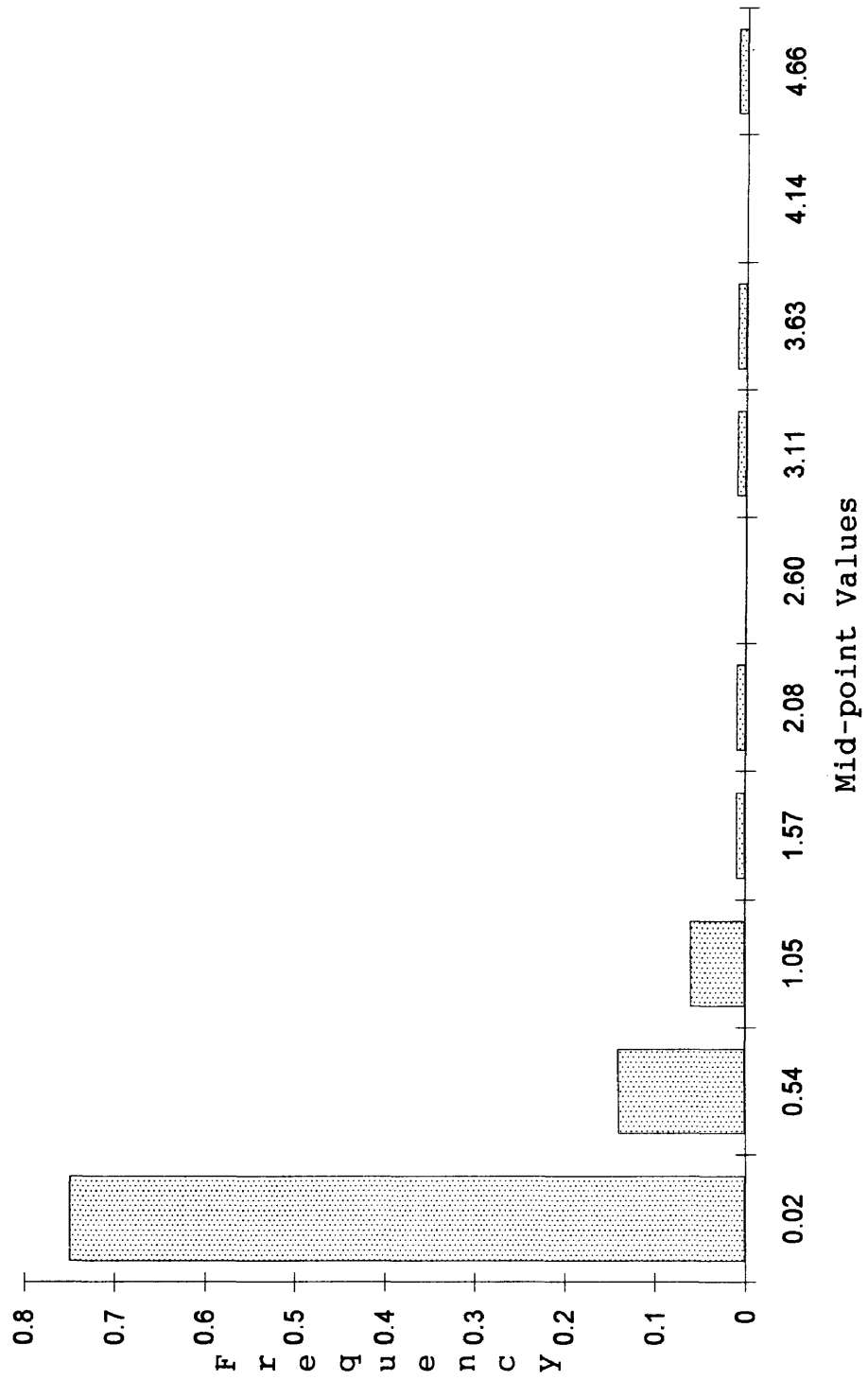
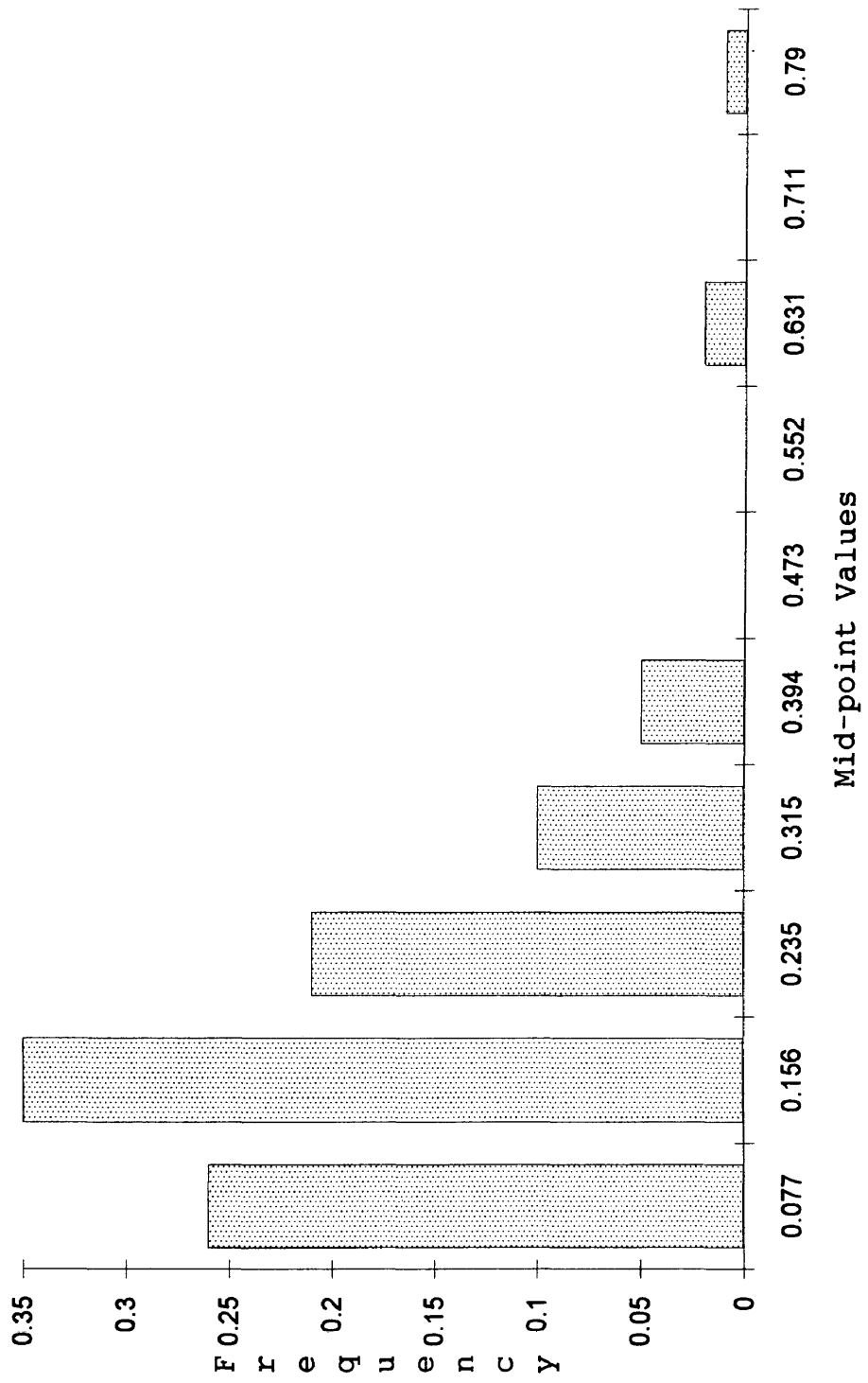


Figure 4.6
 Histogram for the Standard Error of the Sample Variance
 (Sample Size of 100 from an Exponential Population)



shows a histogram of the bootstrap estimates of standard error. The theoretical value of the standard error is .2832. Notice that the bootstrap again is underestimating the actual standard error; however, many values are closer to the actual value. This is an improvement from Figure 4.5 and is similar to Figure 4.4. The bootstrap estimate of standard error improves as the sample size increases.

It is possible to generate bootstrap confidence intervals for the sample variance from the exponential population, however, there is no commonly used confidence interval to use as a comparison. Without a comparison point, it is difficult to interpret the bootstrap confidence intervals. So bootstrap confidence intervals for sample variance from an exponential population are not listed in this paper.

4.6 Summary

In this chapter the limits and capabilities of the bootstrap were explored using the sample mean and sample variance. The population distribution, the sample size, and the statistic of interest have the biggest effect and are interrelated. The size of the bootstrap replication is of minor importance only; larger replications improve the stability of the bootstrap estimate of standard error.

The results of the experiments indicated that the statistic used in the bootstrap plays an important role in the outcome of the bootstrap simulation and is related to population distribution. For the sample mean, the bootstrap gave good results for both the normal and exponential populations. However, for the sample variance the bootstrap

had better results with the normal population than with the highly skewed exponential.

The bootstrap performance for both the standard error and confidence interval improved with sample size. For example, when the sample size was small, the confidence intervals for the sample variance did not exhibit the long tail. The actual sample size needed will depend on the population's distribution, which is often unknown, and the statistic of interest. For the sample mean, regardless of the population distribution, we achieve adequate results with sample sizes of 10. With sample variance, for a normal population a sample size of 25 would be appropriate while for the exponential we would need 50 to 100. The best procedure is to test the statistic of interest with simulated samples of varying sizes from populations that might be possible.

CHAPTER 5

APPLICATION OF THE BOOTSTRAP TO THE PROCESS CONTROL RATIO

5.1 Introduction

In this chapter the bootstrap simulation is used on the process control ratio called C_{pk} to demonstrate how the bootstrap can be adapted for lesser known statistics.

Process control ratios are used in statistical process control situations. They offer an advantage over zero defect quality control, which simply reports whether a product is in specification, because they account for process variability. For example, C_{pk} which is defined as

$$C_{pk} = \min\left(\frac{\text{upper spec} - \mu}{3\sigma}, \frac{\mu - \text{lower spec}}{3\sigma}\right), \quad (5.1)$$

is affected by the process variability and also by how well the process is centered between the two specifications. A higher C_{pk} indicates a higher quality product (Gunther 1989).

The problem with C_{pk} is that often μ and σ are not known. Estimates of these parameters have to be used in equation 5.1. Sample mean and sample standard deviation are commonly used as estimates of population mean and population standard deviation for C_{pk} . Thus

$$\hat{C}_{pk} = \min\left(\frac{\text{upper spec} - \bar{x}}{3s}, \frac{\bar{x} - \text{lower spec}}{3s}\right). \quad (5.2)$$

Note that this result is only an estimate of C_{pk} . Like all estimates, this estimate will have variability and therefore some error. The bootstrap provides a method to estimate the

standard error and confidence interval for the estimated C_{pk} .

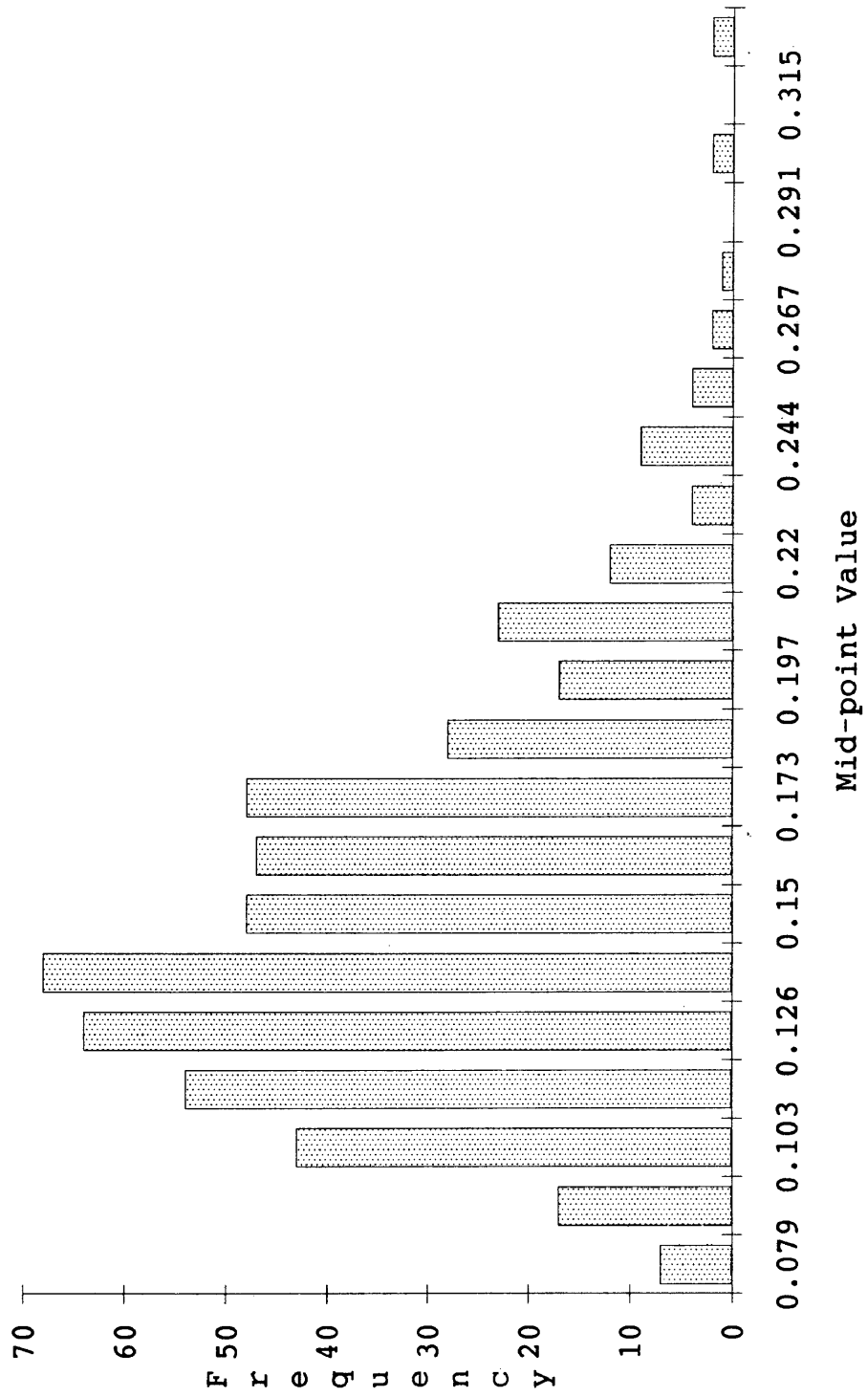
The bootstrap has been proposed for use with C_{pk} not only as a means to estimate confidence intervals but also to reduce sample size from 100 to only 20 (Wasserman, Mohsen, Franklin 1991). In Chapter 4, however, it was demonstrated the bootstrap improves with sample size. The claim of reduced sample size will be explored.

5.2 Bootstrap Standard Error for C_{pk} (Normal Population)

The population distribution from which C_{pk} is calculated is often assumed to be normal so that confidence intervals can be generated. This is the only population distribution that there are published confidence intervals for C_{pk} . The study of using the bootstrap on C_{pk} was started by taking 500 independent random samples each with 25 observations from the standard normal population. For each of these samples, a bootstrap estimate of the standard error for C_{pk} was generated based on 1000 bootstrap replications. Since the population was a standard normal, an upper specification of 3 and a lower specification of -3 were used in the calculation of C_{pk} . Figure 5.1 is a histogram of the standard error estimates. The mean of these 500 estimates is .1569. However, the maximum standard error is .338 and the minimum standard error is .079.

The distribution of C_{pk} is complex. So to obtain an estimate of standard error to compare with the bootstrap estimates, a simulation was used. The distribution of C_{pk} was simulated by generating 10000 random independent samples each with 25 observations. For each of these samples, an estimate of C_{pk} was calculated using equation 5.2. The

Figure 5.1
 Histogram for Bootstrap Standard Error for Cpk
 (Sample Sizes of 25 from a Standard Normal Population)



standard deviation of these 10000 values was used as an estimate of the standard error of C_{pk} . The value of this estimate of the standard error was .1538.

The mean of the 500 bootstrap estimates of standard error closely matches the value from the simulation. However, the minimum and maximum bootstrap estimates of standard error demonstrate there exists the possibility that for certain samples the estimate of standard error can be off by at least 100%. This must be tempered by the fact that 90% of the bootstrap estimates of standard error fell in the range of .1035 to .2307 and the standard deviation of the 500 bootstrap estimates was only .0413.

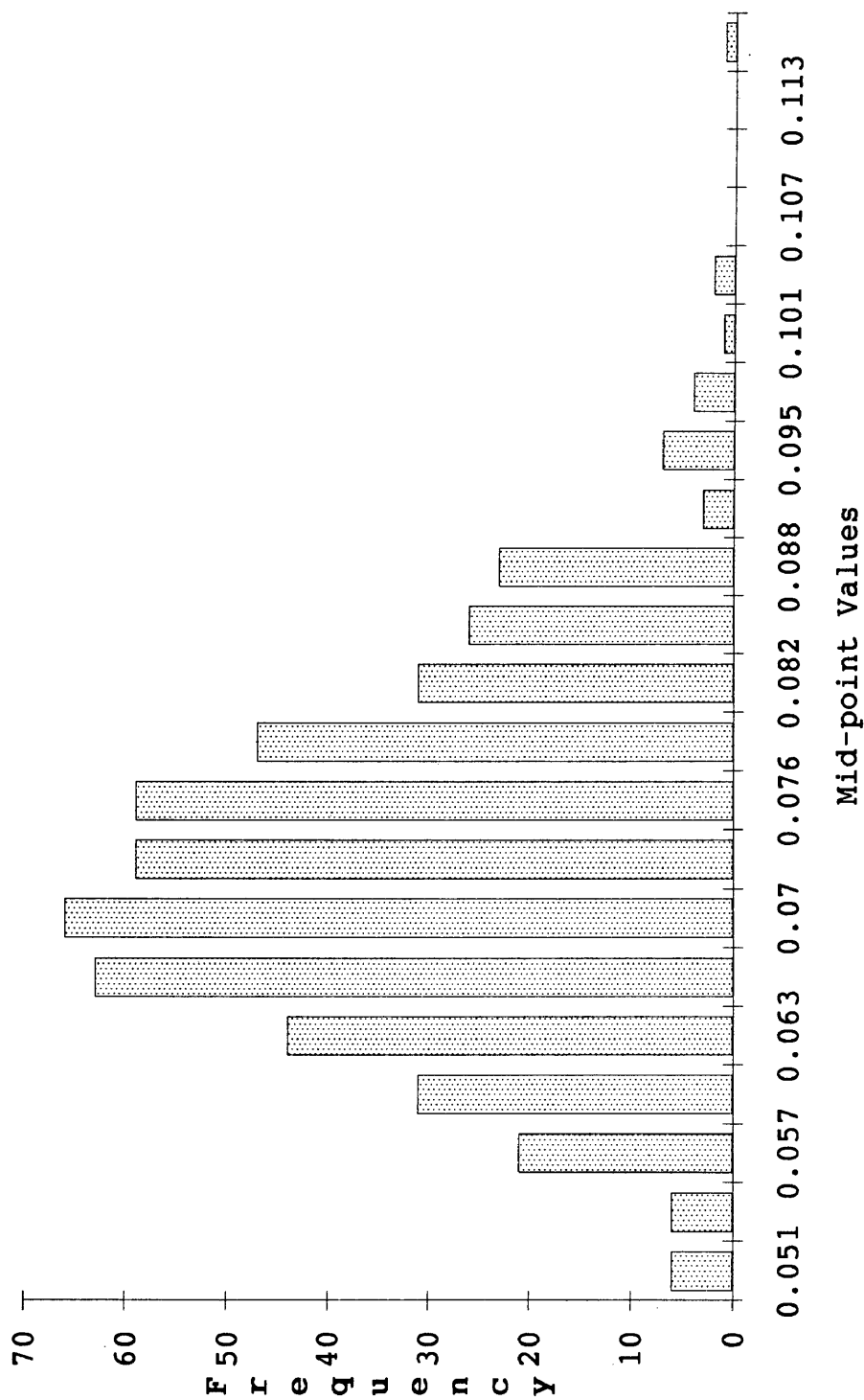
The same experiment was repeated again except that the sample size was increased to 100 observations. Figure 5.2 shows the histogram for the bootstrap estimates of standard error. The mean was .0741 with 90% of the values between .0592 to .0906. The standard deviation of the bootstrap estimate of standard error was .01. The results from the simulation of the distribution of C_{pk} by using 10000 values yielded an estimate of standard error at .0733.

The bootstrap provides an accurate method to estimate the standard error of C_{pk} even for sample sizes as small as 25. As sample size increases the accuracy of the bootstrap estimate of standard error improves because the size of the standard error is reduced and the variability in the estimate of standard error is smaller.

5.3 Bootstrap Confidence Intervals for C_{pk}

Confidence intervals are preferred over standard error because they offer a bound on the estimate of C_{pk} . The distribution of C_{pk} does not have to be symmetrical, so it

Figure 5.2
 Histogram for Bootstrap Standard Error for Cpk
 (Sample Sizes of 100 from a Standard Normal Population)



is difficult to translate standard error into a lower bound. In this section, the performances of the bootstrap confidence intervals for C_{pk} are examined. Two samples from a standard normal population, one with twenty observations and the other with one hundred observations, will be used to generate confidence intervals. Table 5.1 summarizes the descriptive statistics for the two samples. An estimate of C_{pk} using equation 5.2 is also presented. Again an upper specification of 3 and a lower specification of -3 were used in the calculation of C_{pk} .

Since it is desired that C_{pk} be as large as possible, we are usually only concerned with the lower limit of the confidence interval. A procedure has been proposed for generating lower confidence limits on C_{pk} when the population has a normal distribution (Chou, Owen, Borrego 1990). The equations developed can not be solved explicitly, so numerical results are tabulated and presented in the referenced document. The reference confidence intervals reported in Chapter 5 are taken from these tables.

Table 5.1
Summary Statistics for Test Samples

	Normal Distribution ($\mu=0, \sigma^2=1$)		
Sample Size	Sample Mean	Sample Variance	Estimated C_{pk}
20	-.1317	1.0026	.9549
100	.2181	.7240	1.2808

The bootstrap program provides a two-sided confidence interval but it can easily be adopted as a one-sided confidence interval. For example, when .90 is entered as the confidence level, then each of the two limits has .05 of the total of .10 uncertainty. Therefore, the lower limit taken by itself in this example would have a 95% confidence level. To achieve a 95% one-sided confidence level from the bootstrap, generate a 90% two-sided confidence level and take only the lower value.

Using the sample size and estimate of C_{pk} a reference 95% lower confidence limit is obtained from the tables. Bootstrap lower confidence levels were generated by using 1000 replications. The results of the experiment are presented in Table 5.2. The true value of C_{pk} from equation 5.1 is 1. The bootstrap estimate of the lower confidence interval was the same as the theoretical result for the sample size of 20. This sample had a sample mean and variance that was nearly the population mean and variance. The sample of 100 had a mean and variance that differed more drastically from the population mean and variance. In

Table 5.2
95% Lower Confidence Limit for C_{pk}
(Standard Normal Population)

Sample Size	Bootstrap Lower Confidence Limit	Reference Lower Confidence Limit
n=10	$0.71 < C_{pk}$	$0.70 < C_{pk}$
n=100	$0.94 < C_{pk}$	$1.13 < C_{pk}$

this case the bootstrap lower confidence limit was better than the theoretical confidence limit since it includes the actual C_{pk} value.

5.4 Bootstrap of C_{pk} for a Log-Normal Population

The bootstrap performed well when the population had a normal distribution. This is an ideal case that seldom exists in practice. To truly test the bootstrap, a skewed distribution must be used. The log-normal distribution was selected. If X is a normally distributed random variable, then $Y=e^X$ has a log-normal distribution. To generate the distributions in this section, random samples from a standard normal distribution were obtained and then e was raised to the power of the observations in the sample.

Since the log-normal distribution is bounded at zero, C_{pk} as defined in equation 5.1 does not apply. The log-normal distribution simulates a one-sided process that does not have a lower specification. So the definition of C_{pk} was modified to

$$C_{pk} = \frac{\text{upper spec} - \mu}{3\sigma}. \quad (5.3)$$

An upper specification of 20.1 was used for the calculations in this section. 20.1 is the transformation of the point from the standard normal distribution that incorporates 99.87% of the distribution. This means that 99.87% of the points in the log-normal distribution will be less than 20.1.

For this study, one sample of size 25 and one sample of size 100 was used. Using 1000 bootstrap replications, a bootstrap estimate of standard error and a lower 95%

confidence limit for each sample was generated. There is no applicable procedure in the literature to develop a reference confidence interval or standard error to compare to the bootstrap results. A simulation was used to generate a reference. In the simulations, 10000 random samples of size 25 and 10000 random samples of size 100 from the log-normal distribution were used. Using equation 5.3, a C_{pk} value from each of the random samples was obtained. Thus there were simulation estimates of the distribution of C_{pk} for sample sizes of 25 and 100 from a log-normal population. The reference estimate of the standard error of C_{pk} is the standard deviation of the 10000 simulation values. The reference estimate of the 95% lower confidence limit is the 500th smallest value from the 10000 simulation values.

Table 5.3 summarizes the results. The actual C_{pk} is 2.8458. Notice that for the sample size of 25 the bootstrap differs greatly from the reference values, especially the

Table 5.3
Bootstrap and Simulation Results for
 C_{pk} from a Log-Normal Population

Sample Size	Bootstrap Results		Simulation Results	
	Standard Error	95% Lower Confidence Limit	Standard Error	95% Lower Confidence Limit
25	1.2555	8.3768 < C_{pk}	1.7881	1.5984 < C_{pk}
100	.7343	2.6062 < C_{pk}	.9054	1.8325 < C_{pk}

95% lower confidence level. For this distribution, you would be better off using a larger sample size such as 100, because as Table 5.3 demonstrates, the sample size of 100 yielded excellent bootstrap results.

5.5 Summary

Similar to the results in Chapter 4, the bootstrap performance for C_{pk} depended on sample size and population distribution. If the population had a normal distribution, the bootstrap yielded acceptable results even with a small sample size of 20. However, for a skewed distribution, the bootstrap was suspect at small sample sizes and it would be prudent to use a large sample size. The bootstrap works well for C_{pk} especially since it gives results even when the assumption of a normal population is violated. Current literature requires the assumption of a normal population to derive confidence limits.

CHAPTER 6

CONCLUSION

This paper has discussed the developed of a computer based simulation of the bootstrap. The bootstrap was discussed, the computer program was explained, the usefulness and limitations of the bootstrap were explored, and applying the bootstrap was demonstrated. The primary contribution of the paper was the use of the bootstrap on two common statistics, sample mean and sample variance, to gain a thorough understanding of the bootstrap.

The bootstrap is a tool that uses resampling from an original set of observations to obtain estimates of statistical error for a statistic of interest. The advantage of the bootstrap is that it makes no assumption about the underlying population distribution. From the bootstrap, estimates of statistical error such as standard error, bias, or a confidence interval can be obtained.

The accuracy of the bootstrap depends on sample size, population distribution, and the statistic of interest. As the sample size increases, the bootstrap becomes asymptotically correct. For small sample sizes the performance of the bootstrap depends both on the population distribution and the statistic of interest. These two factors are interrelated. For some statistics, sample mean, the population distribution makes little difference. However, for other statistics the bootstrap performs well for small sample sizes for some population distributions and for other population distributions the sample size must be

increased to achieve the same level of performance from the bootstrap.

The computer program discussed in this paper provides a method to simulate the bootstrap for any statistic of interest. The program is written in the C programming language and compiled on Borland's Turbo C compiler. A user can easily add any statistic of interest; however, they must be able to program in C and have the original source files to link with their code. The code is structured so that the user has to only have minimal familiarity with the program.

REFERENCES CITED

- Chou, Youn-Min, D. B. Owen, and Salvador A. Borrego. 1990. Lower Confidence Limits on Process Capability Indices. Journal of Quality Technology, Vol. 22, No. 3: 223-229.
- Davison, A. C., D. V. Hinkley, and E. Schechtman. 1986. Efficient bootstrap simulation. Biometrika, Vol. 73, No. 3: 555-566.
- Efron, Bradley. 1982. The Jackknife, the Bootstrap and Other Resampling Plans. Philadelphia: Conference Series in Applied Mathematics, Report 38, Society for Industrial and Applied Mathematics.
- Efron, Bradley. 1987. Better Bootstrap Confidence Intervals. Journal of the American Statistical Association, Vol. 82, No. 397: 171-185.
- Fruend, John E. 1992. Mathematical Statistics. Englewood Cliffs: Prentice-Hall.
- Gleason, John R. Algorithms for Balanced Bootstrap Simulations. The American Statistician, Vol. 42, No. 4: 263-266.
- Gunther, Berton H. 1989. The Use and Abuse of C_{pk} . Quality Progress, Vol. XXII, No. 1: 72-73.
- Mood, Alexander M., Franklin A. Graybill, and Duane C. Boes. 1974. Introduction to the Theory of Statistics. New York: McGraw-Hill.
- Wasserman, Gary S., LeRoy A. Franklin, and Hassan A. Mohsen. 1991. A Program to Calculate Bootstrap Confidence Intervals for the Process Capability Index, C_{pk} .

Communications in Statistics-Simulation, Vol. 20, No.
2&3: 497-510.

ADDITIONAL READINGS

- Craig, R. J. 1984. Normal Family distribution Functions: FORTRAN and BASIC Programs. Journal of Quality Technology, Vol. 16, No. 1-4: 232-236.
- DiCiccio, Thomas, and Robert Tibshirani. 1987. Bootstrap Confidence Intervals and Bootstrap Approximations. Journal of the American Statistical Association, Vol. 82, No. 397: 163-170.
- Efron, Bradley, and Gail Gong. 1983. A Leisurely Look at the Bootstrap, the Jackknife, and Cross-Validation. The American Statistician, Vol. 37, No. 1: 36-48.
- Efron, Bradley. 1988. Bootstrap Confidence Intervals: Good or Bad ?. Psychological Bulletin, Vol. 104, No. 2: 293-296.
- Efron, Bradley, and Robert Tibshirani. 1991. Statistical Data Analysis in the Computer Age. Science, Vol. 253: 390-395.
- Griffiths, P., and I. D. Hill. 1985. Applied Statistics Algorithms. West Sussex, England: Ellis Horwood Limited.
- Shi, Sheng G. 1992. Accurate and efficient double-bootstrap confidence limit method. Computational Statistics & Data Analysis, Vol. 13: 21-32.

APPENDIX

File: allus.h

```
# include <stdio.h>
# include <stddef.h>
# include <conio.h>
# define DIM1 1000
# define DIM2 10000

/*Functions in input.c */

extern int read_input();
extern FILE *myopen();
extern void write_out();
extern float flt_query();
extern int int_query();
extern int char_query();
extern void keyboard_input();
extern void my_pause();
extern void hist();

/* Functions in boot.c */
extern float cpn();
extern void boot();
extern float mean();
extern float booterr();
extern float boot_stderr();
extern float reboot();
extern float my_median();
extern float norm_cdf();
extern float norm_inverse();
extern void ci();
extern float cpk();

/* Functions in menu.c */
extern text_mode();
extern init_menu();
extern credit_menu();
extern boot_menu();
```

```
extern stat_menu();  
extern input_menu();  
extern begin_screen();
```

File: boot.c

```

/* *****
Copyright 1993

This file contains the functions to perform the bootstrap:
    booterr
    boot_stderr
    boot
    ci
    cpk
    mean
    my_median
    my_sort
    norm_cdf
    norm_inverse
    reboot

Brad Warner      Revision 0      Feb. 23, 1993
***** */

# include <stdio.h>
# include <stddef.h>
# include <stdlib.h>
# include <math.h>
# include <time.h>

# define DIM1 1000      /* The maximum number of data points */
# define DIM2 10000    /* The maximum number of bootstraps */
# define BOOT_ITEM_1 1 /* First menu item on boot menu */
# define BOOT_ITEM_2 2 /* Second menu item on boot menu */
# define BOOT_ITEM_3 3 /* Third item on boot menu */
# define BOOT_ITEM_4 4 /* Fourth item on boot menu */
# define TRUE 1
# define FALSE 0
# define ZERO 0.0
# define ONE 1.0
# define HALF 0.5
# define PI 3.141592653589793
# define SQRT2 1.414213562373095

extern float lower_cpn;      /* Lower value for Cpn calculations */
extern float upper_cpn;    /* Upper value for Cpn calculations */
extern int BOOT_FLAG;      /* Which statistic did we use */
extern void percent_done();

```

```
int big_m=32767;      /* Upper bound on integers 215 -1 make global */
int flag=0;          /*A flag so that randomize is used once per program */
```

```
/* *****
Function: boot
```

Description: This function uses a balanced bootstrap approach to calculate the bootstrap distribution for the statistic desired by the user.

Arguments: The number of data points, the number of bootstrap replications,
a pointer to the array holding the data, a pointer to the array
the holds the bootstrap distribution, and a flag if a
printout of the percent complete.

Pseudo-code: 1. Set up array with observations and another array with the number of of observations (# of bootstrap replications).

2. Choose at random from the list
 - a) Select a random j
 - b) Accept j if $s/big_m < cj/c_m$ this generates a random list
 - c) If necessary pick new largest cj and adjust arrays.
Note: the variable adjust allows you to reduce the number of updates that occur. With adjust equal to zero c_m is updated every time. Other choices will reduce the number of updates of c_m.

3. Process the bootstrap sample

Returns: None.

```
***** */
```

```
void boot(n,b,n_array,b_array,reboot_flag)
int b;          /* The number of bootstraps */
int reboot_flag; /* Do you want a print out of % complete? */
int n;         /* The number of data points */
float n_array[]; /* The pointer to the array holding the data */
float b_array[]; /*Pointer to array that holds bootstrap distribution*/
{
double c_m;     /* Used for calculating prob. for accepting j */
int adjust;    /* Determines if we have to find new max_c */
int array_cj[DIM1]; /* Number of observation j left */
int h;        /* Counter variable for loops */
```

```

int i;                /* Counter variable for loops */
int ii;              /* Counter variable for loops */
int j;              /* The data value [1 to n] */
int dist;          /* Number of distinct data points left */
int largest_cj;    /* Subscript on largest index */
int max_c;        /* Largest index */
int randint;      /* Random integer up to big_m */
float *ar_of_ptrs[DIM1]; /* Pointers to the original data */
float array_boot[DIM1]; /* Bootstrap sample */
float booterr();
float boot_stderr();
float cpk();
float mean();
float my_median();
float *selected;   /* Pointer to the value selected */
float std_err;    /* Bootstrap standard error */

/* Psuedo-Code 1 */

/* Initialization */

while(!flag){
  randomize();
  flag = 1;} /* End while */

for(h=0;h < n; h++){
  array_cj[h] = b;
  ar_of_ptrs[h] = (n_array + h);} /* End for */

dist = n-1; /* Since arrays run 0 to n-1 */
largest_cj = dist;
max_c = b;
c_m = (double) max_c / (double) big_m;

/* Psuedo-code 2a */

/* Choose at random from the list */

for(h=0;h<b;h++){ /* This loop controls the replications */
  for(i=0;i<n;i++){ /* This loop makes a sample of size n */
    do{
      /* Random number between 0 and 32766 */
      randint = random(big_m) ;

      /* Gives a number from 0 to dist */
      j = (randint % (dist+1));

      /* Psuedo-code 2b */

```

```

/* Accepts j with Prob. array_cj/max_c */
}while(((double) randint*c_m) > (double) array_cj[j]);

/* stores the address of data value */
selected = ar_of_ptrs[j];

/* Now one less value in array_cj to select */
array_cj[j] -= 1;

/* Psuedo-code 2c */

/* Need to update what the largest value is */
if(j == largest_cj){

/* A conditional so max_c is not updated every loop */
adjust = ((b-h+dist+2)/(dist+1));
if((max_c - array_cj[largest_cj]) > adjust){

/* Find the point in the array with the most elements left*/

        for(ii=0;ii<=dist;ii++){
            if(array_cj[ii] > array_cj[largest_cj])
                largest_cj = ii;} /* End for */

/*Now set max_c equal to the largest number of data points left */
max_c=array_cj[largest_cj];
c_m = (double) max_c / (double) big_m;}} /*End ifs */

/* Check to see if all of the elements in array_cj[j]
have been selected */

if(array_cj[j] == 0){

/* Reconfigure so the empty slot will not be selected */

        if(largest_cj == dist)largest_cj = j;
        ar_of_ptrs[j] = ar_of_ptrs[dist];
        array_cj[j] = array_cj[dist];
        dist -= 1;} /* End if */
array_boot[i] = *selected; /*Store the current bootstrap sample*/
} /* End for */

/* Psuedo-code 3 */

switch (BOOT_FLAG){
case BOOT_ITEM_1:
    b_array[h] = mean(array_boot,n);

```

```

        break;

    case BOOT_ITEM_2:
        b_array[h] = booterr(array_boot,n); /* booterr is variance */
        break;

    case BOOT_ITEM_3:
        b_array[h] = my_median(array_boot,n);
        break;

    case BOOT_ITEM_4:
        b_array[h] = cpk(array_boot,n);
        break;} /* End switch */
if(reboot_flag)percent_done(h,b);} /* End for */
return;
}

/* *****
Function: booterr

Description: This function calculates the sample variance.

Arguements: The sample array and the number of data points.

Returns: A float of the variance.
***** */
float booterr(b_array,b)
int b;
float b_array[];
{
    int counter;
    float boot_mean;
    float delta;
    float mean();
    float sq_err=0;

    /* Get the mean of the bootstrap samples */

    boot_mean = mean(b_array,b);

    /* Get the sum of the squares of sample minus mean */

    for(counter=0;counter<b;counter++){
        delta = b_array[counter] - boot_mean;
        sq_err += (delta * delta);} /* End for */

    /* Variance is the square error divided by # of data points minus 1 */

```

```

    sq_err = sq_err/(float)(b-1);
    return sq_err;
}

```

```

/* *****
Function: boot_stderr

```

Description: This function calculates the sample standard error.

Arguements: The sample array and the number of data points.

Returns: A float of the standard error.

```

***** */

```

```

float boot_stderr(b_array,b)

```

```

int b;

```

```

float b_array[];

```

```

{

```

```

    float std_err;

```

```

    /* Get the variance of the sample */

```

```

    std_err = booterr(b_array,b);

```

```

    std_err = sqrt((double)std_err);

```

```

    return std_err;

```

```

}

```

```

/* *****

```

```

Function: ci

```

Description: This function calculates the confidence interval for the bootstrap distribution. It uses a bias corrected method.

Arguements: The confidence interval array, bootstrap array, the number of data points, the original array, and its number of data points.

Returns: None.

```

***** */

```

```

void ci(ci_limits,b_array,b,n_array,number)

```

```

float ci_limits[];

```

```

float b_array[];

```

```

float n_array[];

```

```

int b;

```

```

int number;

```

```

{

```

```

extern float con_level;
float alpha;
float cpk();
float mean();
float my_median();
float norm_cdf();
float norm_inverse();
float orig_value;
float p;
float per_lower;
float per_upper;
float temp;
float z_bias;
float z_normal;
int counter;
int int_per_lower;
int int_per_upper;
int j=0;          /* Number of values less than the original value */
int k=0;          /* Number of values equal to the original value */
int list;         /* For partial sorting */
int sorted = FALSE; /* For partial sorting */

/* Find the rank of the original statistic */
switch (BOOT_FLAG){
  case BOOT_ITEM_1:
    orig_value = mean(n_array,number);
    break;

  case BOOT_ITEM_2:
    orig_value = booterr(n_array,number);
    break;

  case BOOT_ITEM_3:
    orig_value = my_median(n_array,number);
    break;

  case BOOT_ITEM_4:
    orig_value = cpk(n_array,number);
    break;} /* End switch */

/* Find number of values less than original value and half the number
equal to original value */
for(counter=0;counter<b;counter++){
  if(orig_value>b_array[counter])j++;
  if(orig_value == b_array[counter])k++;} /* End for */
  p=(float)(j+(k+1)/2)/(float)b;

/* Find invervse normal of p and alpha values */

```

```

    alpha = con_level + (1.0-con_level)/2.0;
    z_normal= norm_inverse(alpha);
    z_bias = norm_inverse(p);

    /* Find the percent corresponding to z-values calculated */
    per_lower = norm_cdf((2*z_bias - z_normal));
    per_upper = norm_cdf((2*z_bias + z_normal));
    int_per_lower = per_lower*(b-1)+ HALF;
    int_per_upper = per_upper*(b-1) + HALF;

    /* Partially sort upper 'int_per_upper' values of the array */
    list=b-1;
    while(!sorted){
        sorted = TRUE;
        for(j=0;j<list;j++){
            if(b_array[j] > b_array[j+1]){
                temp = b_array[j];
                b_array[j] = b_array[j+1];
                b_array[j+1] = temp;
                sorted = FALSE;} /* End if */
            if(list == int_per_upper)break; /* Ending criteria */
            list--;} /* End while */
    /* Partially sort lower 'int_per_lower' values of the array */
    list=0;
    sorted = FALSE;
    while(!sorted){
        sorted = TRUE;
        for(j=int_per_upper-1;j>list;j--){
            if(b_array[j] < b_array[j-1]){
                temp = b_array[j];
                b_array[j] = b_array[j-1];
                b_array[j-1] = temp;
                sorted = FALSE;} /* End if */
            if(list == int_per_lower)break; /* Ending criteria */
            list++;} /* End while */

    /* Set confidence interval */
    ci_limits[0] = b_array[int_per_lower];
    ci_limits[1] = b_array[int_per_upper];

    return;
}

```

```

/* *****
Function: cpk

Description: This function calculates the process control ratio Cpk.
This function is defined as  $\min[(\text{mean}-\text{lower})/3\tau, (\text{upper}-\text{mean})/3\tau]$ 
Where:  $\tau = (\text{varaince})^{.5}$ 
lower = lower specification limit
upper = upper specification limit

Arguements: The sample array and the number of data points.

Returns: A float of the Cpk.
***** */
float cpk(cpk_array,number)

float cpk_array[];
int number;

{
float cpk_lower;
float cpk_upper;
float mean();
float mean_cpk;
float tau;
float var_cpk;

mean_cpk = mean(cpk_array,number);
var_cpk = booterr(cpk_array,number);
tau = sqrt(var_cpk);
cpk_lower = (mean_cpk - lower_cpn)/(3 * tau);
cpk_upper = (upper_cpn - mean_cpk)/(3 * tau);
if (cpk_lower > cpk_upper) return cpk_upper;
else return cpk_lower;
}

/* *****
Function: mean

Description: This function calculates the mean of the array.

Arguements: The sample array and the number of data points.

Returns: A float of the mean of the sample.
***** */
float mean(array_boot,n)
int n; /* Number of sample points */
float array_boot[]; /* Sample data */

```

```

{
  int counter;
  float result=0;          /* The mean */

  for(counter=0;counter<n;counter++)result += array_boot[counter];

  result = result/(float) n;
  return result;
}

```

```

/* *****
Function: my_median

```

Description: This function calculates the median of the array.

Arguements: The sample array and the number of data points.
If the number of points is even it takes the larger order point.

Returns: A float of the median of the sample.

```

***** */

```

```

float my_median(array_boot,n)
int n;          /* Number of sample points */
float array_boot[]; /* Sample data */
{
  float result;
  int med;
  void my_sort();

  my_sort(array_boot,n);
  med = n/2;
  result = array_boot[med];
  return result;
}

```

```

/* *****
Function: my_sort

```

Description: This function sorts an array of floats using a bubble sort.

Arguements: The sample array and the number of data points.

Returns: None.

```

***** */

```

```

void my_sort(array_boot,n)
int n;          /* Number of sample points */
float array_boot[]; /* Sample data */

```

```

{
  int j;
  int k;
  int list = n;
  int sorted = FALSE;
  float temp;
  while(!sorted){
    sorted = TRUE;
    for(j=0;j<list-1;j++){
      if(array_boot[j] > array_boot[j+1]){
        temp = array_boot[j];
        array_boot[j] = array_boot[j+1];
        array_boot[j+1] = temp;
        sorted = FALSE;} /* End if */
    list--;} /* End while */
  return;
}

```

```

/* *****
Function: norm_cdf

```

Description: This function calculates the lower tail area of the standardized normal curve.

Arguments: The z_value.

Returns: A float of the area.

```

***** */

```

```

float norm_cdf(z_value)
float z_value;
{
  float erf;
  float rn;
  float s;
  float y;
  float area;
  int counter;

  y = z_value/SQRT2;
  if(z_value < ZERO)y=-y;
  s=ZERO;
  for(counter=1;counter<38;counter++){
    rn=counter;
    s=s+exp(-rn*rn/25.0)/counter*sin(2.0*counter*y/5.0);} /* End for */
  s=s+y/5.0;
  erf=2.0*s/PI;
  area=(ONE+erf)/2.0;

```

```

    if(z_value < ZERO) return (ONE-erf)/2.0;
    if(z_value < -8.3) return ZERO;
    if (z_value > 8.3) return ONE;
    return area;
}

```

```

/* *****
Function: norm_inverse

```

Description: This function calculates the z_value given the p value of area for the standardized normal curve.

Arguements: The p value.

Returns: A float of the z_value.

```

***** */
float norm_inverse(p_value)
float p_value;
{
    float a0 = 2.50662823884;
    float a1 = -18.61500062529;
    float a2 = 41.39119773534;
    float a3 = -25.44106049637;
    float b1 = -8.47351093090;
    float b2 = 23.08336743743;
    float b3 = -21.06224101826;
    float b4 = 3.13082909833;
    float c0 = -2.78718931138;
    float c1 = -2.29796479134;
    float c2 = 4.85014127135;
    float c3 = 2.32121276858;
    float d1 = 3.54388924762;
    float d2 = 1.63706781897;
    float split = .42;
    float q;
    float r;
    float z_value;

    q = p_value - HALF;
    if(fabs(q) <= split){
        r = q*q;
        z_value = q*(((a3*r+a2)*r+a1)*r+a0)/((((b4*r+b3)*r+b2)*r+b1)*r+ONE);
        return z_value;} /* End if */
    r = p_value;
    if (q > ZERO)r = ONE - p_value;
    if (r <= ZERO){
        z_value = -1000.0;
        return z_value;} /* End if */
}

```

```

    r=sqrt(-log(r));
    z_value=((c3*r+c2)*r+c1)*r+c0/((d2*r+d1)*r+ONE);
    if(q < ZERO)z_value=-z_value;
    return z_value;
}
/* *****
Function: reboot

Description:  This function calculates the distribution of standard
error.

Arguements:  The sample array, boot strap array, and reboot array
              and the number of data points for each.

Returns:      A float of the standard error of the standard error.
***** */
float reboot(n_array,b_array,re_boot,number,b,reb)

float b_array[];
float n_array[];
float re_boot[];
int b;
int number;
int reb;
{
float re_stderr;
int counter;

for(counter=0;counter<reb;counter++){
    boot(number,b,n_array,b_array,FALSE);
    re_boot[counter] = boot_stderr(b_array,b);
    percent_done(counter,reb);} /* End for */
re_stderr = boot_stderr(re_boot,reb);
return re_stderr;
}

```

File: input.c

```
/* *****
```

```
Copyright 1993
```

```
This file contains functions to input and output data:
```

```
char_query
flt_query
hist
int_query
myopen
my_pause
percent_done
read_input
write_out
```

```
Brad Warner      Revision 0      Feb. 23, 1993
```

```
***** */
```

```
# include <stdio.h>
# include <stddef.h>
# include <math.h>
# include <dos.h>
# include <dir.h>
# include <conio.h>
```

```
/* *****
```

```
Function: char_query
```

```
Description: This function asks for a character answer
and checks if it is yes or no.
```

```
Arguements: Question string.
```

```
Returns: Returns a 1 if yes and a 0 if no.
```

```
***** */
```

```
int char_query(str)
```

```
char *str;
{
char input_char;
int x;
int y;
void my_pause();
```

```

x=wherex();
y=wherey();
while(1){                                     /* Loop until get y or n */
gotoxy(x,y);
fflush(stdin);
printf("\n%s? [y/n]:",str);
scanf("%c",&input_char);
    switch (input_char)
    {
    case 'y':
    case 'Y':
        return 1;
    case 'n':
    case 'N':
        return 0;
    default :
        printf("\nPlease enter y or n.");
        my_pause();
        gotoxy(x,y+4);
        clreol();
        gotoxy(x,y+5);
        clreol();
        gotoxy(x,y+2);
        clreol();    }
}
}

```

```

/* *****
Function: flt_query

```

Description: This function asks for an float input and checks if it is in the specified range.

Arguements: Question string, lower bound, upper bound.

Returns: Returns float value.

```

***** */
float flt_query(str,lb,ub)

```

```

char *str;
float lb;
float ub;

```

```

{
float input_int;
int x;
int y;

```

```

void my_pause();

x=wherex();
y=wherey();
do
{
gotoxy(x,y);
printf("\n%s [%.2f...%.2f]:",str,lb,ub);
scanf("%f",&input_int);
if(input_int < lb || input_int > ub)
    {
        printf("\n\t\t%.2f is not in the range
[%.2f...%.2f]",input_int,lb,ub);
        my_pause();
        gotoxy(x,y+3);
        clreol();
        gotoxy(x,y+4);
        clreol();
        gotoxy(x,y+1);
        clreol();
    }
}while(input_int < lb || input_int > ub);
return input_int;
}

```

```

/* *****
Function: hist

```

Description: This function produces a crude histogram.

Arguements: A pointer to the data array, the number of data points in the array, and the output device flag.

Returns: None.

```

***** */
void hist(data_array,number,output_stream)

```

```

float data_array[];
int number;
FILE *output_stream;    /* pointer to output file */
{
    char a_blank = ' ';
    char a_dash = '-';
    char a_star = '*';
    char iout[10];
    float c;
    float div;
    float eta = 1.0E-38;

```

```

float out[10];
float range;          /* xmax - xmin */
float step;
float temp;
float tk;
float xm;
float xmax;          /* maximum of data */
float xmin;          /* minimum of data */
int classes;         /* Number of groups to divide the data */
int counter;
int counter2;
int flag;            /* Tell if gone through range adjustment loop */
int freq[10];        /* The frequency output */
int index;
int int_step;
int int_temp;        /* integer value of temp */
int k;
int key;
int l;
int leng;
int length = 50;     /* Number of lines of output */
int scale;

/* Get initial data and initialize screen */
clrscr();
fprintf(output_stream, "\t\t\t Histogram \n\n\n\n");
classes = int_query("Enter the number of groups", 2, 10);
leng = length - 10;
textmode(64);
clrscr();

/* Initialize data and arrays */
for(counter=0; counter<classes; counter++) freq[counter]=0;
xmin = data_array[0];
xmax = xmin;
for(counter=1; counter<number; counter++){
    if(data_array[counter] < xmin) xmin=data_array[counter];
    if(data_array[counter] > xmax) xmax=data_array[counter];} /*End for */
if((xmax-xmin) < (eta*classes)){
    printf("\nAll data values are equal!\n");
    textmode(3);
    clrscr();
    return;} /* End if */

/* Determine scale */
key = 1;
k= 0;
step = 0.0;

```

```

do{
  range = xmax - xmin;
  temp = xmin;
  do{
    flag = 0;
    while(range <= 1.0){
      k++;
      range *= 10.0;} /* End while */
    while(range > 10.0){
      k--;
      range /= 10.0;} /* End while */
    if(key <= 2){
      tk = pow10(k);
      temp *= tk;
      int_temp = temp;
      if(temp < 0.0 && int_temp != temp) temp -= 1.0;
      int_temp = temp;
      temp = int_temp/tk;
      range = (xmax - temp) / classes;
      k = 0;
      key += 2;
      flag = 1;} /* End if */
  }while(flag); /* End do */
  int_step = range;
  step = int_step; /* Get the integer part of range */
  if (step != range)step += 1.0;
  if (range < 1.5)step -= 0.5;
  step /= pow10(k);
  if (key != 4){
    k=1;
    key=2;}
  }while(key != 4 && (xmax -xmin <= (0.8*classes*step))); /* End do */
  xmin = temp;
  int_step = (temp / step);
  c = step * int_step;
  if ( c < 0.0 && c != range) c -= step;
  if (c+classes*step >= xmax)xmin = c;

  /* Calculate frequencies for each interval */
  for(counter=0;counter<number;counter++){
    int_step = (data_array[counter]-xmin)/step;
    freq[int_step] += 1;} /* End for */

  /* Print frequency vector */
  fprintf(output_stream,"Frequency ");
  for(counter=0;counter<classes;counter++)
    fprintf(output_stream,"%5d",freq[counter]);
  fprintf(output_stream,"\n");

```

```

    for(counter=0;counter<(classes*5+15);counter++)
        fprintf(output_stream,"%c",a_dash);
    fprintf(output_stream,"\n");

/* Find largest frequency and scale */
int_step = 0;
for(counter=0;counter<classes;counter++)
    if(freq[counter] > int_step)int_step = freq[counter];
scale = 1;
div = 1.0;
if(int_step > leng){
    scale = (int_step + leng - 1)/leng;
fprintf(output_stream,"\n      Each %c equals %d
points.\n\n",a_star,scale);
    div = 1.0/(float)scale;} /* End if */

/* Clear output array */
for(counter=0;counter<classes;counter++)iout[counter]=a_blank;

/* For line of print fill output array */
int_step = (float) int_step * div +.5;
for(counter=0;counter<int_step;counter++){
    k = int_step - counter;
    for(counter2=0;counter2<classes;counter2++){
        index = (float)freq[counter2] * div + .5;
        if(index == k)iout[counter2]=a_star;} /* End for */
    l = k*scale;
    fprintf(output_stream,"%8d  ",l);
    for(counter2=0;counter2<classes;counter2++)
        fprintf(output_stream,"  %c",iout[counter2]);
    fprintf(output_stream,"\n");} /* End for */
for(counter=0;counter<(classes*5+15);counter++)
    fprintf(output_stream,"%c",a_dash);
fprintf(output_stream,"\n");

/* Compute Interval mid-points and scale if necessary */
k=0;
xmin=xmin+step*.5;
xmax = xmin + step * (float) (classes -1);
xm = fabs(xmin);
if (xm > fabs(xmax))xm=fabs(xmax);
if(xm < eta)xm=xm+step;
while(xm<0.1){
    k += 1;
    xm *= 10.0;} /* End while */
xm=xmax;
if(xm < -xmin)xm=-xmin;
while(xm > 1000.0){

```

```

    k -= 1;
    xm /= 10.0;} /* End while */
tk = pow10(k);
step = step*tk;
out[0]=xmin*tk;
for(counter=1;counter<classes;counter++)
out[counter]=out[counter-1]+step;

/* Print out interval mid-points */
fprintf(output_stream," Interval  ");
for(counter=0;counter<classes;counter+=2)
    fprintf(output_stream,"%8.3f  ",out[counter]);
fprintf(output_stream,"\n Mid-points  ");
for(counter=1;counter<classes;counter+=2)
    fprintf(output_stream,"%8.3f  ",out[counter]);
fprintf(output_stream,"\n");
k = -k;
if(k != 0)
    fprintf(output_stream,"The printed values must be multiplied by
10**%d.",k);
return;
}

/* *****
Function: int_query

Description: This function asks for an integer input and
checks if it is in the specified range.

Arguments: Question string, lower bound, upper bound.

Returns: Returns integer value.
***** */
int int_query(str,lb,ub)

char *str;
int lb;
int ub;

{
int input_int;
int x;
int y;
void my_pause();

x=wherex();
y=wherey();
do

```

```

{
gotoxy(x,y);
printf("\n%s [%d...%d]:",str,lb,ub);
scanf("%d",&input_int);
if(input_int < lb || input_int > ub)
    {
    printf("\n\t\t%d is not in the range [%d...%d]",input_int,lb,ub);
    my_pause();
    gotoxy(x,y+3);
    clreol();
    gotoxy(x,y+4);
    clreol();
    gotoxy(x,y+1);
    clreol();
    }
}while(input_int < lb || input_int > ub);
return input_int;
}

```

```

/* *****
Function: keyboard_input

```

Description: This function asks for the data to be input by the user directly from the keyboard.

Arguements: the number of data points and the pointer to the array to hold the data.

Returns: None.

```

***** */
void keyboard_input(number,n_array)

float n_array[];          /* Data array */
int number;              /* Number of data points */
{
    int answer;          /* Holds result of querries */
    int answer2;        /* Holds result of querries */
    int correct;        /* The number of the value to be corrected */
    int counter;        /* Counter to enter data */

    printf("Enter data by typing number and pressing the return key\n\n");
    for(counter=0;counter<number;counter++){
        printf("Enter value %d:",counter+1);
        scanf("%f",n_array+counter);} /* End for */

    /* View and Edit data */
    answer = char_query("\nDo you want to view data for mistakes");
    if(answer){

```

```

clrscr();
printf("The input data is:\n");
counter=0;
while(counter!=number){
    printf("%d\t\t%-6.4f\n",counter+1,*(n_array+counter));
    if(!((counter+1) % 20)||((counter==(number-1)))){
        answer2 = char_query("\nDo you want to correct a data point");
        if(answer2){
            correct=int_query("Which data point",1,number);
            printf("Enter new value for data point %d: ",correct--);
            scanf("%f",n_array+correct);
            if(!((counter+1)%20))counter -= 20;
            else counter=number-(number %20)-1;} /* End if */
            printf("\n\n\n\n");} /* End if */
        counter++;}} /* End while and if */
return;
}

```

```

/* *****
Function: myopen

```

Description: This function asks for the data file name, opens the file and returns a pointer to the file stream. If there is an error it will return the NULL pointer.

Arguements: iostatus tells whether it is for input or output.

Returns: Returns a pointer to the file stream.

```

***** */

```

```

FILE *myopen(iostatus) /*Will return a pointer to the open file */
int iostatus;
{
    char buffer[MAXPATH];
    char file_name[20]; /*variable for file name */
    FILE *fp; /*File pointer*/
    int answer;
    void my_pause();

    getcwd(buffer, MAXPATH);
    printf("Enter the name of file (include the directory).\n");
    printf("The default directory is %s: ",buffer);
    scanf("%s",file_name);

    /*open file */
    switch (iostatus){
        case 0: /* Input */
            if ((fp=fopen(file_name,"r")) == NULL){

```

```

        fprintf(stderr,"Could not open %s.",file_name);
        my_pause();
        return NULL; /* If file could not be opened return the null
pointer*/
    } /* End if */
    return fp;

    case 1:          /* Writing or output */
    if ((fp=fopen(file_name,"r")) != NULL){
        fprintf(stderr,"%s already exists!\n",file_name);
        answer = char_query("\nDo you want to write over it");
        if(!answer){
            printf("\nPlease try again.\n");
            my_pause();
            return NULL;} /* End if */
        } /* End if */
    if ((fp=fopen(file_name,"w")) == NULL){
        fprintf(stderr,"Could not open %s.\nPlease try
again.\n",file_name);
        my_pause();
        return NULL; /* If file could not be opened return the null
pointer*/
    } /* End if */
    return fp;
} /* End switch */
return NULL;
}

```

```

/* *****
Function: my_pause

```

Description: This function holds the screen.

Arguements: None.

Returns: None.

```

***** */

```

```

void my_pause(void)
{
printf("\nPress any key to continue");
getch();
return;
}

```

```
/* *****
```

Function: percent_done

Description: This function tells how much of a long iteration is complete.

Arguments: Integer counter and integer for total iteration number.

Returns: None.

```
***** */
```

```
void percent_done(counter,total)
int counter;
int total;
{
    int y;
    int result;
    long midresult;

    y = wherey();
    gotoxy(1,y);
    midresult = (float)counter*100.0;
    result = midresult/total;
    printf("%d percent complete",result);
    return;
}
```

```
/* *****
```

Function: read_input

Description: This function reads the data from the file stream and stores it in an array.

Arguments: Pointer to file steam, pointer to float array for data.

Returns: Returns an integer for the number of data points read.

```
***** */
```

```
int read_input(fp,n_array)    /* Returns the number of data points read
*/
FILE *fp;                    /* fp is the pointer to the file stream */
float n_array[];             /* Data array */
{
    int answer;
    int counter=0;           /* Counter for how many values */
```

```
/* Since !feof executes one time too many want to
   print first then read data so must have one initial
```

```

    read line also ask if data should be viewed */

answer = char_query("\nDo you want to view input data");
if(answer){
    clrscr();
    printf("The input data is:\n");} /* End if */

/*Get first data point */

fscanf(fp,"%f",n_array+counter);

/* Read input until the end of file is reached */
while (!feof(fp)){
    if(answer){
        printf("%-6.4f\n",*(n_array+counter));
        if(!((counter+1) % 20)){
            my_pause();
            printf("\n\n\n\n");}} /* End ifs */
        counter++;
        fscanf(fp,"%f",n_array+counter);} /* End while */

/* Close the file and reset feof*/
printf("\nData has been entered.");
my_pause();
clearerr(fp);
fclose(fp);
return (counter); /* Since array starts at zero no need to decrement */
}

/* *****
Function: write_out

Description: This function writes the data to the file stream.

Arguements: Pointer to file steam, pointer to float array for data,
and an integer for the number of values to print.

Returns: Returns a void.
***** */

void write_out(fout,n_array,num)

FILE *fout; /* fout is the pointer to the file stream */
float n_array[]; /* Data array */
int num; /* number of data points in Data array */

{
    char str[40];

```

```
int answer;
int counter;
struct date d;      /* For date in header line */

/* Check to see if user wants a header in file */
answer = char_query("\nDo you want a header line in your output
file");
if(answer){
    getdate(&d);
    fprintf(fout,"The date this data was created was
%d/%d/%d\n",d.da_mon,\
d.da_day,d.da_year);
    printf("Enter the header you want printed:\n");
    fflush(stdin);
    while((fgets(str,40,stdin)) == NULL){
        printf("Please type header again, do not use more than 40
charaters");
        my_pause();} /* End while */
    fprintf(fout,"%s",str);} /* End if */

/* Write Data */
for(counter=0;counter<num;counter++)
    fprintf(fout,"%-6.4f\n",n_array[counter]);

return;
}
```

File: main.c

```

/* *****
Copyright 1993

This file is the main program

Brad Warner      Revision 0      Feb. 23, 1993
***** */

# include <stdio.h>
# include <stddef.h>
# include <bios.h>
# include <conio.h>
# include "allus.h"

# define MAIN_MENU 1
# define INPUT_MENU 2
# define BOOT_MENU 3
# define STAT_MENU 4
# define NO 0
# define YES 1

float con_level;      /* The confidence level for confidence intervals */
float lower_cpn;      /* Upper value for Cpn calculations */
float upper_cpn;      /* Lower value for Cpn calculations */
int BOOT_FLAG=0; /* Which statistic did we use global because used
                  in boot.c */

main ()
{
    FILE *fp;          /* Pointer to input file */
    FILE *fpout;       /* Pointer to output data file */
    float b_array[DIM2]; /* Bootstrap distribution */
    float ci_limits[2]; /* Confidence interval */
    float n_array[DIM1]; /* Input data */
    float expected;
    float re_boot[DIM1]; /* The distribution of the std err */
    float re_stderr;
    float std_err;
    int answer;
    int b;
    int ci_flag=0;
    int cont;
    int flag_bootexpect=0; /* Did you calculate expected value */
    int flag_bootstderr=0; /* Did you calculate std err */
    int flag_hist=0;      /* Should ask for histogram */

```

```

int flag_input = 0;
int flag_menu = INPUT_MENU;      /* Tells which menu currently using */
int flag_reboot = 0;             /* Did you reboot */
int dontwrite;                   /* Allows an escape from write menu */
int number;                       /* Number of input data points */
int reb;                           /* Number of reboot replications */
int status;                       /* Check printer status */
static char *stat[5] = {"Nothing","mean","variance","median","Cpk"};
/* Start Program with credit */
    text_mode(3);
    credit_menu();

/* Menu System */
while(flag_menu){
    switch(flag_menu){
        case MAIN_MENU:
            answer = init_menu();
            switch(answer){
                case 1:
                    flag_menu = INPUT_MENU;
                    break;

                case 2:
                    begin_screen();
                    dontwrite = 0;
                    while ((fpout = myopen(1)) == NULL) /*Keep asking for file*/
                    {
                        cont=char_query("\nDo you to exit to the Main Menu");
                        if(cont){
                            dontwrite = 1;
                            break;} /* End if */
                        } /* End while */
                        if(dontwrite)break;
                        begin_screen();
                    cont=char_query("\nDo you want to write input data to your file");
                    if(cont)write_out(fpout,n_array,number);
                    cont=char_query("\nDo you want to write histogram of input data to your
file");
                        if(cont)hist(n_array,number,fpout);
                    cont=char_query("\nDo you want your bootstrap distribution printed to
your file");
                        if(cont && !BOOT_FLAG){
                            printf("\nYou have not executed a bootstrap!");
                            break;} /* End if */
                        if(cont && BOOT_FLAG){
                            write_out(fpout,b_array,b);
                    cont=char_query("\nDo you want to write a histogram of the bootstrap
data to your file");

```

```

        if(cont)hist(b_array,b,fpout);} /* End if */
        if(flag_reboot){
cont=char_querry("\nDo you want your reboot distribution printed to your
file");
        if(cont)write_out(fpout,re_boot,reb);} /* End if */
cont=char_querry("\nDo you want a summary printed to your file");
        if(cont && BOOT_FLAG){
            fprintf(fpout,"A bootstrap simulation of the %s was
carried out.\n \
            The input data had %d points.\n \
            There were %d bootstrap replications.\n", \
stat[BOOT_FLAG],number,b);
            if(flag_bootexpect)
                fprintf(fpout,"The expected value of the bootstrap
distribution \
of the %s was %-6.4f.\n",stat[BOOT_FLAG],expected);
            if(flag_bootstderr)
                fprintf(fpout,"The standard error of the bootstrap
distribution \
of the %s was %-6.4f.\n",stat[BOOT_FLAG],std_err);
            if(flag_reboot)
                fprintf(fpout,"The standard error of the standard error
distribution \
of the %s was %-6.4f.\n",stat[BOOT_FLAG],re_stderr);
            if(ci_flag)
                fprintf(fpout,"\n\t\t\tThe %.2f confidence interval is:
\n\n \
\t\t\t%-6.4f < %s < %-
6.4f",con_level,ci_limits[0],stat[BOOT_FLAG],ci_limits[1]);
        } /* End if */
        break;

    case 3:
        if(!flag_input){
            printf("\nYou have no data to execute a bootstrap!");
            my_pause();
            break;} /* End if */
        flag_menu=BOOT_MENU;
        break;

    case 4:
        close(fpout);
        textmode(2);
        clrscr();
        return;} /* End switch */
break;

case INPUT_MENU:

```

```

answer = input_menu();
switch(answer){
  case 1:
    begin_screen();
    number = int_query("How many data points",1,1000);
    keyboard_input(number,n_array);
    flag_input=YES;
    flag_hist=YES;
    flag_menu = MAIN_MENU;
    break;
  case 2:
    begin_screen();
    while ((fp = myopen(0)) == NULL) /* Keep asking for file */
    {begin_screen();
      cont=char_query("\nDo you want to exit to the Main
Menu");

      if(cont){
        flag_menu = MAIN_MENU;
        flag_hist = NO;
        break;} /* End if */
    } /* End while */
    if(flag_menu == MAIN_MENU)break;
    number = read_input(fp,n_array);
    printf("\n\nThere are %d values in the array.",number);
    my_pause();
    flag_menu = MAIN_MENU;
    flag_input = YES;
    flag_bootexpect = NO;
    flag_bootstderr= NO;
    ci_flag=NO;
    flag_reboot = NO;
    flag_hist=YES;
    break;

  case 3:
    flag_menu = MAIN_MENU;
    flag_hist=NO;
    break;
} /* End switch */
flag_bootexpect = 0;
flag_bootstderr= 0;
ci_flag=0;
flag_reboot = 0;
/* Check if there is input data */
if(flag_hist){
  begin_screen();
  cont=char_query("\nDo you want to display a histogram of your
input");

```

```

    if(cont){
    hist(n_array,number,stdout);
    my_pause();
    text_mode(3);
    clrscr();}} /* End ifs */
    break;

case BOOT_MENU:
    answer = boot_menu();
    switch(answer){
    case 1:
    case 2:
    case 3:
    case 4:
        if(answer == 4){
            begin_screen();
            printf("\n\n\nWhat is the upper specification value? ");
            scanf("%f",&upper_cpn);
            printf("\n\n\nWhat is the lower specification value? ");
            scanf("%f",&lower_cpn);} /* End if */
            b = int_query("\nHow many bootstrap replications?",2,10000);
            BOOT_FLAG = answer;
            boot(number,b,n_array,b_array,YES);
            flag_menu = STAT_MENU;
            break;

        case 5:
            flag_menu = MAIN_MENU;
            break;
    } /* End switch */
    break;

case STAT_MENU:
    answer = stat_menu();
    switch(answer){
    case 1:
        expected = mean(b_array,b);
        flag_bootexpect=1;
        begin_screen();
        printf("\n\nThe expected value of the bootstrap distribution is:
\n
%-6.4f",expected);
        my_pause();
        break;

    case 2:
        std_err = boot_stderr(b_array,b);
        flag_bootstderr=1;

```

```

        begin_screen();
        printf("\nThe standard error of the bootstrap distribution is:
\
%-6.4f",std_err);
        my_pause();
        break;

/* CI */ case 3:
        begin_screen();
con_level=flt_query("\n\nWhat is the confidence level (1 -
alpha)",0.0,1.0);
        ci(ci_limits,b_array,b,n_array,number);
        ci_flag=1;
        begin_screen();
        printf("\n\t\t\t\tThe %.2f confidence interval is: \n\n \
\t\t\t\t%-6.4f < %s < %-
6.4f\n",con_level,ci_limits[0],stat[BOOT_FLAG],ci_limits[1]);
        my_pause();
        break;

        case 4:
            reb = int_query("This is a long procedure.\n \
Enter the number of reboot replications",1,1000);
            re_stderr=reboot(n_array,b_array,re_boot,number,b,reb);
            begin_screen();
            printf("\nThe standard error of the standard error of the %s
is: \
%-6.4f",stat[BOOT_FLAG],re_stderr);
            my_pause();
            flag_menu = MAIN_MENU;
            flag_reboot = 1;
            break;

        case 5:
            hist(b_array,b,stdout);
            my_pause();
            text_mode(3);
            clrscr();
            break;

        case 6:
            flag_menu = BOOT_MENU;
            break;
    } /* End switch */
break;
} /* End switch */
} /* End while */
return;}

```

File: menu.c

```
/* *****
Copyright 1993
```

This file contains functions for menu and user interface

```
begin_screen
boot_menu
credit_menu
help_menu
init_menu
input_menu
menu
stat_menu
text_mode
```

```
Brad Warner      Revision 0      Feb. 23, 1993
```

```
***** */
```

```
# include <stdio.h>
# include <conio.h>
# include <stddef.h>
```

```
extern int_query();
```

```
/* *****
Function: begin_screen
```

Description: Set background screen

Arguements: None

Return: void

```
***** */
```

```
void begin_screen(void)
{

clrscr();
printf("\t\t\t BOOTSTRAP SIMULATION\n\n");
return;
}
```

```
/* *****
```

```
Function: boot_menu
```

```
Description: Sets up bootstrap menu
```

```
Arguements: None.
```

```
Returns: integer for choice desired
```

```
***** */
```

```
int boot_menu(void)
```

```
{
```

```
static char *str[6] = { "Distribution for", "Sample Mean",
                        "Variance", "Median", "Process Control Ratio (Cpk)",
                        "Return to Main Menu"};
```

```
int menu();
```

```
int num=5;
```

```
int answer;
```

```
void help_menu();
```

```
answer = menu(str,num);
```

```
while(answer == 0){
```

```
    help_menu(3);
```

```
    answer = menu(str,num);} /* End while */
```

```
return answer;
```

```
}
```

```
/* *****
```

```
Function: credit_menu
```

```
Description: Sets up credit display
```

```
Arguements: None.
```

```
Returns: None.
```

```
***** */
```

```
void credit_menu(void)
```

```
{
```

```
begin_screen();
```

```
printf("\n\n\n\n\n\n\n\n\n\n\n\n\n\n\n");
```

```
printf("\t\t\t\t\tDeveloped by\n\t\t\t\t\tBrad Warner");
```

```
printf("\n\n\t\t\t\t\tColorado School of Mines");
```

```
printf("\n\n\t\t\t\t\tMarch 15, 1993");
```

```
printf("\n\n\n\n\n\n\t\t\t\t\tPress any key to continue");
```

```
getch();
```

```
return;
```

```
}
```

```
/* *****
```

```
Function: help_menu
```

```
Description: Sets up help screens
```

```
Arguements: Menu help is called from.
```

```
Returns: Void
```

```
***** */
```

```
void help_menu(number)
int number;
{
FILE *fhlp;
int counter;
switch (number){
case 1:
fhlp=fopen("help1.hlp","r");
if(fhlp == NULL){
clrscr();
printf("\n\n\t\t\tHelp menu is not available\n\n");
printf("\n\n\n\t\t\tPress any key to continue");
getch();
break;} /* End if */
while(!feof(fhlp)){
clrscr();
printf("\t\t\t HELP SCREEN\n\n");
counter=0;
while(counter<18 & !feof(fhlp)){
if(putc(getc(fhlp),stdout)=='\n')counter++;} /* End while and if
*/
printf("\n\n\n\t\t\tPress any key to continue");
getch();} /* End while */
break;
case 2:
fhlp=fopen("help2.hlp","r");
if(fhlp == NULL){
clrscr();
printf("\n\n\t\t\tHelp menu is not available\n\n");
printf("\n\n\n\t\t\tPress any key to continue");
getch();
break;} /* End if */
while(!feof(fhlp)){
clrscr();
printf("\t\t\t HELP SCREEN\n\n");
counter=0;
while(counter<18 & !feof(fhlp)){
if(putc(getc(fhlp),stdout)=='\n')counter++;}/* End while and if */
```

```

    printf("\n\n\n\t\t\tPress any key to continue");
    getch();} /* End while */
break;
case 3:
fhlp=fopen("help3.hlp","r");
if(fhlp == NULL){
    clrscr();
    printf("\n\n\t\t\tHelp menu is not available\n\n");
    printf("\n\n\n\t\t\tPress any key to continue");
    getch();
    break;} /* End if */
while(!feof(fhlp)){
    clrscr();
    printf("\t\t\t HELP SCREEN\n\n");
    counter=0;
    while(counter<18 & !feof(fhlp)){
        if(putc(getc(fhlp),stdout)=='\n')counter++;} /*End while and if */
    printf("\n\n\n\t\t\tPress any key to continue");
    getch();} /* End while */
break;
case 4:
fhlp=fopen("help4.hlp","r");
if(fhlp == NULL){
    clrscr();
    printf("\n\n\t\t\tHelp menu is not available\n\n");
    printf("\n\n\n\t\t\tPress any key to continue");
    getch();
    break;} /* End if */
while(!feof(fhlp)){
    clrscr();
    printf("\t\t\t HELP SCREEN\n\n");
    counter=0;
    while(counter<18 & !feof(fhlp)){
        if(putc(getc(fhlp),stdout)=='\n')counter++;} /*End while and if */
    printf("\n\n\n\t\t\tPress any key to continue");
    getch();} /* End while */
break;} /* End switch */
return;
}
/* *****
Function:  init_menu

Description: Sets up initial menu

Arguements: None.

Returns: integer for choice desired
***** */

```

```

int init_menu(void)
{
static char *str[5] = { "Main Menu", "Input data","Write to file",
                        "Perform Bootstrap", "Exit Program"};

int menu();
int num=4;
int answer;

answer = menu(str,num);
while(answer == 0){
    help_menu(1);
    answer = menu(str,num);} /* End while */
return answer;
}

/* *****
Function:  input_menu

Description: Sets up input menu

Arguements: None.

Returns: integer for choice desired
***** */
int input_menu(void)
{
static char *str[4]={"Data Input","Input from keyboard","Import data
file",
                    "Return to Main Menu"};

int menu();
int num=3;
int answer;

answer = menu(str,num);
while(answer == 0){
    help_menu(2);
    answer = menu(str,num);} /* End while */
return answer;
}

```

```
/* *****
```

```
Function: menu
```

```
Description: Sets up a menu
```

```
Arguements: Character strings for questions, integer for number of
questions
```

```
Returns: integer for choice desired
```

```
***** */
```

```
int menu(str,num)
```

```
char **str;
int num;
```

```
{
int answer;
int counter;
```

```
begin_screen();
printf("\n\n\t\t\t %s",*(str));
printf("\n\n");
printf("\t\t\t0: Help Screen");
for(counter=1;counter<=num;counter++)
{
printf("\n\n");
printf("\t\t\t%d: %s",counter,*(str+counter));
}
printf("\n\n");
answer=int_query("\t\tInput your choice:",0,num);
return answer;
}
```

```
/* *****
```

```
Function: stat_menu
```

```
Description: Selects sample statistic
```

```
Arguements: None.
```

```
Returns: integer for choice desired
```

```
***** */
```

```
int stat_menu(void)
```

```
{
static char *str[7] = { "Statistic from the Distribution", "Expected
Value",
"Standard Error","Confidence Interval",
"Reboot","Histogram","Return to Previous Menu"};
```

```
int num=6;
int answer;

answer = menu(str,num);
while(answer == 0){
    help_menu(4);
    answer = menu(str,num);} /* End while */
return answer;
}

/* *****
Function: Text mode

Description: Sets the text mode

Arguements: Integer for text mode

Returns: Void

***** */
void text_mode(mode)

int mode;

{
textmode(mode);
textbackground(BLUE);
textcolor(WHITE);
clrscr();
return;
}
```

File: help1.hlp

The bootstrap is a non-parametric procedure to estimate standard error and confidence intervals. This program provides a computer based simulation of the bootstrap.

The first step is to input data. Data can be entered directly from the keyboard or imported as ASCII file. Data can be entered by selecting option 1) from the current menu.

Option 2) will allow you to write output to a file of your choice. This step can be performed at any time after data has been entered. BEWARE that this step will write over the file if the name of the file you give already exists. The program will take you through a number of options. First you will be allowed to write your input data out. Then you can write your bootstrap data to the output file. For both of these options you can write the histogram of the data and a header. The header is one line of whatever identification data you desire. The summary option will print the number of data points, bootstrap replications, and the bootstrap results you selected. The output file generated is in ASCII format. Once data has been entered, a bootstrap can be performed. To print the file you must exit the program and print it from DOS or some text editor.

Option 3) from the current menu will produce the bootstrap menu. The bootstrap menu has a list of statistics, for example sample mean. The bootstrap will generate an estimate of the distribution of the statistic selected by using the input data. The number of bootstrap replications determines how many data points will be in the distribution. The larger the number of replications the longer the procedure takes. When the bootstrap is complete, you are taken to a menu which allows you to extract information such as standard error and confidence intervals.

Option 4) closes all files and returns you to your operating system.

File: help2.hlp

There two ways to enter data into the bootstrap program. Option 1) allows you to enter and edit the data directly from the keyboard. The information is first entered and then the user has the option of viewing and editing the data. The data is entered by typing the value and hitting return. Numbers are printed next to each value during viewing to act as a tag. This allows you to identify the value you wish to correct.

Option 2) has the program read the information from an ASCII format file. The data in the file must be seperated by either a blank or a return. The program looks for the file in the default directory, that is the directory the program was executed from. If the file is in a directory other than the default directory include the path in the file name. For example, if you started the program from C:\ but your file, thefile.dat, is in C:\data\, then when the program asks for the file name enter it as C:\data\thefile.dat.

After the data has been entered you have the option of viewing a histogram of the data. The only option for the histogram is the number of groups to break the data into. You are limited between 2 and 10. Read the graph carefully because each * may represent more than one point and the mid-point value may also be scaled. The numbers at the top of each column represent the total number of values in that group. If you want to print the histogram you must go to the Main Menu and write the histogram to the output file.

File: help3.hlp

This menu lists the statistic available for performing the bootstrap. The bootstrap will produce an estimate of the distribution of the statistic. The number of data points in the distribution equals the number of bootstrap replications. The larger the number of replications the longer the procedure will take.

The current choices are:

Mean: Defined as the sum of the values divided by the total number of values.

Variance: Defined as the sum of the square of the values minus the mean all divided by the total number of values minus one.

Median: Defined as having an equal number of values above and below it. If n is the total number of values and is odd then the median is the value with $(n-1)/2$ values less than it and greater than it. If n is even then the median is the $(n/2)+1$ value.

Cpk: Is used in statistically process control. It is used to determine the how stable a process is. It is defined as the minimum of the mean minus a lower specification all divided by three times the standard deviation or the upper specification minus the mean all divided by three times the standard deviation.

File: help4.hlp

This menu allows you to manipulate the bootstrap distribution of the statistic selected on the bootstrap menu. The options are:

Expected value: This is the mean of the bootstrap distribution. This is an estimate of the expected value of the statistic from the bootstrap menu.

Standard Error: This is the bootstrap estimate of the standard error of the statistic from the bootstrap menu. It is a measure of the error in the statistic from the data you entered.

Confidence Interval: This option lets you generate a bootstrap confidence interval. You can enter the confidence level and the program will return a two-sided limit.

Reboot: This option generates an estimate of the standard error of the standard error of the statistic selected from the bootstrap menu. This allows you to get an understanding of the stability of your standard error estimate. This is important if you want to reduce the number of bootstrap replications. The bootstrap estimate will be most stable if you use the entire 10000 replications but this can be slow. The reboot procedure is extremely slow. It is recommended that you perform this only if you are generating repeated bootstrap estimates for the same statistic of interest with approximately the same sample sizes.

Histogram: This generates a histogram of the bootstrap distribution. This distribution is an estimate of the statistic's distribution selected in the bootstrap menu. You are given the choice of how many groups to break the data into, a number between 2 and 10. If you want a printout of the histogram, this can be done from the Main Menu. Read the graph carefully because the * can represent more than one value and the mid-point value could also be scaled. The numbers at the top of each column is the total number of values in that group. The numbers on the side are the frequency of the values in that column.