COMPARISON OF SIMULATION TECHNIQUES BASED ON MILK YIELD DATA

A Linear Mixed Model Approach

By

Yabebal Ayalew

ADDIS ABABA UNIVERSITY

OFFICE OF GRADUATE STUDIES

DEPARTMENT OF STATISTICS

Addis Ababa

June 2010

COMPARISON OF SIMULATION TECHNIQUES BASED ON MILK YIELD DATA

A Linear Mixed Model Approach

By

Yabebal Ayalew

A thesis submitted to the office of graduate programs of Addis Ababa University in partial fulfillment of the requirement for the degree of Master of Science in Statistics (Biostatistics).

June 2010

Addis Ababa

Research Title: Comparison of Simulation Techniques Based on Milk Yield Data: A Linear Mixed Model Approach

Name of Candidate: Yabebal Ayalew

College: Natural Sciences

Department: Statistics

Approved by the board of examiners:

	Advisor	Examiner	Examiner
Name:	Dr. M.K. Sharma	Prof. Eshetu Woncheko	Dr. Girma Taye
Sig:	_____	_____	_____
Date:	_____	_____	_____

Acknowledgements

This study would never be completed without the contribution of many people to whom I would like to express my thanks. My special gratitude goes to my advisor, Dr. M.K. Sharma for his encouragement, guidance and support from the initial to the final level which enabled me to develop an understanding of the subject. I owe my deepest gratitude to Dr. Aynalem Haile, Animal scientist, ILRI for his detailed and constructive comments, and for his important support throughout this work.

I am indebted to Meseret Molla of Gondar University, Defaru Debebe of Arba Minch University, Taddele Cherenet of Addis Ababa University, Wolansa Mengesha and Meron Demissie, secretaries of statistics department, AAU. Despite their busy schedule, they were always willing to discuss my work.

Last but not the least my appreciation goes to my beloved mother Aregie Getu, to whom I dedicate this thesis, for her endless support.

Table of Content

Acronyms and Symbols

AIC	Akaike Information Criteria
Ber	Bernoulli distribution
BLUE	Best Linear Unbiased Estimator
BLUP	Best Linear Unbiased Predictor
CDF	Cumulative Distribution Function
Den DF	Denominator Degree of Freedom
DF	Degree of Freedom
EIAR	Ethiopian Institute of Agricultural Research
Gen.Pareto	Generalized Pareto distribution
HF	Holstein Friesian
iid	Independently Identically Distributed
ILRI	International Livestock Research Institute
JSB	Johnson Bounded distribution
LL_{3p}	Log-Logistic with three parameters
LMM	Linear Mixed Model
ML	Maximum Likelihood
MLE	Maximum Likelihood Estimator
N	Normal distribution
Num DF	Numerator Degree of Freedom
Pdf	Probability Density Function
$PT6_{4p}$	Pearson type VI distribution
REML	Restricted Maximum Likelihood
SAS	Statistical Analysis Software

U(0, 1)	Uniform distribution with parameter 0 and 1
UNRRA	United Nations Relief and Rehabilitation Administration
WWII	World War II
~	distributed
	No
	Yes
	Flow direction
	Process
	Calling sub function
	Start or end
	Output
	Condition checking
	Connector

Abstract

The term simulation has been used in a number of scientific disciplines. Agriculture is one of scientific disciplines in which simulation technique has been used to investigate theoretical as well as practical problems. Simulation has a capability of producing relevant answers based on incomplete and small datasets. In Ethiopia, researchers have often been challenged by such type of data. As a result, this study is aimed to compare which computer based simulation techniques to approximate the results of the previously accomplished researches. We have obtained 15 years of data from Debre Zeit Research Station of the International Livestock Research Institute and Holetta Agricultural Research Centre of the Ethiopian Institute of Agricultural Research for this study. We compared the two most familiar simulation techniques namely Monte Carlo and bootstrap simulations by using the results of linear mixed model fitted for each dataset. We found that both Monte Carlo and bootstrap simulations can approximate the farm and genetic group effects equally. Lactation length and daily milk yield are found to be significant ($P<0.0001$) in both simulation techniques. Unlike for bootstrap simulation, season and period of calving are found to be significant for Monte Carlo simulation. On the basis of the findings, this study reached a conclusion that Monte Carlo simulation has a better approximation.

Key words: Monte Carlo, bootstrap, simulation, model, linear mixed model, milk yield.

1 Introduction

1.1 Background of the study

Simulation is essentially a controlled statistical sampling technique (experiment) that is used in conjunction with a model, to obtain approximate answers for questions about complex, multifactor probabilistic problems (Lewis and Orav 1989).

Simulation has long been an important tool of designers, whether they are simulating a supersonic jet flight, a telephone communication system, a wind tunnel, a large-scale military battle (to evaluate defensive or offensive weapon systems), or a maintenance operation (to determine the optimal size of repair crews).

Although simulation is often viewed as a *method of last resort* to be employed when everything else has failed, recent advances in simulation methodologies, availability of software, and technical developments have made simulation one of the most widely used and accepted tools in system analysis and operation research. That is why more than 3000 articles on simulation have been published until 1981 (Rubinstein 1981).

Ethiopia, after getting the first batch of dairy cattle through the United Nations Relief and Rehabilitation Administration, UNRRA, under Marshall Plan[1], has organized seven modern dairy research centers which are distributed in four agro-ecological zones[2] (Alemu *et al.* 1998; Ofcansky and Berry 1991; Hizkias 1998). Even though Ethiopia has large livestock population

[1] The Marshall Plan is called the European Recovery Plan. It was enacted by the United States in 1974 as a way to help rebuild Europe after WWII. The genius behind the plan was Gorge Marshall, who was at the time the US secretary of state. Thought part of the Marshall Plan was meant to help the badly damaged Europe recover from WWII, the other part of the Marshall plan was meant to prevent communism from gaining strong hold in war torn countries.

[2] The four agro-ecological zones are high land, sub-humid, semi-arid & arid and mid altitude zones.

in the high land region, it has not been getting the desired profit from them (Mukasa-Mugerwa *et al.* 1989).

The researchers have been continuously trying to find out factors which can maximize the profit of the country by there researches. But due to non-availability and shortage of high quality data, it was not possible to achieve the objectives.

1.2 Statement of the Problem

The estimated Ethiopian population growth is about 3.208% and the estimated milk yield is about 1.2 million tones per annum. The annual milk demand growth is at a rate of 2.5%. The contribution of different livestock species to the milk supply is about 68% from cattle, 18% from camels, 8% from goats and 6% from sheep (Alemu 1998).

The annual per capita consumption of milk is 19 kg. This value is lower than African and world per capita averages, which are 24kg and 1000kg per year, respectively (Saxena *et al.* 1997). This clearly indicates the status of the country's dairy production development. In other words, there is a wide gap between current supply and demand for milk in the country.

In order to meet the ever increasing demand of milk and milk products, continual research must be done on those factors which affect lactation and milk yield such as breed type, number of parity, season of calving, geographic region, farm management factors (nutrition, frequency of milking) age and body weight at calving, season of calving and so forth (Wood 1969; Danell 1982; Wilmink 1987). One of the problems that prevent researchers for conducting series research is that there is not enough and complete data and database system in the country regarding milk yield. The data which are available and recorded in different research centers are not that much satisfactory. Therefore, we must find alternative statistical methods which can use these incomplete datasets[3] efficiently and reach at valid conclusion.

Simulation is one of the methods which can use a small and incomplete data effectively and efficiently. Even though simulation techniques use small and incomplete dataset effectively and

[3] Incomplete dataset is a dataset which doesn't contain the full records of all cows in the study. For instance, if a record of a cow is begin at 11th parity and there is no other parities' record about this cow, then the data is said to be incomplete. We must get all records (history) of a cow from the first up to the last parity.

efficiently, all techniques don't have the same power of reaching at valid or sound conclusion. Moreover, to the best of the researcher's knowledge, there has been no research on simulation that is applied on milk in Ethiopia.

1.3 Objective of the study

The general objective of this study is to compare which computer based simulation technique is better approximate the results of previously accomplished researches regarding milk production traits.

Moreover, this paper has the following specific objectives:

- Comparisons of fixed effect estimates: Examining the significance of fixed effects as well as the coefficients of covariates.
- Contrast testing: Examining the significance of genetic group[4] and herd (farm).

1.4 Significance of the study

Since simulation techniques use small and incomplete datasets along with professionals' or experts' suggestion about the subject matter as an input for generating the desired datasets, a new era of using simulation, as means of problem solving and data generating mechanism, will be increased. As a result, in order to have a solution for the ever increasing demand of the consumption of milk, this paper is expected to significantly initiate or motivate animal scientist to turn their faces to this method.

This method needs the knowledge of general-purpose programming language (computer science) and statistical methods as well as animal science. Due to this, this paper will initiate computer science professionals, statisticians and animal scientists to work together.

1.5 Scope and Limitation of the study

The level of husbandry under smallholder cattle production system, unlike the situation in the two farms (Holetta & Debre Zeit), is characterized by the prevalence of many diseases, feed shortage and lack of skill on modern practices (Mohamed *et al*. 2004). Therefore, results of the

[4] Classified cow on the basis of Holstein inheritance

present study should be interpreted with an intensive dairy farm in mind. Moreover, the simulation technique that would be selected as best in this study didn't imply its power and accuracy of approximating the answer in every real world problem.

1.6 Organization of the study

Organization of the rest of this paper is presented as follows: Chapter two reviews the literature. Chapter three devotes to provide research methods and source of data. Chapter four provides the basic findings along with discussion. Chapter five is reserved for conclusion.

2 Literature Review

2.1 Steps in Simulation Study

According to Banks (1998), there are twelve basic steps in simulation study.

1. *Problem formulation:* Every simulation study begins with a statement of the problem
2. *Setting of objectives and overall project plan:* Another way to state this step is "prepare proposal." The objectives indicate the questions that are answered by the simulation study.
3. *Model conceptualization:* The real-world system under investigation is abstracted by a conceptual model, a series of mathematical and logical relationships concerning the components and structure of the system.
4. *Data collection:* At this stage, the necessary data which will serve as input for simulation will be collected.
5. *Model translation:* The conceptual model constructed in step 3 is coded into a computer-recognizable form, an operational model.
6. *Verification* concerns the operational model. So, verification is a determination of whether the computer implementation of the conceptual model is correct. Does the operational model represent the conceptual model?
7. *Validation* is a determination that the conceptual model is an accurate representation of the real system.
8. *Experimental design:* For each scenario[5] that is to be simulated, decisions need to be made concerning the length of the simulation run, the number of runs (also called replications), and the manner of initialization, as required.
9. *Production run and analysis:* Production runs, and their subsequent analysis, are used to estimate measures of performance for the scenarios that are being simulated.

[5] Scenario is a synthetic description of an event or series of actions and events

10. *More runs?* Based on the analysis of runs that have been completed, the researcher determines if additional runs are needed and if any additional scenarios need to be simulated.
11. *Documentation and reporting:* Documentation is necessary for numerous reasons. If the simulation model is going to be used again by the same or different researcher, it may be necessary to understand how the simulation model operates. The results of all analysis should be reported.
12. *Implementation:* the researcher acts as a reporter rather than an advocate.

2.2 Simulation Model Building

As Rubinstein (1981) mentioned, unlike data mining[6], the fist step in studying a system[7] is building a model. The importance of models and model building as well has been highly emphasized by Rosbenbluth and Wiener (1945), who wrote:

> *No substantial part of the universe is so simple that it can be grasped and controlled without abstraction. Abstraction consists in placing the part of the universe under consideration by a model of similar and simpler structure. Models ... are thus the central necessity of scientific procedure* (Rosbenbluth and Wiener 1945).

A scientific model can be defined as an abstraction of some real system, an abstraction that can be used for prediction and control. The vital step in model building is construction of objective functions.[8] In simulation study we have had so many model types such as *ionic, analog, verbal, symbolic* etc. Perros (2009) and Rubinstein (1981) defined some type of models as follows: Ionic Model is a model that pictorially or visually represents a system or is an exact replica of the properties of the real-life system, but in smaller scale. Analog Model is a model that uses one set of properties to represent some other set of properties that the system being studied possesses. Symbolic Model is a model requires mathematical or logical operations and can be

[6] Data mining is the process of extracting patterns from data. It always goes from data to model building.
[7] System is a set of interacting or interdependent entities forming an integrated whole. The concept of an *'integrated whole'* can also be stated in terms of a system embodying a set of relationships which are differentiated from relationships of the set to other elements, and from relationships between an element of the set and elements not a part of the relational regime.
[8] A function that defines how well data fit a particular hypothesis

used to formulate a solution to the problem at hand. This type of model consists of mathematical symbols or flowcharts.

Fishman (1973) emphasized that symbolic model has many advantages than the other types of model and thus most simulation studies have been done merely by taking this type of model and integrating with the other one. Some of the advantages are that a symbolic model enables investigators to organize their theoretical beliefs and empirical observations about a system and to reduce the logical implications of this organization, leads to improve system understanding, brings into perspective the need for detail and relevance, expedites the analysis, provides a framework for testing the desirability of system modifications, allows for easier manipulation than the system itself permits, permits control over more sources of variation than direct study of a system would allow, and is generally less costly than the system.

An additional advantage is that a mathematical model describes a problem more concisely than, for instance, a verbal description does. On the other hand, there are at least three reservations in Fishman's monograph, which we should always bear in mind while constructing a model. First, there is no guarantee that the time and effort devoted to modeling will return a useful result and satisfactory benefits. Occasional failures occur because the level of resources is too low. Most often, however, failure results when the investigator relays too much on method and not enough on ingenuity; the proper balance between the two leads to the greatest probability of success. The second reservation concerns the tendency of an investigator to treat his or her particular depiction of a problem as the best representation of reality. This is often the case after much time and effort have been spent and the investigator expects some useful results. The third reservation concerns the use of the model to predict the range of its applicability without proper qualification.

While building a model, care must be taken to ensure that it remains a valid representation of the problem. In order to be useful or valid, a scientific model necessarily embodies elements of two conflicting attributes, realism and simplicity. On the other hand, the model should serve as a reasonably close approximation to the real system and incorporate most of important aspects of the system.

2.3 Monte Carlo Vs Bootstrap Simulation

Simulation is defined as a technique of performing sampling experiments on the model of the system. This general definition is often called simulation in a wide sense. Simulation in a narrow sense or stochastic simulation is defined as experimenting with the model over time. It includes sampling stochastic variates from probability distribution. Since sampling from a particular distribution involves the use random numbers, stochastic simulation is sometimes called *Monte Carlo simulation*. Historically, the Monte Carlo method was considered to be a technique, using random or pseudorandom numbers, for solution of a model.

One of the earliest problems associated with Monte Carlo method is the famous Buffon's needle problem. In the beginning of the 20^{th} century, the Monte Carlo was used to examine the Boltzmann equation[9]. In 1908 the famous statistician William S. Gosset used the Monte Carlo method for estimating the correlation coefficient in his t distribution.

The term *Monte Carlo* was introduced by von Neumann and Ulam during World War II, as a code word for the secret work at Los Alamos. It was suggested by the gambling casinos at the city of Monte Carlo in Monaco (Rubinstein 1981; Law 2007).

The Monte Carlo method was then applied to problems related to the atomic bomb. The work involved direct simulation of behavior concerned with random neutron diffusion in fissionable material. Shortly, thereafter, Monte Carlo methods were used to evaluate complex multidimensional integrals and to solve certain integral equations, occurring in physics that were not amenable to analytic solution.

The main feature of simulation, as described by Sobol (1975), is the simple structure of the computational algorithm. As a rule, a program is prepared to perform only one random trial. This trial is repeated N times. Each trial being independent of all others and the results of all trials are averaged.

When you "pull yourself up by your bootstraps," you succeed, on your own, despite limited resources. This idiom is derived from *The Surprising Adventures of Baron Munchausen* by

[9] Boltzmann equation describes the statistical distribution of one particle in a fluid and is used to study how a fluid transports physical quantities such as heat and charge.

Rudolph E. Raspe. The baron tells a series of tall tales about his travels, including various impossible feats and daring escapes. Bradley Efron chose "the bootstrap" to describe a particular re-sampling scheme he was working on because

> *The use of the term bootstrap derives from the phrase to pull oneself up by one's own bootstrap . . . The Baron had fallen to the bottom of a deep lake. Just when it looked like all was lost, he thought to pick himself up by his own bootstraps* (Efron and Tibshirani 1993).

There are many types of bootstrapping because there are many ways to resample, and there are a variety of ways to use the bootstrapped samples. Among them simple (naive), balanced and double bootstrapping[10] are commonly practiced by simulation practitioners.

The tie between the bootstrap and Monte Carlo simulation of a statistic is that both are based on repetitive sampling and then direct examination of the results. A big difference between the methods, however, is that bootstrapping uses the original, initial sample as the population from which to resample, whereas Monte Carlo simulation is based on setting up a data generation process (with known values of the parameters).

2.4 Simulation Packages Vs General-Purpose Programming Languages

One of the most important decisions a modeler or an analyst must make in performing a simulation study concerns the choice of software or general-purpose programming language. If the selected simulation package or general-purpose programming language is not flexible enough or is too difficult to use, then the simulation project may produce erroneous results or may not even be completed.

The general-purpose programming languages are general in nature, and model development is done by writing a code. These types of languages provided, in general, a great deal of modeling flexibilities, but are often difficult to use. Additionally, these general-purpose programming

[10] Double bootstrapping involves bootstrap sampling of bootstrap samples. This can lead to improved confidence limit for unknown parameters, but at the expense of much more computing time.

languages may require less execution time than a model development in a simulation package, are Object-Oriented[11], which is of considerably important to many analysts and programmers, such as those in defense industry. On the other hand, simulation packages are not truly Object-Oriented and cost is generally lower, but total project cost may not be.

However, there are some advantages of using simulation packages rather than a general-purpose programming language (Law 2007). That is, simulation packages (such as Arena, Extend, SIMULA8, GoldSim etc) automatically provide most of the features needed to build a simulation model, resulting in a significant decrease in programming time and reduction in the overall project cost. They provide a natural framework for simulation modeling. Their basic modeling constructs are those in a general-purpose programming language like C++. They are generally easier to modify and maintain when written in a simulation package and provide better error detection because many potential types of errors are checked automatically.

[11] Object-Oriented programming refers to a programming methodology based on objects, instead of just functions and procedures. These objects are organized into classes, which allow individual objects to be group together.

3 Materials and Methods

3.1 Sources of Data

Data for the study were collected from experimental dairy cattle herds of Ethiopian Boran and Ethiopian Boran-Holstein crossbred cattle maintained at the Debre Zeit Research Station of the International Livestock Research Institute (ILRI) and at the Holetta Agricultural Research Centre of the Ethiopian Institute of Agricultural Research (EIAR). Five genetic groups (Ethiopian Boran (0%), 50%, 62.5%, 75% and 87.5% of Holstein inheritance) represented in the two station were used. Fifteen years (1990 to 2004) of data were used for this study.

3.1.1 Variable Definition

Dairy Farm is a class of agriculture, where female cattle, goats, or other mammals are raised for their milk, which may be either processed on-site or transported to a dairy processing and eventual retail sale. For this study we will use two dairy farm so called Debre Zeit Research Station of the ILRI, which is located on the outskirts of the town of Debre Zeit, about 50km south-east of Addis Ababa, in the Ethiopian highlands (8° 45′ 0″ N, 38° 59′ 0″ E), and annual rain fall, average temperature and average monthly humidity are 866mm, 18.7^{o}C and 52.4%, respectively, and the Holetta Agricultural Research Centre which is located 45km west of Addis Ababa at 38.5^{o}E longitude and 9.8^{o}N latitude, and elevation of 2400m above see level. It is suited in the central highlands of Ethiopia. The annual temperature ranges between 18°C and 24°C and rainfall is recorded between 1000 and 1100 mm (Haile *et al*. 2009; Wikipedia 2009; Alemu *et al.* 1998; Tesfaye 1992).

Parity: The number of live-born calves a cow has delivered

Season of calving: A period of the year marked by the pattern of the annual rainfall distribution in the area (November to February: dry period; March to June: light rain and July to October: main rainy season).

Period of calving: The year group that the cattle give birth. It has five classes: 1990-1992, 1993-1995, 1996-1998, 1999-2001 and 2002-2004.

Genetic group: We have five genetic groups based on the inheritance of Holstein. These are Ethiopian Boran, 50%, 62.5%, 75% and 87.5% of Holstein inheritance.

Lactation length: The number of days a cow stays in giving milk. In other word, the gap between dry off date and calving date. Lactation is the period following birth during which milk is secreted.

Lactation milk yield: The total milk yield in the lactation period.

Daily milk yield: The average milk yield per a day

3.2 Monte Carlo Simulation

Law (2007) defined Monte Carlo simulation as a scheme of employing random numbers, that is, $U(0,1)$ random variates which is used for solving certain stochastic and deterministic problems[12]. Thus, Monte Carol simulation is a versatile computerized statistical method of analysis based on artificial recreating a chance process or process involves uncertainty (usually with a computer), running it many times and directly observing results.

3.2.1 Selecting Input Probability Distribution

The first step in selecting a particular input distribution is to decide what general families appeared to be appropriate on the basis of their shapes without worrying about the specific parameter values for these families. In some situations, use can be made of prior knowledge about certain random variable's role in a system to select a modeling distribution or at least rule out some distribution. This is done on the theoretical grounds and doesn't require any data at all.

In practice, we seldom have enough of this kind of theoretical prior information to select a single distribution. The task of hypothesizing a distribution family from observed data is somewhat less structured. Some of the heuristics or guidelines that can be used to choose

[12] a stochastic discrete-event simulation is included in this definition

appropriate families of distributions are summary statistics, histogram, quantile summaries and box plot.

The second step is estimation of parameters of the hypothesized distribution families. An *estimator* is a numerical function of the data. There are many ways to specify the form of an estimator for a particular parameter of a given distribution.

We considered explicitly only one type, *maximum-likelihood estimator* (MLE). The MLE, based on the maximum likelihood function, is appropriate for a variety of parametric distributions. Moreover, it has the capability of estimating parameters from censored or binned data (Law 2007; Frey and Burmaster 1999).

The most important cases which we shall consider are those in which $X_1, X_2, \cdots, X_n$ is a *random sample* from some probability density $f(x;\theta)$, so that the likelihood function is

$$L(\theta) = f(x_1;\theta)f(x_2;\theta)\cdots f(x_n;\theta) \qquad [3.1]$$

Under regularity conditions[13], the maximum likelihood estimator is the solution of the equation

$$\frac{\partial L(\theta)}{\partial \theta} = 0 \qquad [3.2]$$

Since $L(\theta)$ and $\log L(\theta)$ have their maxima at the same value of θ, it is sometimes easier to find the maximum of logarithm of the likelihood (Mood *et al.* 1974). If the likelihood function contains k parameters, that is, if

$$L(\theta_1, \theta_2, \cdots, \theta_k) = \prod_{i=1}^{n} f(x;\theta_1, \theta_2, \cdots, \theta_k) \qquad [3.3]$$

Then the maximum-likelihood estimators of the parameters $\theta_1, \theta_2, \cdots, \theta_k$ are the random variables $\widehat{\Theta}_1 = \hat{\vartheta}_1(x_1, x_2, \cdots, x_n), \widehat{\Theta}_2 = \hat{\vartheta}_2(x_1, x_2, \cdots, x_n), \cdots, \widehat{\Theta}_k = \hat{\vartheta}_k(x_1, x_2, \cdots, x_n)$, where $\hat{\theta}_1, \hat{\theta}_2, \cdots, \hat{\theta}_k$ are the values in Φ which maximize $L(\theta_1, \theta_2, \cdots, \theta_k)$.

---

[13] *i.* The Fisher information is always defined; equivalently, for all x such that $f(x;\ \theta) > 0$, $\frac{\partial \ln f(x;\theta)}{\partial \theta}$ exists and is finite

ii. The operations of integration with respect to *x* and differentiation with respect to θ can be interchanged in the expectation of *T* (the estimator)

Analogues with equation [3.2], the point where the likelihood is a maximum for the solutions of the following k equations

$$\frac{\partial L(\theta_1, \theta_2, \cdots, \theta_k)}{\partial \theta_1} = 0$$

$$\frac{\partial L(\theta_1, \theta_2, \cdots, \theta_k)}{\partial \theta_2} = 0 \qquad [3.4]$$

$$\vdots$$

$$\frac{\partial L(\theta_1, \theta_2, \cdots, \theta_k)}{\partial \theta_k} = 0$$

After determining one or more probability distributions that might fit our observed data, the next task is observing how well they represent the true underlying distribution for the observed data. If several of these distributions are "representative," we must also determine which distribution provides the best fit.

In general, none of our fitted distributions will probably be exactly correct. What we are really trying to do is to determine a distribution that is accurate enough for the intended purposes of the model. We chose the Kolmogorov-Smirnov one-sample test due to its power even though there are different goodness-of-fit tests (Law 2007). The process of fitting distribution to the data was done by EasyFit® software[14].

3.2.1.1 Kolmogorov-Smirnov Goodness-of-fit test

Several goodness of fit test statistics are functions of the deviations between the observed cumulative distribution and the corresponding cumulative probability expected under the null hypothesis. The Kolmogorov-Smirnov one-sample is based on the difference between the

[14] EasyFit is a data analysis and simulation application software allowing to fit probability distributions to sample data, select the best model, and apply the analysis results to make better decisions. EasyFit can be used as a stand-alone Windows application or with Microsoft Excel and other third party Excel-based simulation tools, leaving the complex technical details behind the scenes and enabling you to focus on your business goals. It gives rank for 61 different theoretical distributions on the basis of three goodness-of-fit tests.

hypothesized cumulative distribution function $F_o(x)$ and the empirical distribution function[15] $S_n(x)$, for all x. Furthermore, it is true that $S_n(x)$ provides a consistent point estimator for the true distribution $F_X(x)$. Therefore, for large n, the deviations between the true function and its statistical image, $|S_n(x) - F_X(x)|$, should be small for all values of x. This result suggests that the statistic

$$D_n = \sup_x |S_n(x) - F_X(x)| \qquad [3.5]$$

or which can also be rewritten for simplicity as

$$D_n = max\left\{\max_{1\le i\le n}\left[\frac{i}{n} - F_X(x_{(i)})\right], \max_{1\le i\le n}\left[F_X(x_{(i)}) - \frac{i-1}{n}\right], 0\right\} \qquad [3.6]$$

is, for any n, a reasonable measure of the accuracy of our estimate. This statistic, called the Kolmogorov-Smirnov goodness-of-fit statistic, is particularly useful in non-parametric statistical inference because the probability distribution of D_n doesn't depend on $F_X(x)$ as long as F_X is continuous. The only assumption of Kolmogorov-Smirnov goodness-of-fit test is that the sample must be random (Law and Kelton 2000).

3.2.2 Generating Pseudo-random Numbers

A simulation of any system or process in which there are inherently random components requires a method of generating numbers that are *random*. Random variates generated from $U(0,1)$ distribution would be called random numbers.

This prominent role of the $U(0,1)$ distribution stems from the fact that random variates from all other distributions (normal, gamma, binomial, etc) and realizations of various random process can be obtained by transforming iid random numbers in a way determined by the desired distribution or process (Kleijinen and Groenendaal 1992; Gardner and Baker 1997).

Though there are a number of random number generators, we used the most familiar random number generator which is called *linear congruential generator*.

[15] The empirical distribution function is defined as the proportion of sample observations that are less than or equal to x for all real number x.

3.2.2.1 Linear Congruential Generator

Many random-number generators in use today are linear congruential generators, introduced by Lehmer (1951). A sequence of integers $Z_1, Z_2, \cdots$ is defined by the recursive formula

$$Z_i = (aZ_{i-1} + c)(mod\ m) \qquad [3.7]$$

where m (the *modulus*), a (the *multiplier*), c (the *increment*) and Z_0(the *seed* or *starting value*) are non negative numbers. Thus, the above equation says that to obtain Z_i, divide $aZ_{i-1} + c$ by m and let Z_i be the *remainder* of this division.

3.2.3 Generating Random Variates

A simulation that has any random aspect at all must involve sampling, or generating, random variates from probability distribution. We used the phrase *generating random variate* to refer to the activity of obtaining an observation on (or a realization of) a random variable from the desired distribution. These distributions were often specified as a result of fitting some appropriate distributional form.

The basic gradient needed for every method of generating random variates from any distribution or random process is a source of $iid\ U(0,1)$ random variates.

3.2.3.1 Generating Bernoulli Random Variate

If $X \sim Ber(p)$, its pdf is of the form

$$f(x) = p^x(1-p)^{1-x}, \quad x = 0, 1 \qquad [3.8]$$

where p is the success probability. Applying the inverse transformation method[16], one readily obtains the following algorithm.

Algorithm 1: Generation of a Bernoulli random variate

1. Generate $U \sim U(0,1)$

[16] Let X be a random variable with *CDF F*. Since F is a non decreasing function, the inverse function F^{-1} may be defined as $F^{-1}(y) = inf\{x: F(y) \geq y\}, 0 \leq y \leq 1$. It is easy to show that if $U \sim U(0,1)$, then $X = F^{-1}(U)$ has *CDF F.*

2. If $U \leq p$, return $X = 1$. Otherwise, return $X = 0$

3.2.3.2 Generating Normal Random Variate

A normal variate X with mean μ and standard deviation σ can be generated from a standard normal variate Z using the transformation $X = \mu + \sigma Z$. The pdf of X is

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} exp\left\{-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right\}, \quad -\infty < x < \infty \qquad [3.9]$$

There are a number of algorithms to generate a normal variate. Among them Box-Muller and Polar methods are popular. However, according to Box-Muller (1958), the Box-Muller algorithm takes much executing time. An improvement to the Box and Muller method, which eliminates the trigonometric calculations and was described in Marsaglia and Bray (1965), has become known as the polar method. It relies on a special property of the normal distribution. It is also expected to be faster than Box-Muller method and its algorithm is

Algorithm 2: Polar Method

1. Generate U_1 and $U_2 \sim U(0,1)$ independently
2. Calculate $W_1 = 2U_1 - 1$ and $W_2 = 2U_2 - 1$
3. If $W = W_1^2 + W_2^2 > 1$ go to step 1
4. Calculate $C = \sqrt{\left(-\frac{2 \ln W}{W}\right)}$, and return $X = CW_1$ and $Y = CW_2$

Since we obtain the desired random variates in pairs, we could, on odd-number calls to the sub-program, actually compute X and Y as just described, but return only X, saving Y for immediate return on the next (even-numbered) call.

3.2.3.3 Generating Johnson Bounded Random Variate

If $X \sim JSB(\alpha_1, \alpha_2, a, b)$, then its pdf is of the form

$$f(x) = \begin{cases} \frac{\alpha_2(b-a)}{(x-a)(b-x)\sqrt{2\pi}} exp\left[-\frac{1}{2}\left\{\alpha_{1+}\alpha_2 \ln\left(\frac{x-a}{b-x}\right)\right\}^2\right] & if\ a < x < b \\ 0 & otherwise \end{cases} \qquad [3.10]$$

where $a \in (-\infty, \infty)$ is *location* parameter, $b - a$ is *scale* parameter $(b > a)$, and $\alpha_1 \in (-\infty, \infty)$ and $\alpha_2 > 0$ are *shape* parameters.

$X \sim JSB(\alpha_1, \alpha_2, a, b)$ if and only if $Z = \alpha_1 + \alpha_2 \ln\left(\frac{x-a}{b-x}\right) \sim N(0,1)$ and we can solve this equation for X in terms of Z to get the following algorithm:

Algorithm 3: Generating Johnson SB random variate

1. Generate $Z \sim N(0,1)$
2. Let $Y = exp\left[\frac{(Z-\alpha_1)}{\alpha_2}\right]$
3. Return $X = \left[(a + bY)/(Y + 1)\right]$

3.2.3.4 Generating Log-Logistic With Three Parameters

If $X \sim LL_{3P}(\alpha, \beta, \gamma)$, its pdf is of the form

$$f(x) = \begin{cases} \dfrac{\alpha\left(x - \gamma/\beta\right)^{\alpha-1}}{\beta\left[1 + \left(x - \gamma/\beta\right)^{\alpha}\right]^2} & if\ x > 0 \\ 0 & otherwise \end{cases} \qquad [3.11]$$

where $\alpha > 0$ *shape* parameter, $\beta > 0$ *scale* parameter and γ is *location*[17] parameter. The log-logistic distribution function can be inverted to obtain

$$F^{-1}(u) = \beta\left(\frac{u}{1-u}\right)^{1/\alpha} \qquad [3.12]$$

which leads to the inverse-transformation algorithm:

Algorithm 4: Generating Log-Logistic with three parameters

1. Generate $U \sim U(0,1)$
2. Return $X = \beta\left(\frac{u}{1-u}\right)^{1/\alpha} + \gamma$

[17] $\gamma \equiv 0$ yields two parameters Log-Logistic distribution

3.2.3.5 Generating Gamma Random Variate

If $X \sim Gamma(\alpha, \beta)$, then its pdf is of the form

$$f(x) = \frac{x^{\alpha-1}\beta^{\alpha}e^{-\beta x}}{\Gamma(\alpha)}, \quad x \geq 0 \qquad [3.13]$$

The parameters $\alpha > 0$ and $\beta > 0$ are called the *shape* and *scale* parameters, respectively.

General gamma random variates are more complicated to generate since the distribution has no simple closed form for which we could try to find an inverse. First, note that given $X \sim Gamma(\alpha, 1)$, we can obtain for any $\beta > 0$, a $Gamma(\alpha, \beta)$ a random variate X' by letting $X' = \beta X$, so that it is sufficient to restrict attention to generating from the $Gamma(\alpha, 1)$ distribution (Fishman 1976; Cheng and Feast 1979).

Algorithm 5: Sampling from the $Gamma(\alpha, 1)$ distribution $(\alpha \geq 1)$

1. Constants: $a = \alpha - 1$, $b = {(\alpha - (6\alpha)^{-1})}/{a}$, $c = {2}/{a}$, $d = c + 2$
2. Generate independent $U(0,1)$ variates U_1 and U_2
3. Let $W = {bU_1}/{U_2}$. If $cU_2 - d + W + W^{-1} \leq 0$ go to step 5
4. If $c \log U_2 - \log W + W - 1 \geq 0$ go to step 2
5. Return $X = aW$

Algorithm 6: Sampling from the $Gamma(\alpha, 1)$ distribution $(0 < \alpha < 1)$

1. Compute $b = {(e + \alpha)}/{e}$ beforehand
2. Generate $U_1 \sim U(0,1)$ and let $P = bU_1$. If $P > 1$ go to step 4
3. Let $Y = P^{1/\alpha}$ and generate $U_2 \sim U(0,1)$. If $U_2 \leq e^{-Y}$, return $X = Y$. Otherwise, go back to step 2
4. Let $Y = -\ln\left[{(b - P)}/{\alpha}\right]$ and generate $U_2 \sim U(0,1)$. If $U_2 \leq Y^{\alpha-1}$, return $X = Y$. Otherwise, go back to step 2.

3.2.3.6 Generating Pearson Type VI Random Variate

If $X \sim PT6_{4P}(\alpha_1,\ \alpha_2,\ \beta,\ \gamma)$, then its pdf is of the form

$$f(x) = \begin{cases} \dfrac{\left((x-\gamma)/\beta\right)^{\alpha_1-1}}{\beta B(\alpha_1,\alpha_2)\left[1+\left((x-\gamma)/\beta\right)\right]^{\alpha_1+\alpha_2}} & if\ x > 0 \\ 0 & otherwise \end{cases} \qquad [3.14]$$

where $\alpha_1 > 0$ and $\alpha_2 > 0$ are *shape* parameter, $\beta > 0$ is *scale* parameter and γ is *location* parameter.[18] $B(\alpha_1, \alpha_2)$ is beta distribution with parameters α_1 and α_2.

We note that if $Y_1 \sim gamma(\alpha_1, \beta)$ and $Y_2 \sim gamma(\alpha_2, 1)$, and Y_1 and Y_2 are independent, then $\left(Y_1/Y_2\right) + \gamma \sim PT6_{4P}(\alpha_1,\ \alpha_2,\ \beta,\ \gamma)$; this leads directly to:

Algorithm 7: Generating Pearson type VI of four parameter variate

1. Generate $Y_1 \sim Gamma(\alpha_1, \beta)$ and $Y_2 \sim Gamma(\alpha_2, 1)$ independent of Y_1
2. Return $X = \left(Y_1/Y_2\right) + \gamma$.

3.2.3.7 Generating Dagum Random Variates

If $X \sim Dagum(k, \alpha, \beta, \gamma)$, then its pdf is of the form

$$f(x) = \frac{\alpha\beta\left(\dfrac{x-\gamma}{\beta}\right)^{\alpha k-1}}{\beta\left(1+\left(\dfrac{x-\gamma}{\beta}\right)^{\alpha}\right)^{k+1}}, \qquad \gamma \le x < +\infty \qquad [3.15]$$

The parameters $k > 0, \alpha > 0,\ \beta > 0$ and $\gamma > 0$ are *shape*, *shape*, *scale* and *location* parameters, respectively.

[18] $\gamma \equiv 0$ yields three parameters Pearson Type VI distribution

If $X \sim Dagum(k, \alpha, \beta)$, then its pdf is of the form

$$f(x) = \frac{\alpha\beta\left(\frac{x}{\beta}\right)^{\alpha k-1}}{\beta\left(1 + \left(\frac{x}{\beta}\right)^{\alpha}\right)^{k+1}}, \quad 0 \leq x < +\infty \qquad [3.16]$$

The Dagum with three parameters distribution can be inverted to obtain

$$F^{-1}(u) = \beta\left[\left(\frac{1}{u}\right)^{1/k} + 1\right]^{-\frac{1}{\alpha}} \qquad [3.17]$$

which leads to the inverse-transform algorithm.

Algorithm 8: Generating Dagum with three parameters random variate

1. Generate $U \sim U(0, 1)$
2. Return $X = \beta\left[\left(\frac{1}{U}\right)^{1/k} + 1\right]^{-\frac{1}{\alpha}}$

The Dagum with four parameters distribution can also be inverted with the same fashion as of Dagum with three parameters to obtain

$$F^{-1}(u) = \beta\left[\left(\frac{1}{u}\right)^{1/k} + 1\right]^{-\frac{1}{\alpha}} + \gamma \qquad [3.18]$$

which is suited to apply inverse-transform algorithm.

Algorithm 9: Generating Dagum with four parameters random variate

1. Generate $U \sim U(0, 1)$
2. Return $X = \beta\left[\left(\frac{1}{U}\right)^{1/k} + 1\right]^{-\frac{1}{\alpha}} + \gamma$

3.2.3.8 Generating Cauchy Random Variate

If $X \sim Cauchy(\mu, \sigma)$, then its pdf is of the form

$$f(x) = \left(\pi\sigma\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-1}, -\infty < x < \infty \quad [3.19]$$

The parameters $\sigma > 0$ and μ are *scale* and *location* parameters, respectively.

The *CDF* is

$$F(x) = \frac{1}{\pi}\tan^{-1}\left(\frac{(x-\mu)}{\sigma}\right) + \frac{1}{2} \quad [3.20]$$

Then the inverse is

$$F^{-1}(u) = \sigma\tan\left(\pi\left(u - \frac{1}{2}\right)\right) + \mu \quad [3.21]$$

which is suited to apply inverse-transform algorithm.

Algorithm 10: Generating Cauchy random variate

1. Generate $U \sim U(0, 1)$
2. Return $X = \sigma\tan\left(\pi\left(U - \frac{1}{2}\right)\right) + \mu$

3.2.3.9 Generating Gumbel Min Random Variate

If $X \sim GumbelMin(\mu, \sigma)$, then its pdf is of the form

$$f(x) = \frac{1}{\sigma}exp\left(\frac{(x-\mu)}{\sigma} - exp\left(\frac{(x-\mu)}{\sigma}\right)\right), \quad -\infty < x < \infty \quad [3.22]$$

The parameters $\sigma > 0$ and μ are *scale* and *location* parameters, respectively.

The *CDF* is

$$F(x) = 1 - exp\left(-exp\left(\frac{(x-\mu)}{\sigma}\right)\right) \quad [3.23]$$

Then the inverse is

$$F^{-1}(u) = \sigma \ln\left(\ln\left(\frac{1}{1-\mathrm{u}}\right)\right) + \mu \qquad [3.24]$$

which is suited to apply inverse-transform algorithm.

Algorithm 11: Generating Gumbel Min random variate

1. Generate $U \sim U(0,1)$
2. Return $X = \sigma \ln\left(\ln\left(\frac{1}{1-\mathrm{U}}\right)\right) + \mu$

3.2.3.10 Generating Laplace (Double Exponential)Random Variate

If $X \sim Laplace(\lambda, \mu)$, then its *pdf* is of the form

$$f(x) = \frac{\lambda}{2} exp(-\lambda|\mu - x|), \qquad \infty < x < \infty \qquad [3.25]$$

where $\lambda > 0$ and μ are *inverse scale* and *location* parameters, respectively.

Its *CDF* is

$$F(x) = \begin{cases} \frac{1}{2} exp\big(-\lambda(x-\mu)\big) & x \leq \mu \\ 1 - \frac{1}{2} exp\big(-\lambda(x-\mu)\big) & x > \mu \end{cases} \qquad [3.26]$$

Then the inverse is

$$F^{-1}(u) = \begin{cases} -\frac{\ln(2u)}{\lambda} + \mu & x \leq \mu \\ -\frac{\ln\big(2(1-u)\big)}{\lambda} + \mu & x > \mu \end{cases} \qquad [3.27]$$

which leads to the inverse-transform algorithm.

Algorithm 12: Generating Laplace random variate

1. Generate $U \sim U(0,1)$
2. If $x \leq \mu$, then return $X = -\frac{\ln(2u)}{\lambda} + \mu$; otherwise, return $X = -\frac{\ln\big(2(1-u)\big)}{\lambda} + \mu$

3.2.3.11 Generating Hyperbolic Secant Random Variate

If $X \sim Hypersecant(\mu, \sigma)$, then its pdf is of the form

$$f(x) = \frac{\operatorname{sech}\left(\frac{\pi(x-\mu)}{2\sigma}\right)}{2\sigma}, \quad -\infty < x < \infty \qquad [3.28]$$

The parameters $\sigma > 0$ and μ are *scale* and *location* parameters, respectively. And its *CDF* is

$$F(x) = \frac{2}{\pi}\tan^{-1}\left(\frac{\pi(x-\mu)}{2\sigma}\right) \qquad [3.29]$$

Then its inverse is

$$F^{-1}(u) = \frac{2\sigma \tan\left(\frac{\pi u}{2}\right)}{\pi} + \mu \qquad [3.30]$$

Therefore, we can directly apply inverse-transform algorithm to generate random variate from this distribution.

Algorithm 13: Generating Hypersecant random variate

1. Generate $U \sim U(0, 1)$
2. Return $X = \frac{2\sigma \tan\left(\frac{\pi u}{2}\right)}{\pi} + \mu$

3.2.3.12 Generating Generalized Pareto Random Variate

If $X \sim Gen.Pareto(k, \sigma, \mu)$, then its pdf is of the form

$$f(x) = \begin{cases} \frac{1}{\sigma}\left(1 + k\frac{(x-\mu)}{\sigma}\right)^{-1-1/k} & k \neq 0 \\ \frac{1}{\sigma} exp\left(-\frac{(x-\mu)}{\sigma}\right) & k = 0 \end{cases} \qquad [3.31]$$

$\mu \leq x < \infty$ for $k \geq 0$ and $\mu \leq x < \mu - \sigma/k$ for $k < 0$.

where k, $\sigma > 0$ and μ are *shape*, *scale* and *location* parameters, respectively. Furthermore, its *CDF* is

$$F(x) = \begin{cases} 1 - \left(1 + k\dfrac{(x-\mu)}{\sigma}\right)^{-1/k} & k \neq 0 \\ 1 - exp\left(-\dfrac{(x-\mu)}{\sigma}\right) & k = 0 \end{cases} \quad [3.32]$$

and its inverse is

$$F^{-1}(u) = \begin{cases} \dfrac{\left[1 - \left(\frac{1}{1-u}\right)^{k}\right]\sigma}{k} + \mu & k \neq 0 \\ -\sigma \ln(1-u) + \mu & k = 0 \end{cases} \quad [3.33]$$

Therefore, we can directly apply inverse-transform algorithm to generate random variate from this distribution.

Algorithm 14: Generating Gen. Pareto random variate

1. Generate $U \sim U(0,1)$
2. If $k = 0$, return $X = \sigma \ln(1-U) + \mu$; otherwise, return, $X = \dfrac{\left[1-\left(\frac{1}{1-U}\right)^{k}\right]\sigma}{k} + \mu$

3.2.3.13 Generating Generalized Extreme Value Random Variate

If $X \sim Gen.\,Extreme\ Value(k,\ \mu,\ \sigma)$, then its pdf is of the form

$$f(x) = \begin{cases} \dfrac{1}{\sigma} exp\left(-(1 + kZ)^{-1/k}\right)(1 + kZ)^{-1-1/k} & k \neq 0 \\ \dfrac{1}{\sigma} exp\big(-Z - exp(-Z)\big) & k = 0 \end{cases} \quad [3.34]$$

$1 + k\frac{(x-\mu)}{\sigma} > 0$ for $k \neq 0$, $-\infty < x < \infty$ for $k = 0$ and $Z = \frac{x-\mu}{\sigma}$.

where k, $\sigma > 0$ and μ are *shape, scale* and *location* parameters, respectively. Furthermore, its *CDF* is

$$F(x) = \begin{cases} exp\left(-(1 + kZ)^{-1/k}\right) & k \neq 0 \\ exp\big(-exp(-Z)\big) & k = 0 \end{cases} \quad [3.35]$$

And its inverse is

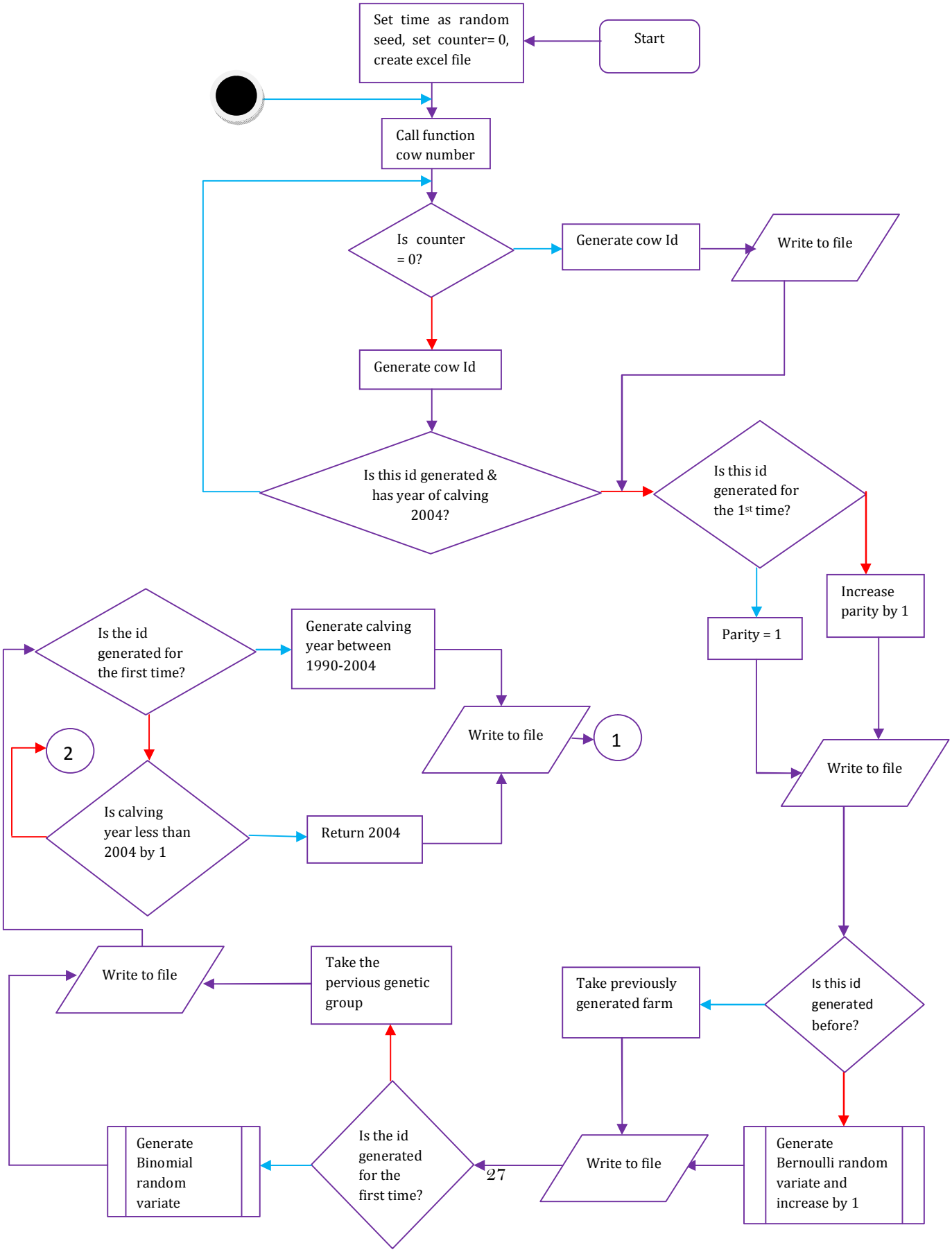
$$F^{-1}(u) = \begin{cases} \frac{\sigma}{k}\left[\left(-\frac{1}{\ln u}\right)^{k} - 1\right] + \mu & k \neq 0 \\ \sigma \ln\left(-\frac{1}{\ln u}\right) + \mu & k = 0 \end{cases} \quad [3.36]$$

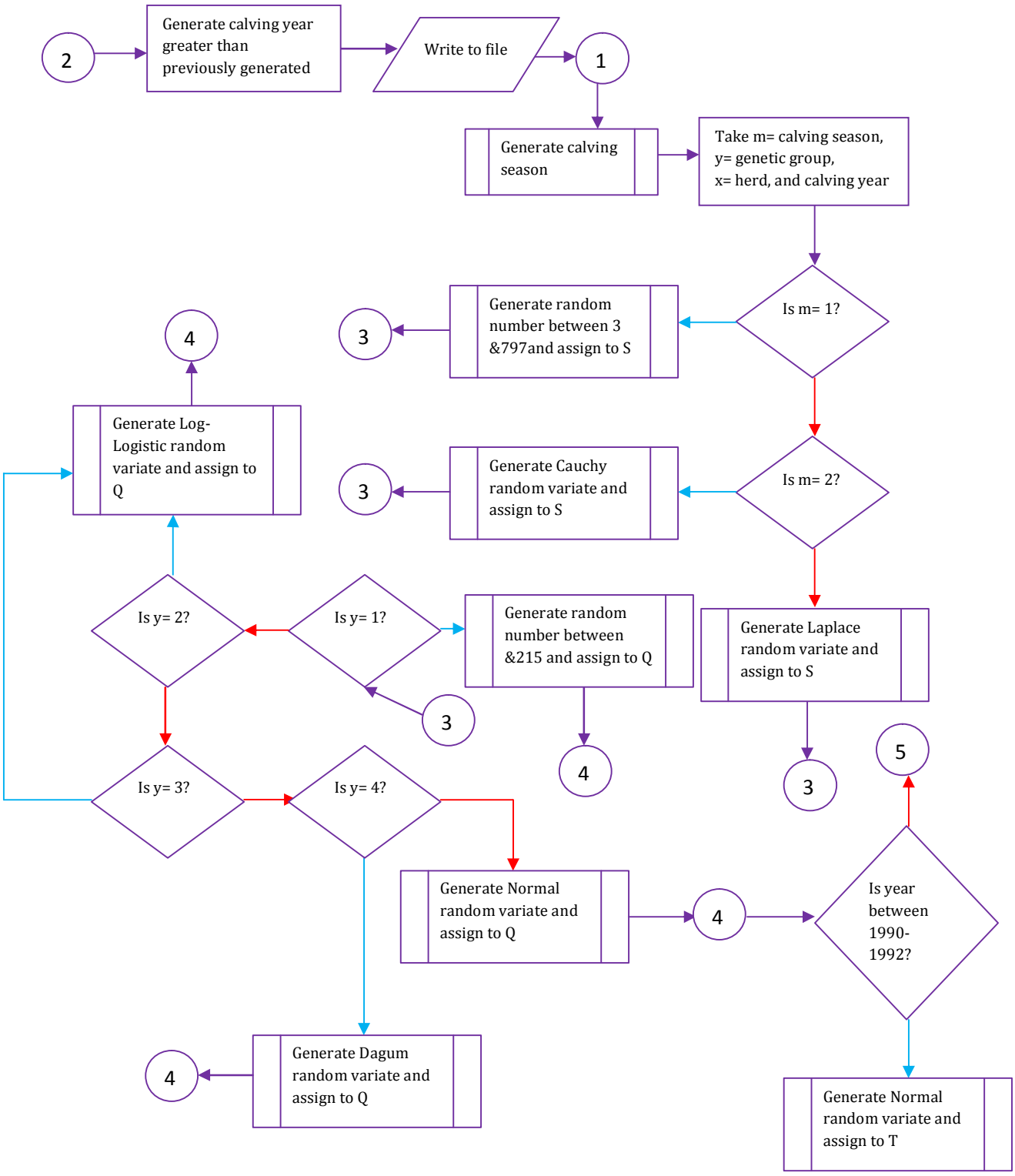
Therefore, we can directly apply inverse-transform algorithm to generate random variate from this distribution.

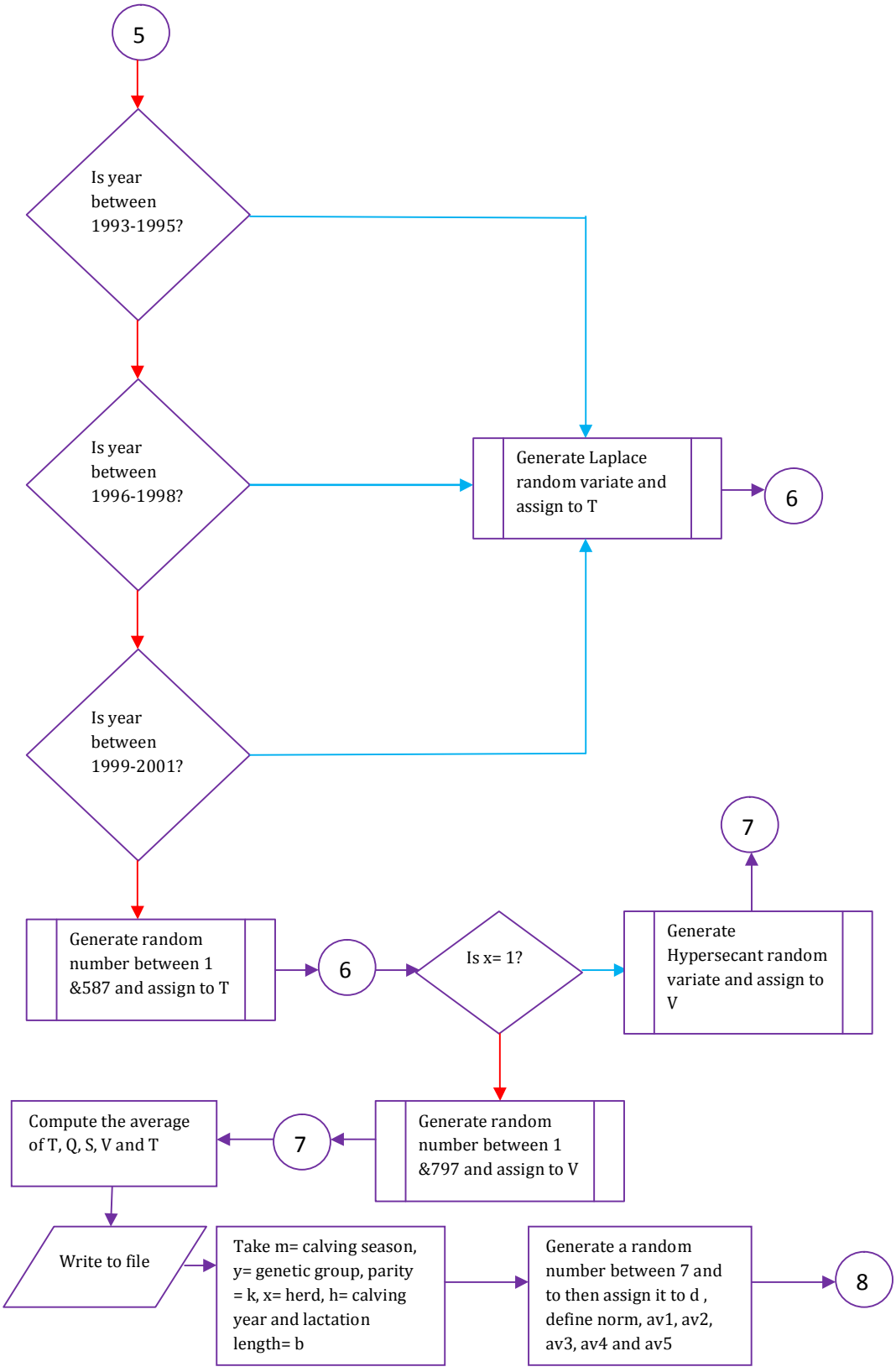
Algorithm 15: Generating Gen. Extreme value random variate

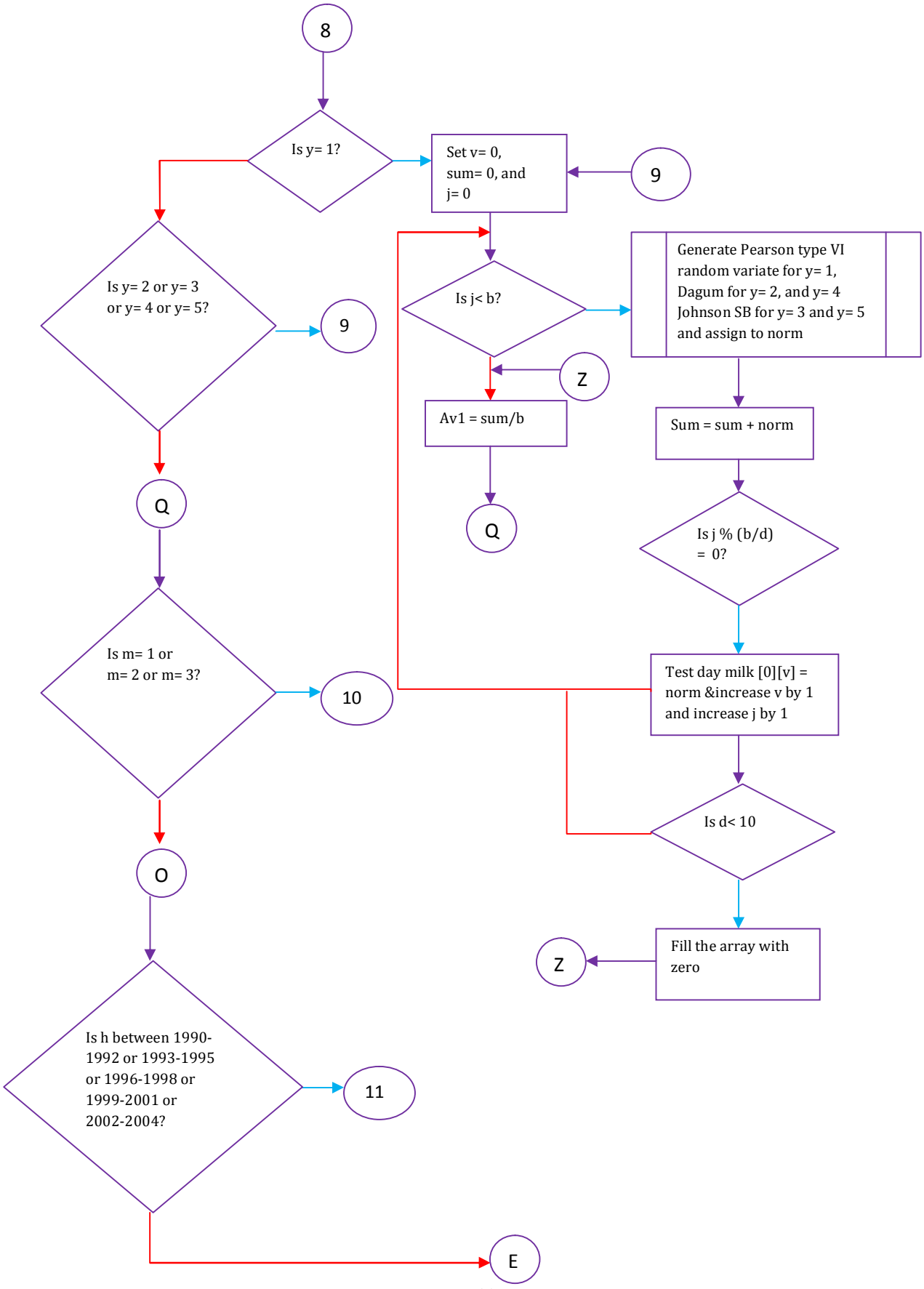
1. Generate $U \sim U(0,1)$
2. If $k \neq 0$, return $X = \frac{\sigma}{k}\left[\left(-\frac{1}{\ln U}\right)^{k} - 1\right] + \mu$; otherwise, return $X = \sigma \ln\left(-\frac{1}{\ln U}\right) + \mu$

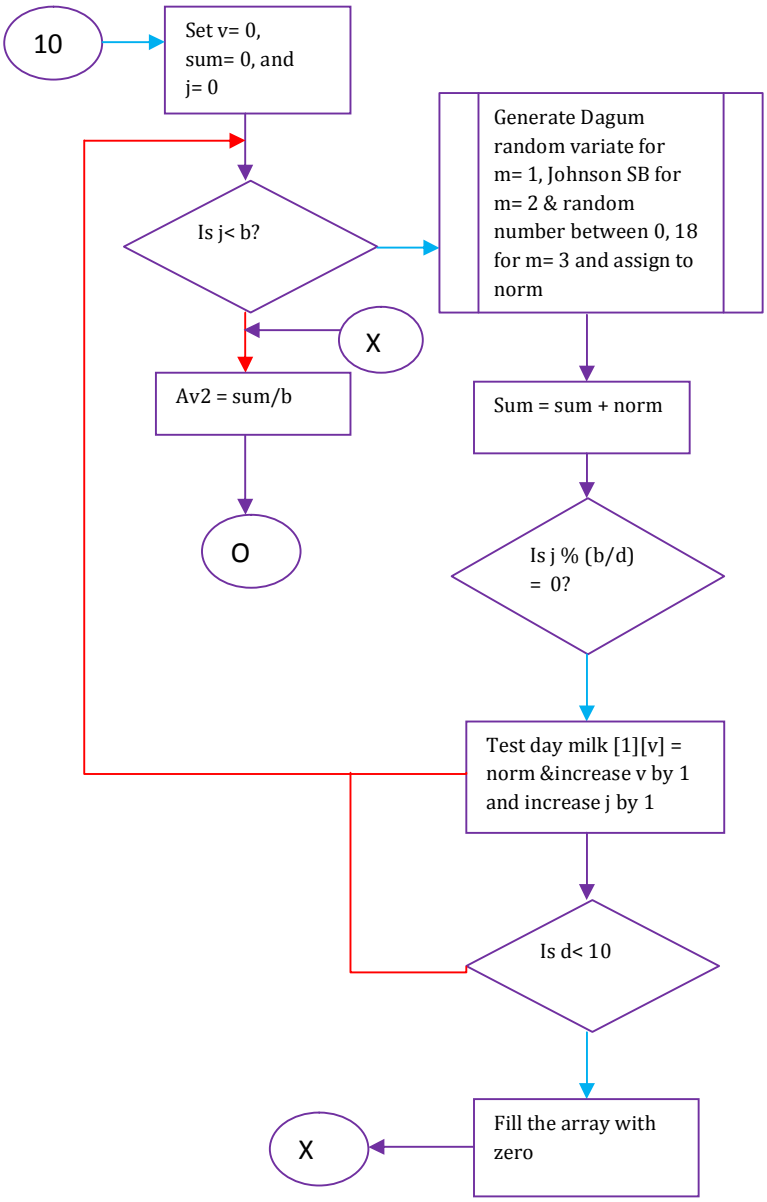
3.2.4 Monte Carlo Simulation Model

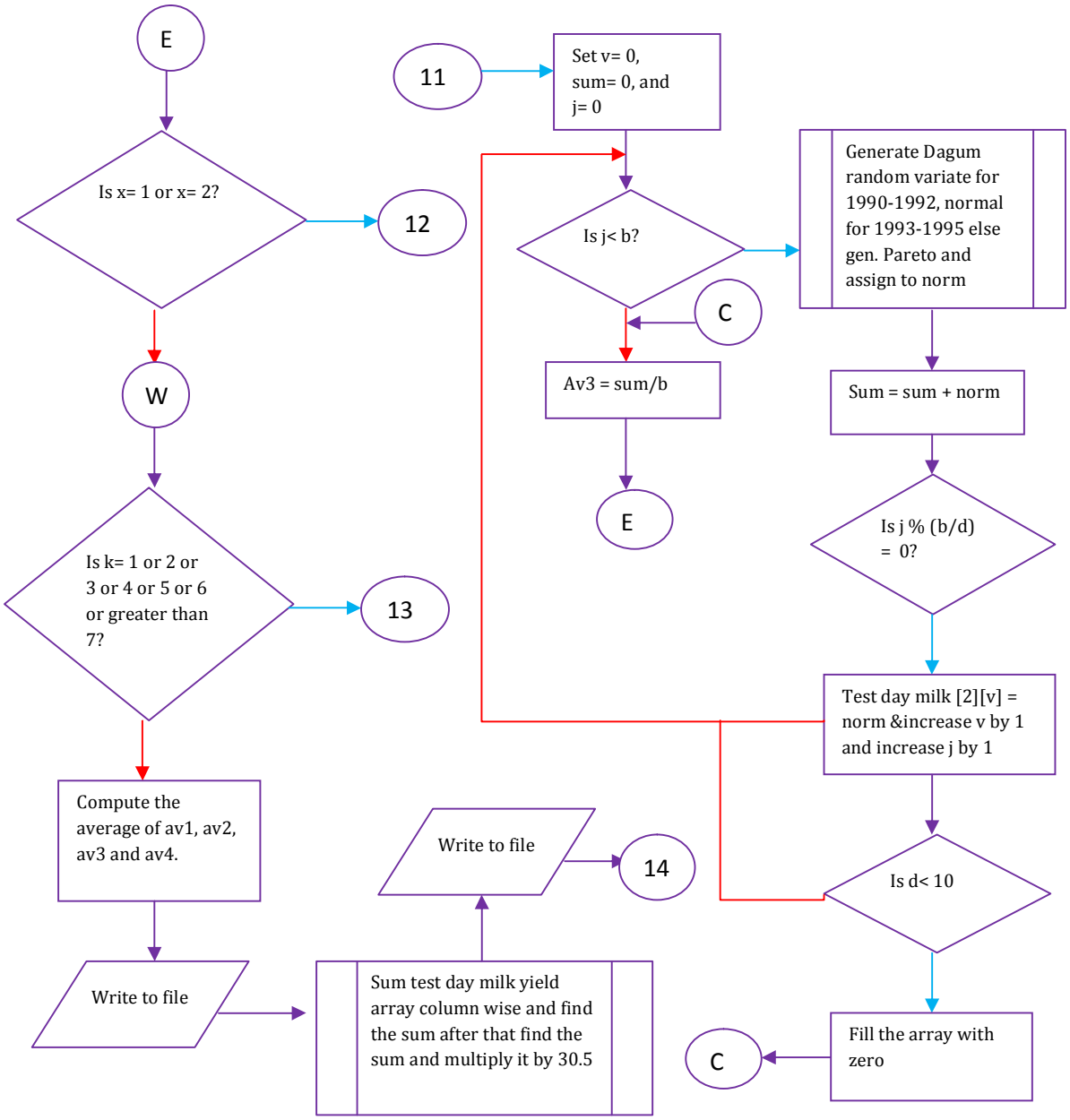


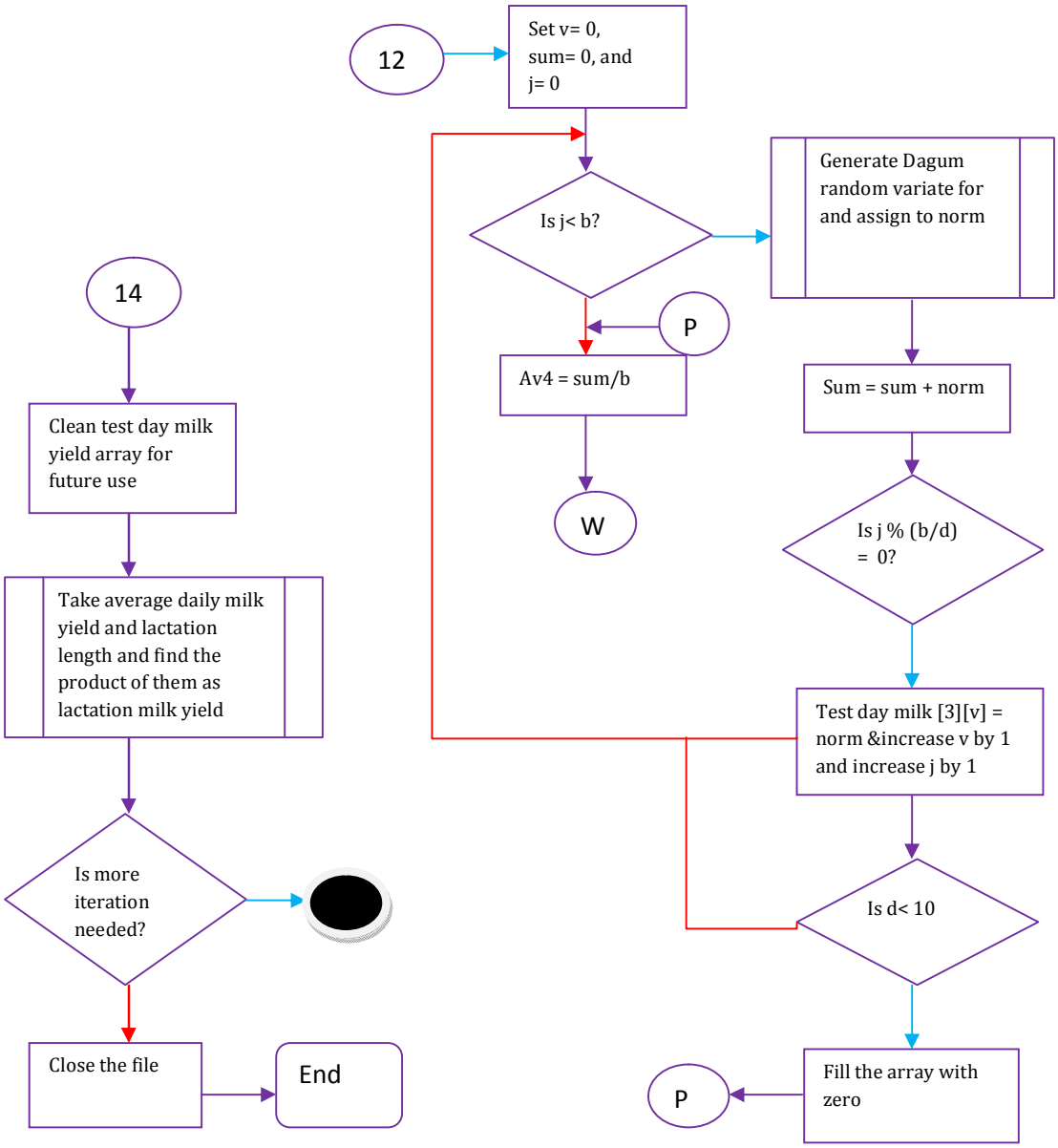

2
Generate calving year greater than previously generated
Write to file
1
Generate calving season
Take m= calving season, y= genetic group, x= herd, and calving year
Is m= 1?
Generate random number between 3 &797and assign to S
3
Is m= 2?
Generate Cauchy random variate and assign to S
3
Generate Laplace random variate and assign to S
3
Is y= 1?
Generate random number between &215 and assign to Q
4
3
Is y= 2?
Generate Log-Logistic random variate and assign to Q
4
Is y= 3?
Is y= 4?
Generate Normal random variate and assign to Q
4
Is year between 1990-1992?
5
Generate Dagum random variate and assign to Q
4
Generate Normal random variate and assign to T


5
Is year between 1993-1995?
Is year between 1996-1998?
Is year between 1999-2001?
Generate Laplace random variate and assign to T
6
Generate random number between 1 &587 and assign to T
6
Is x= 1?
7
Generate Hypersecant random variate and assign to V
Generate random number between 1 &797 and assign to V
7
Compute the average of T, Q, S, V and T
Write to file
Take m= calving season, y= genetic group, parity = k, x= herd, h= calving year and lactation length= b
Generate a random number between 7 and to then assign it to d , define norm, av1, av2, av3, av4 and av5
8


8
Is y= 1?
Set v= 0,
sum= 0, and
j= 0
9
Is y= 2 or y= 3
or y= 4 or y= 5?
9
Is j< b?
Generate Pearson type VI
random variate for y= 1,
Dagum for y= 2, and y= 4
Johnson SB for y= 3 and y= 5
and assign to norm
Z
Av1 = sum/b
Sum = sum + norm
Q
Q
Is j % (b/d)
= 0?
Is m= 1 or
m= 2 or m= 3?
10
Test day milk [0][v] =
norm &increase v by 1
and increase j by 1
Is d< 10
O
Z
Fill the array with
zero
Is h between 1990-
1992 or 1993-1995
or 1996-1998 or
1999-2001 or
2002-2004?
11
E


10
Set v= 0, sum= 0, and j= 0
Is j< b?
Generate Dagum random variate for m= 1, Johnson SB for m= 2 & random number between 0, 18 for m= 3 and assign to norm
X
Av2 = sum/b
Sum = sum + norm
O
Is j % (b/d) = 0?
Test day milk [1][v] = norm &increase v by 1 and increase j by 1
Is d< 10
Fill the array with zero
X


E
Is x= 1 or x= 2?
12
W
Is k= 1 or 2 or 3 or 4 or 5 or 6 or greater than 7?
13
Compute the average of av1, av2, av3 and av4.
Write to file
11
Set v= 0, sum= 0, and j= 0
Is j< b?
C
Av3 = sum/b
E
Generate Dagum random variate for 1990-1992, normal for 1993-1995 else gen. Pareto and assign to norm
Sum = sum + norm
Is j % (b/d) = 0?
Test day milk [2][v] = norm &increase v by 1 and increase j by 1
Is d< 10
Fill the array with zero
C
Sum test day milk yield array column wise and find the sum after that find the sum and multiply it by 30.5
Write to file
14


12
Set v= 0,
sum= 0, and
j= 0
Is j< b?
Generate Dagum
random variate for
and assign to norm
P
Av4 = sum/b
Sum = sum + norm
W
Is j % (b/d)
= 0?
14
Clean test day milk
yield array for
future use
Take average daily milk
yield and lactation
length and find the
product of them as
lactation milk yield
Test day milk [3][v] =
norm &increase v by 1
and increase j by 1
Is more
iteration
needed?
Is d< 10
Close the file
End
P
Fill the array with
zero

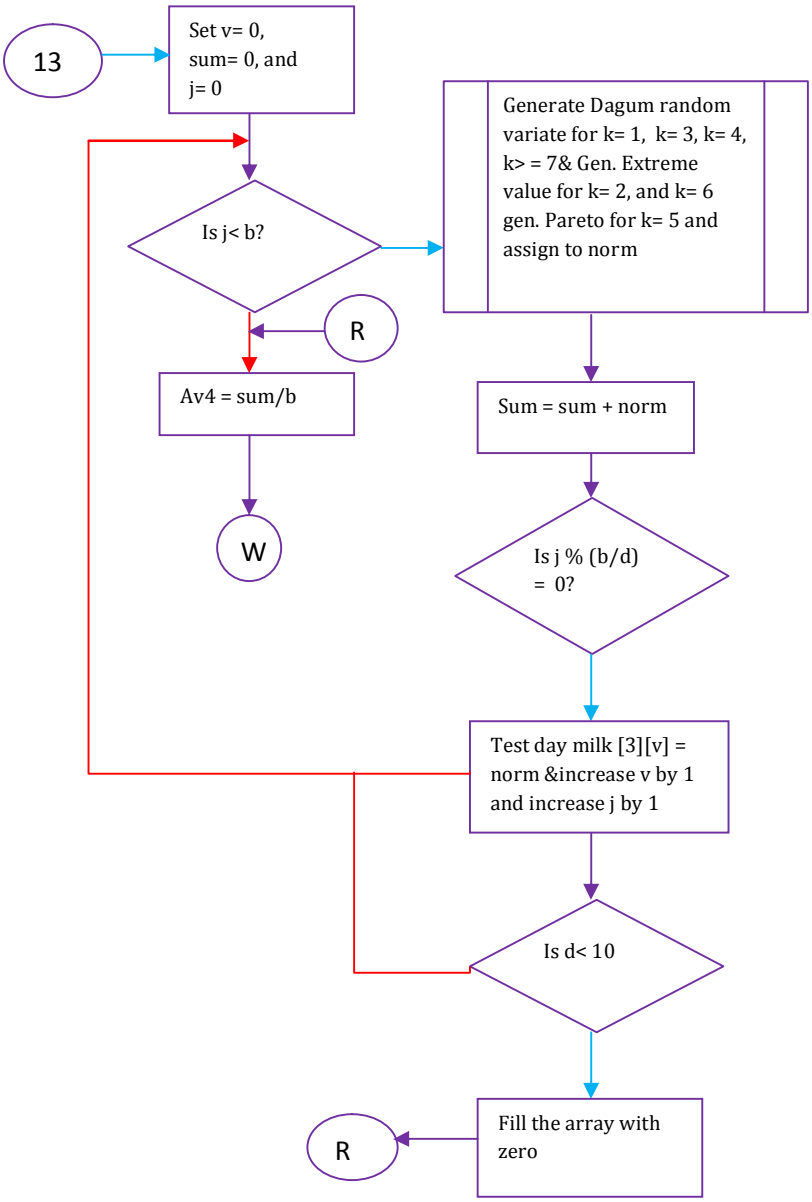


Figure 3.1: Monte Carlo Simulation model

3.3 Bootstrap Simulation

Bootstrap simulation was introduced by Efron in 1970 for the purpose of estimating confidence intervals for a parameter using numerical methods. Assume that we have a dataset with *n* data points. As defined by Efron and Tibshirani (1993), Bootstrap simulation is based upon drawing multiple random samples, each of size *n*, with replacement, from an empirical distribution F. This approach is referred as *re-sampling.* Each random sample of size *n* is referred as a *bootstrap sample*.

If the original dataset is $X = (X_1, X_2, \cdots, X_n)$, then probability of sampling any discrete value within the dataset is $1/n$. A random sample of size *n* from the original dataset is denoted by $X^* = (X_1^*, X_2^*, \cdots, X_n^*)$.

The asterisks indicate that X^* is not the actual dataset X, but rather a randomized or re-sampled version of it. The re-sampled data describe an empirical distribution,

$$\hat{F} \rightarrow (X_1^*, X_2^*, \cdots, X_n^*) \qquad [3.37]$$

Since the sampling is done with replacement, it is possible to have repeated values within any given bootstrap sample.

Among the four variants of bootstrap re-sampling methods, the nonparametric one was used in this study. The particular contributions of Efron (1982, 1983) have realized the possibility of simulation without parametric models. Different authors have been trying to improve the efficiency of simulation in Efron's nonparametric bootstrap method. One idea advanced was that of performing a *balanced bootstrap simulation*; that is, one that reuses each of the sample observations exactly equally often. Davison *et al.* (1986) demonstrated that such balancing can yield sizable gains in terms of bias and variance reduction over the usual bootstrap.

3.3.1 Balanced Bootstrap Simulation Algorithm

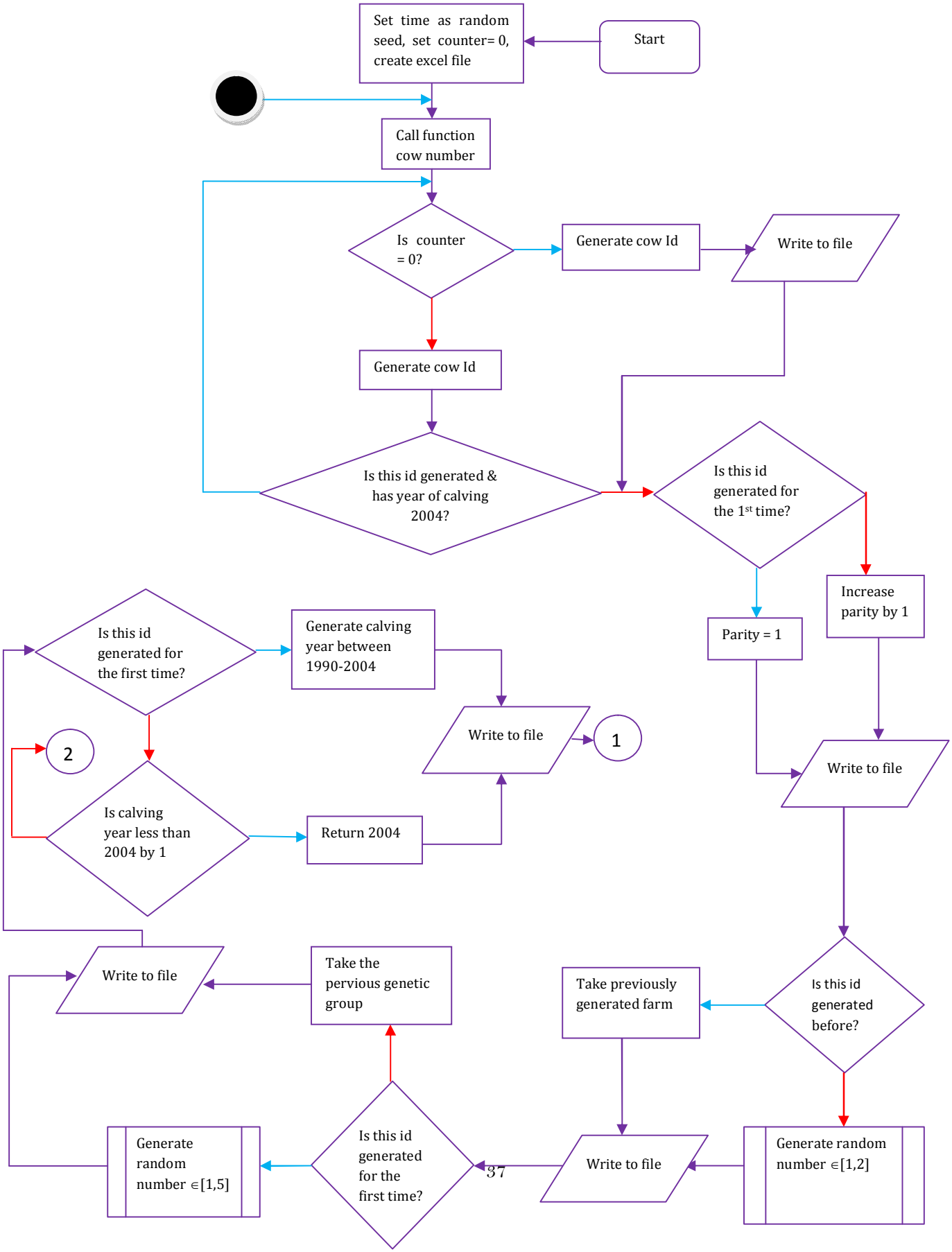
Denoting the sample by $X = (X_1, X_2, \cdots, X_n)$ and the indexes of its elements by $I = \{1, 2, \cdots, n\}$. Let B be the number of bootstrap replications. Suppose also that $Rand(a, b)$ is a function that returns a pseudorandom integer, uniformly distributed on $[a, b]$. An algorithm will be described in pseudo code, with the symbols ← and ↔ denoting assignment and exchange of two values, respectively.

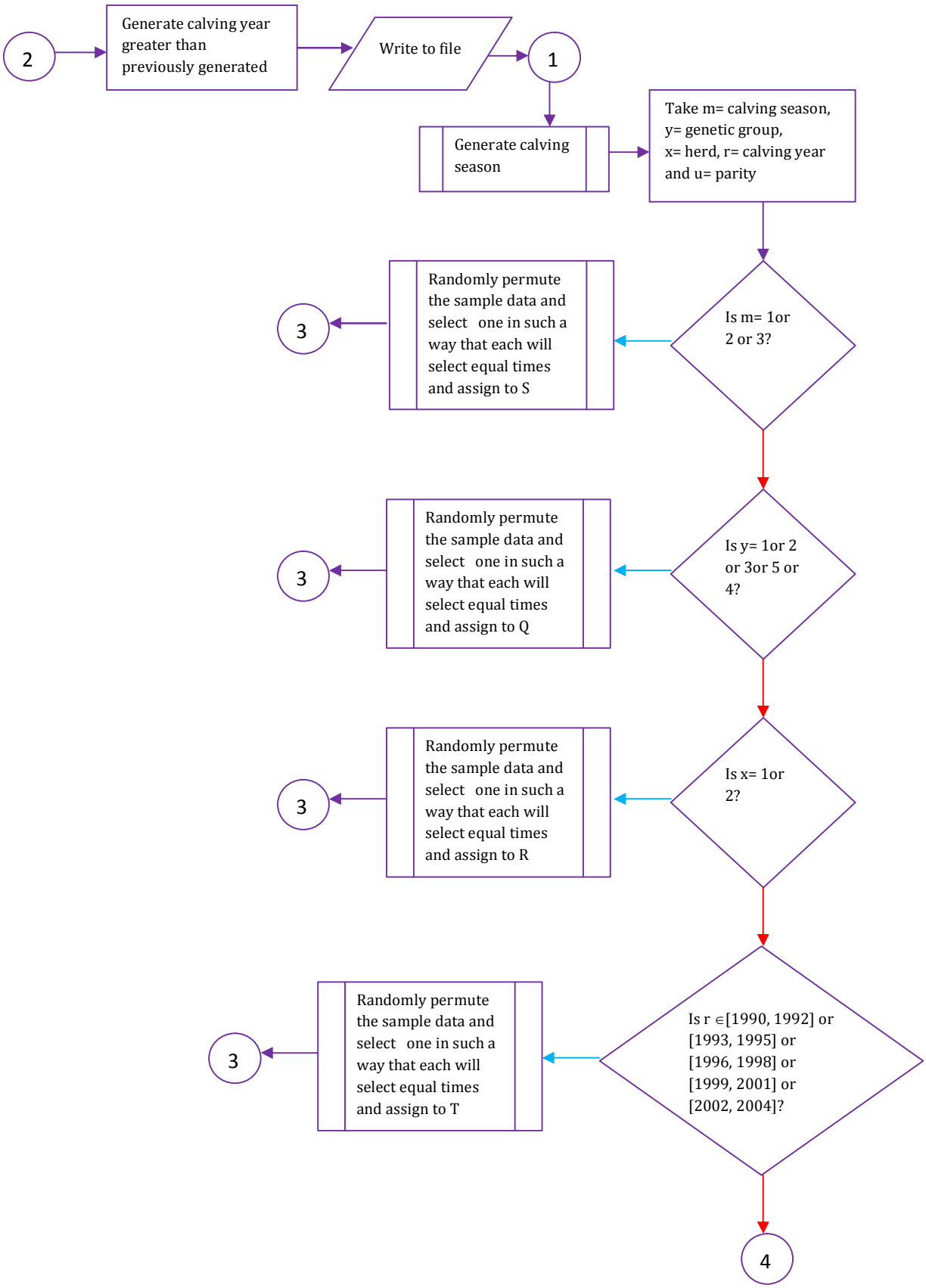
The balanced bootstrap consisting of forcing l $(l = j,\ j \in I)$to assume each of its values exactly B times during the nB replications. Davison *et al.* (1986) pointed out that the balanced bootstrap is, in principle, easy to execute: Concatenate B copies of I to form a list L of length nB. Then randomly permute L and return as bootstrap sample $\{X_j\}$ corresponding to successive sets of n integers from L.

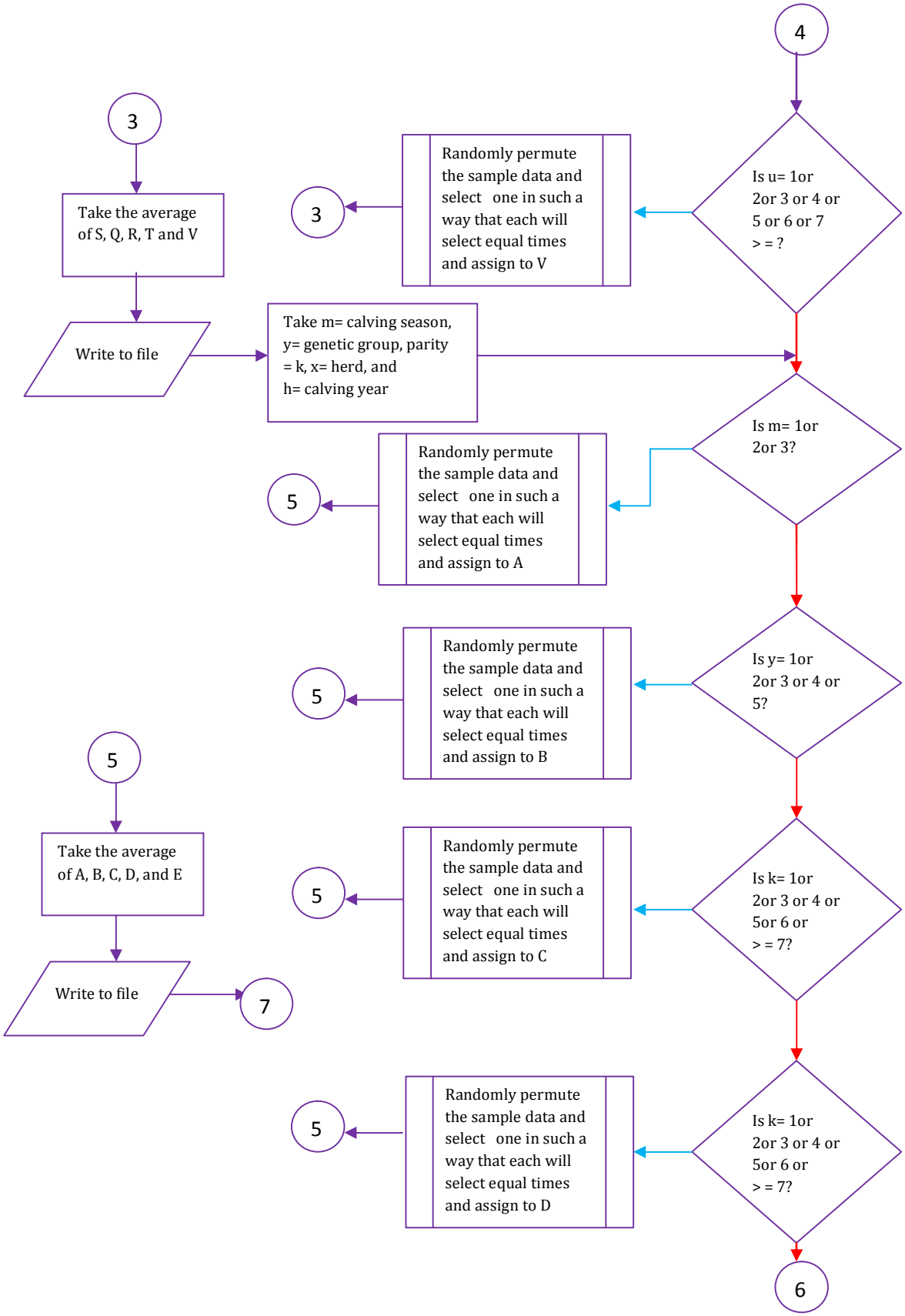
Algorithm BB1:

1. Form the list $L = \{I, I, \cdots, I\}$
2. $FOR\ i \leftarrow n * B$ DOWNTO 2 DO
   $j \leftarrow Rand(1, i)$;
   $L_j \leftrightarrow L_i$;
   END FOR

3.3.2 Bootstrap Simulation Model




2
Generate calving year greater than previously generated
Write to file
1
Generate calving season
Take m= calving season, y= genetic group, x= herd, r= calving year and u= parity
Is m= 1or 2 or 3?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to S
3
Is y= 1or 2 or 3or 5 or 4?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to Q
3
Is x= 1or 2?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to R
3
Is r ∈[1990, 1992] or [1993, 1995] or [1996, 1998] or [1999, 2001] or [2002, 2004]?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to T
3
4


4
Is u= 1or 2or 3 or 4 or 5 or 6 or 7 > = ?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to V
3
3
Take the average of S, Q, R, T and V
Write to file
Take m= calving season, y= genetic group, parity = k, x= herd, and h= calving year
Is m= 1or 2or 3?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to A
5
Is y= 1or 2or 3 or 4 or 5?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to B
5
Is k= 1or 2or 3 or 4 or 5or 6 or > = 7?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to C
5
Is k= 1or 2or 3 or 4 or 5or 6 or > = 7?
Randomly permute the sample data and select one in such a way that each will select equal times and assign to D
5
6
5
Take the average of A, B, C, D, and E
Write to file
7

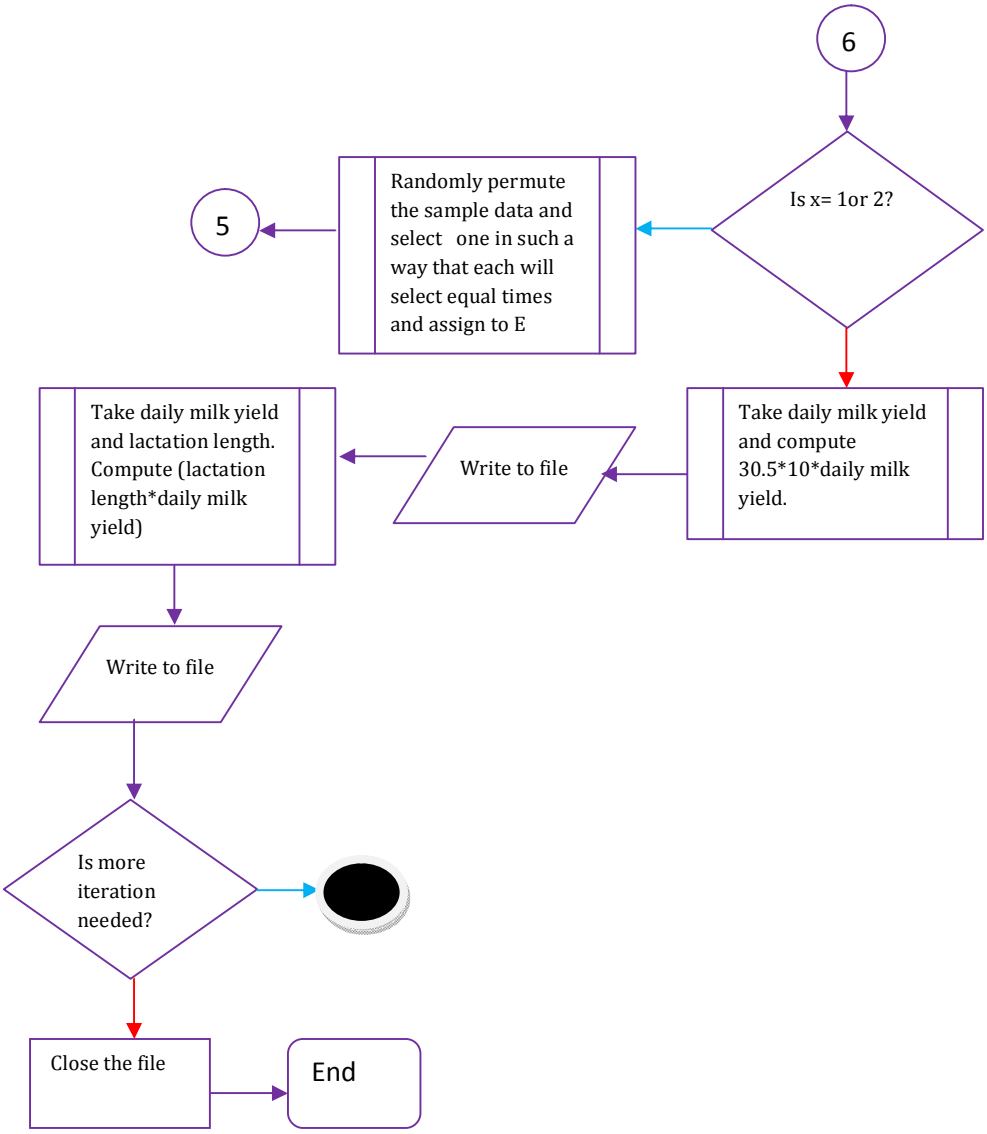


Figure 3.2: Bootstrap Simulation model

3.4 Number of Replications Needed

The big concern after the program of simulation written is answering how many replications are needed to have a good estimator and good approximation.

The formula for the required number of replication (N), given m initial replications is presented as follows:

$$N(m) = \left(\frac{S(m) t_{m-1,\alpha/2}}{\bar{X}(m)\varepsilon} \right)^2 \quad [3.38]$$

Where,

$N(m)$ is the number of replications required, given m replications

$\bar{X}(m)$ is the estimated mean from m simulation run

$S(m)$ is the estimated of the real standard deviation from m simulation run

ε is allowable percentage error of the estimate $\bar{X}(m)$

Some additional simplifications are used to be able to calculate a fixed number $N(m)$ based on initial estimates of m and s, namely, the initial estimates $\bar{X}(m)$ and $S(m)$ based on m observations are close enough to the real mean and standard deviation, and thus do not change much when the number of replications is increased to $N(m)$.

3.5 Method of Comparison

3.5.1 Linear Mixed Model

A linear mixed model is an abstract representation of the real world by using fixed and random effects as its component. Fixed effects are those components which are attributable to a limited set of levels of factors. By default covariates are fixed effects in the mixed model. Random effects are effects whose factor levels are randomly selected from a set of all possible levels of a factor. Furthermore, if the interaction contains either one or two random effects, then the interaction treated as random effect. In contrast, if both factors built the interaction up are fixed, then the interaction is also treated as fixed effect (Searle 1997, 2006a; Searle *et al.* 2006b).

We consider the linear mixed model

$$\boldsymbol{y} = \boldsymbol{X\beta} + \boldsymbol{Zu} + \boldsymbol{\varepsilon} \quad [3.39]$$

where

$\boldsymbol{y}$ is a column vector of N observations,

$\boldsymbol{\beta}$ is a column vector of N_f fixed effects including covariates,

$\boldsymbol{X}$ is the $N \times N_f$ design matrix for fixed effects,

$\boldsymbol{u}$ is a column vector of N_u random effects,

$\boldsymbol{Z}$ is the $N \times N_f$ design matrix for random effects, and

$\boldsymbol{\varepsilon}$ is the column vector of N errors

For the (co) variance structure of $\boldsymbol{y}$ the assumptions are $var(\boldsymbol{u}) = \boldsymbol{G}$, $var(\boldsymbol{\varepsilon}) = \boldsymbol{R}$, $cov(\boldsymbol{u}, \boldsymbol{\varepsilon}) = \boldsymbol{0}$ and $var(\boldsymbol{y}) = \boldsymbol{V} = \boldsymbol{ZGZ'} + \boldsymbol{R}$.

For our data vector $\boldsymbol{y} \sim N(\boldsymbol{X\beta}, \boldsymbol{V})$, the likelihood function which is a function of $\boldsymbol{\beta}$ and $\boldsymbol{V}$ is

$$\boldsymbol{L} = \boldsymbol{L}(\boldsymbol{\beta}, \boldsymbol{V}|\boldsymbol{y}) = \frac{exp\left\{-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{X\beta})'\boldsymbol{V}^{-1}(\boldsymbol{y} - \boldsymbol{X\beta})\right\}}{(2\pi)^{\frac{1}{2}N}|\boldsymbol{V}|^{\frac{1}{2}}} \qquad [3.40]$$

Maximization of $\boldsymbol{L}$ can be achieved by maximizing the logarithm of $\boldsymbol{L}$ of [3.40] which shall be denoted by $\boldsymbol{l}$:

$$\boldsymbol{l} = \log \boldsymbol{L} = -\frac{1}{2}N \log 2\pi - \frac{1}{2}\log|\boldsymbol{V}| - \frac{1}{2}(\boldsymbol{y} - \boldsymbol{X\beta})'\boldsymbol{V}^{-1}(\boldsymbol{y} - \boldsymbol{X\beta}) \qquad [3.41]$$

To maximize $\boldsymbol{l}$, we differentiate [3.41], with respect of $\boldsymbol{\beta}$

$$\boldsymbol{l}_{\boldsymbol{\beta}} = \frac{\partial \boldsymbol{l}}{\partial \boldsymbol{\beta}} = \boldsymbol{X'V^{-1}y} - \boldsymbol{X'V^{-1}X\beta} \qquad [3.42]$$

A general principle of maximizing $\boldsymbol{l}$ with respect to $\boldsymbol{\beta}$ is to equate [3.42] to zero and solve the resulting equation for $\boldsymbol{\beta}$. Let $\widehat{\boldsymbol{\beta}}$ be the maximum likelihood estimator of $\boldsymbol{\beta}$.

$$\boldsymbol{X'V^{-1}X}\widehat{\boldsymbol{\beta}} = \boldsymbol{X'V^{-1}y} \qquad [3.43]$$

$$\widehat{\boldsymbol{\beta}} = (\boldsymbol{X'V^{-1}X})^{-1}\boldsymbol{X'V^{-1}y} \qquad [3.44]$$

Recently Maximum Likelihood (ML) and related procedures to estimate variance components for unbalanced data have become popular. Restricted Maximum Likelihood (REML), developed

by Patterson and Thompson (1971), which in contrast to ML accounts for the loss in degree of freedom due to fitting fixed effects, and has become accepted as the preferred method to estimate variance components for animal experiments.

To account for the loss in degrees of freedom due to fitting of fixed effects, REML, in contrast to ML, maximizes only part of the likelihood of the data vector $\boldsymbol{y}$ which is independent of the fixed effects. That is, rather than using $\boldsymbol{y}$ (the data vector) directly, REML is based on the linear combinations of elements of $\boldsymbol{y}$, chosen in such a way that those combinations don't contain any fixed effects, no matter what their value. These linear combinations turn out to be equivalent to residuals obtained after fitting the fixed effects. This results from starting with a set of values $\boldsymbol{K'y}$ where vectors $\boldsymbol{K'}$ are chosen so that

$$\boldsymbol{K'y} = \boldsymbol{K'X\beta} + \boldsymbol{K'Zu} + \boldsymbol{k'\varepsilon} \qquad [3.45]$$

contains no term in $\boldsymbol{\beta}$, i.e., so that

$$\boldsymbol{K'X\beta} = \boldsymbol{0} \qquad \forall\boldsymbol{\beta} \qquad [3.46]$$

Hence

$$\boldsymbol{K'X} = \boldsymbol{0} \qquad [3.47]$$

Therefore, the form of $\boldsymbol{K}$ must be $\boldsymbol{K'} = \boldsymbol{c'}(\boldsymbol{I} - \boldsymbol{XX}^{+})$ or

$$\boldsymbol{K'} = \boldsymbol{c'}[\boldsymbol{I} - \boldsymbol{X}(\boldsymbol{X'X})^{-}\boldsymbol{X'}] = \boldsymbol{c'}(\boldsymbol{I} - \boldsymbol{XX}^{+}) = \boldsymbol{c'M} \qquad [3.48]$$

for any $\boldsymbol{c'}$ and where

$$\boldsymbol{X}^{+} = (\boldsymbol{X'X})^{-}\boldsymbol{X'} \text{ and } \boldsymbol{M} = \boldsymbol{I} - \boldsymbol{XX}^{+} \qquad [3.49]$$

With $\boldsymbol{y} \sim N(\boldsymbol{X\beta}, \boldsymbol{V})$ we have, for $\boldsymbol{K'X} = 0, \boldsymbol{K'y} \sim N(\boldsymbol{0}, \boldsymbol{K'VK})$. The REML equations can therefore be derived from the ML equation of [3.40] by replacing $\boldsymbol{y}$ by $\boldsymbol{K'y}$, $\boldsymbol{X}$ by $\boldsymbol{K'X}$, $\boldsymbol{Z}$ by $\boldsymbol{K'Z}$ and $\boldsymbol{V}$ by $\boldsymbol{K'VK}$.

Then the logarithm of the likelihood function of the REML profile is

$$\boldsymbol{l_p} = -\frac{1}{2}\log|\boldsymbol{V}| - \frac{1}{2}\log|\boldsymbol{X'V}^{-1}\boldsymbol{X}| - \frac{1}{2}\boldsymbol{yPy} - \frac{N - rank(\boldsymbol{X})}{2}\log 2\pi \qquad [3.50]$$

where

$$\boldsymbol{P} = \boldsymbol{V}^{-1} - \boldsymbol{V}^{-1}\boldsymbol{X}(\boldsymbol{X}'\boldsymbol{V}^{-1}\boldsymbol{X})^{-}\boldsymbol{X}'\boldsymbol{V}^{-1} \qquad [3.51]$$

The next step is differentiating $\boldsymbol{l_p}$ with respect to $\boldsymbol{V}$. Let φ be the arbitrary element of $\boldsymbol{V}^{-1}$ which needs to be estimated. Mathematically,

$$\frac{\partial \boldsymbol{l_p}}{\partial \varphi} = 0, \qquad \forall \varphi \epsilon \boldsymbol{V} \qquad [3.52]$$

An iterative method must be used to find estimates for φ. The conventional optimization methods, which require first and second derivatives, may be applied. Thus, Newton-Raphson and Fisher scoring will be used.

Best linear unbiased predictor (BLUP) is used in linear mixed model to predict the random effects. By taking model in [3.39], with $E(\boldsymbol{u}) = \boldsymbol{0}$ we consider the problem of predicting

$$\boldsymbol{w} = \boldsymbol{L}'\boldsymbol{\beta} + \boldsymbol{u} \qquad [3.53]$$

for some known matrix $\boldsymbol{L}'$, such that $\boldsymbol{L}'\boldsymbol{\beta}$ is estimable; i.e., $\boldsymbol{L}' = \boldsymbol{T}'\boldsymbol{X}$ for some $\boldsymbol{T}$. Since $\boldsymbol{w}$ involves both fixed effects and random variables, there might be a debate as to whether we should estimate $\boldsymbol{w}$ or predict $\boldsymbol{w}$. We will predict $\boldsymbol{w}$, and will choose $\widetilde{\boldsymbol{w}}$ as a predictor to have three properties: Best in a sense that minimize

$$E\{(\widetilde{\boldsymbol{w}} - \boldsymbol{w})'\boldsymbol{A}(\widetilde{\boldsymbol{w}} - \boldsymbol{w})\} \qquad [3.54]$$

where $\boldsymbol{A}$ is any positive definite symmetric matrix, linear in y : $\widetilde{\boldsymbol{w}} = \boldsymbol{a} + \boldsymbol{B}\boldsymbol{y}$, with $\boldsymbol{a}$ and $\boldsymbol{B}$ not involving $\boldsymbol{\beta}$. Unbiased: $E(\widetilde{\boldsymbol{w}}) = E(\boldsymbol{w})$

The resulting predictor is a best linear unbiased predictor (BLUP). Note that unbiasedness is now a criterion of the prediction procedure and not just a by-product of it. Introducing it as a criterion arises from the presence of $\boldsymbol{\beta}$ in [3.53].

It is clear that $E(\boldsymbol{w}) = \boldsymbol{L}'\boldsymbol{\beta}$. We then have

$$\begin{bmatrix}\boldsymbol{w}\\ \boldsymbol{y}\end{bmatrix} \sim \left\{\begin{bmatrix}\boldsymbol{L}'\boldsymbol{\beta}\\ \boldsymbol{X}\boldsymbol{\beta}\end{bmatrix}, \begin{bmatrix}\boldsymbol{D} & \boldsymbol{C}\\ \boldsymbol{C}' & \boldsymbol{V}\end{bmatrix}\right\} \qquad [3.55]$$

The unbiasedness required of $\tilde{\boldsymbol{w}}$ demands that $\boldsymbol{a} + \boldsymbol{BX\beta} = \boldsymbol{L'\beta}$ for all $\boldsymbol{\beta}$, and if $\boldsymbol{a}$ is not to depend on $\boldsymbol{\beta}$ then $\boldsymbol{a} = \boldsymbol{0}$ and $\boldsymbol{BX} = \boldsymbol{L'}$. Consequently, the predictor is $\tilde{\boldsymbol{w}} = \boldsymbol{By}$ and in $\boldsymbol{w} = \boldsymbol{L'\beta} + \boldsymbol{u}$ the term $\boldsymbol{L'\beta}$ is estimable function of $\boldsymbol{\beta}$ in the model $E(\boldsymbol{y}) = \boldsymbol{X\beta}$. This limits the form of $\boldsymbol{L'}$ in $\boldsymbol{w}$, but it is obviously a reasonable limitation and the predictor is called BLUP. It is

$$BLUP(\boldsymbol{w}) = \tilde{\boldsymbol{w}} = \boldsymbol{L'}\hat{\boldsymbol{\beta}} + \boldsymbol{CV}^{-1}(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}) \quad [3.56]$$

$$BLUE(\boldsymbol{X\beta}) = \boldsymbol{X}\hat{\boldsymbol{\beta}} = \boldsymbol{X}(\boldsymbol{X'V}^{-1}\boldsymbol{X})^{-}\boldsymbol{X'V}^{-1}\boldsymbol{y} \quad [3.57]$$

We know that $\tilde{\boldsymbol{u}} = \boldsymbol{\mu_u} + \boldsymbol{CV}^{-1}(\boldsymbol{y} - \boldsymbol{\mu_y})$. When $E(\boldsymbol{u}) = \boldsymbol{0} = \boldsymbol{\mu_u}$ and $E(\boldsymbol{y}) = \boldsymbol{\mu_y} = \boldsymbol{X\beta}$, then $\tilde{\boldsymbol{u}} = \boldsymbol{CV}^{-1}(\boldsymbol{y} - \boldsymbol{X\beta})$ but let us replace $\boldsymbol{X\beta}$ by its $BLUE(\boldsymbol{X\beta}) = \boldsymbol{X}\hat{\boldsymbol{\beta}}$. As a result,

$$BLUP(\boldsymbol{u}) = \tilde{\boldsymbol{u}} = \boldsymbol{CV}^{-1}(\boldsymbol{y} - \boldsymbol{X}\hat{\boldsymbol{\beta}}) \quad [3.58]$$

To emphasize this we write the predictor as

$$\tilde{\boldsymbol{w}} = \boldsymbol{L'}\hat{\boldsymbol{\beta}} + \tilde{\boldsymbol{u}} \quad [3.59]$$

$\tilde{\boldsymbol{w}}$ is thus the sum of BLUE of $\boldsymbol{L'\beta}$ and the BLUP of $\boldsymbol{u}$.

3.5.1.1 Model Building Strategy

A primary goal of model selection is to choose the simplest model that provides the best fit to the observed data. The process of building a LMM for a given set of clustered data is an iterative process that requires a series of model fitting steps and investigations, and selection of appropriate mean and covariance structures for the observed data.

Model building typically involves a balance of statistical and subject matter considerations; there is no single strategy that applies to every application. One of the most widely used model building strategies is the top-down (West *et al.* 2007; Fernandez 2007).

This strategy begins with a model that includes the maximum number of fixed effects that we wish to consider in a model.

1. *Start with a well-specified mean structure for the model:* This step typically involves adding the fixed effects as many covariates (and interaction between the covariates) as

possible to the model to make sure that the systematic variation in the response is well explained before investigating various covariance structures to describe random variation in the data.

2. *Select a structure for the random effects in the model:* This step involves the selection of random effects to include in the model. The need for including the selected random effects can be tested by performing REML-based likelihood ratio tests for the associated covariance parameters.
3. *Select a covariance structure for the residuals in the model:* Once fixed effects and random effects have been added to the model, the remaining variation in the observed responses is due to residual error, and an appropriate covariance structure for the residuals should be investigated.
4. *Reduce the model:* This step involves using appropriate statistical tests to determine whether certain fixed-effect parameters are needed in the model. At this stage we use Akaike information criterion (AIC). That is, AIC may be calculated based on the (ML or REML) log-likelihood of the fitted model.

$$AIC = -2l\left(\widehat{\boldsymbol{\beta}}, \widetilde{\boldsymbol{U}}\right) + 2p \qquad [3.60]$$

where p represents the total number of parameters being estimated in the model for both the fixed and random effects. The model having smallest AIC is best.

3.6 Influence Measures

As Littell *et al.* (2006) disclosed the goal of influence analysis is to determine how observations impact the analysis. Of course, all observations are influential, in the sense that their removal changes the numerical results of the analysis. The goal is to identify those observations that are so influential that their presence or absence from the data changes an important aspect of the analysis, yields qualitatively different inferences, or violates of assumptions of the model. The goal is not to determine the observations for removal from the analysis[19], but to decide prior to a perturbation analysis what matters (Schabenberger 2004).

[19] If removing a data point changes considerably the estimate of the residual variance but not the fixed effect solutions, and the goal of the analysis is prediction, the data point does not present a problem. If, however, you

Overall Influence

An overall influence statistic measures the global impact on the model, by quantifying a change in the objective function. The basic concept is to do the following:

1. Removing an observation or group of observations
2. Compute the parameter estimates for the reduced dataset
3. Assess the height of the original (restricted) likelihood surface at the delete-data parameter estimate

Influence on the Parameter Estimates

A common way to measure the impact on a vector of parameter estimates is to compute a quadratic form in the difference between the full data and reduced data estimates. Cook's D statistic is of this nature. In general, it can be expressed as

$$D(\boldsymbol{\beta}) = \frac{\left(\widehat{\boldsymbol{\beta}} - \widehat{\boldsymbol{\beta}}_U\right)' Var\left(\widehat{\boldsymbol{\beta}}\right)^{-} \left(\widehat{\boldsymbol{\beta}} - \widehat{\boldsymbol{\beta}}_U\right)}{rank(\boldsymbol{X})} \qquad [3.61]$$

where the subscript U denotes the estimates after deleting observations in the set U (to allow multiple point deletion).

A closely related statistic is the multivariate DFFITS statistic

$$MDFFITS(\boldsymbol{\beta}) = \frac{\left(\widehat{\boldsymbol{\beta}} - \widehat{\boldsymbol{\beta}}_U\right)' Var\left(\widehat{\boldsymbol{\beta}}\right)^{-} \left(\widehat{\boldsymbol{\beta}} - \widehat{\boldsymbol{\beta}}_U\right)}{rank(\boldsymbol{X})} \qquad [3.62]$$

The primary difference between the two statistics is that the variance matrix of the fixed effects is based on an externalized estimate that does not involve the observations in the set *U*. The idea of *D* and *MDFFITS* can be applied to the covariance parameters as well. For either statistic, influence increases with the magnitude of the statistics.

---

are also interested in confidence interval for the predicted values, the influence on the residual variance estimate matters.

Influence on the Precision of estimates

D is a quadratic form in the difference of the full data and reduced data parameter estimates. To contrast the change in the precision, we need to engage not the vectors themselves, but their covariance matrices. The most common way to do this is through trace and determinant operation. These lead to the COVTRACE and COVRATIO statistics.

$$COVTRACE(\boldsymbol{\beta}) = \left| trace\left(\widehat{Var}(\widehat{\boldsymbol{\beta}})^{-} \widehat{Var}(\widehat{\boldsymbol{\beta}}_U) - rank(\boldsymbol{X}) \right) \right| \quad [3.63]$$

$$COVRATIO(\boldsymbol{\beta}) = \frac{\left| \widehat{Var}(\widehat{\boldsymbol{\beta}}_U) \right|}{\left| \widehat{Var}(\widehat{\boldsymbol{\beta}}) \right|} \quad [3.64]$$

The reason behind the COVTRACE statistic is that $\boldsymbol{X}$ is a positive semi-definite matrix, and then $trace\ (\boldsymbol{X}^{-}\boldsymbol{X})$ is equal the rank of $\boldsymbol{X}$. If the variance parameter estimates is not affected by the removal of observations, then the trace in the COVTRACE statistic should equal the rank of $\boldsymbol{X}$. The yardstick for lack of influence is thus a value of zero. As a result if COVTRACE statistic increases, so does the influential on the precision of the parameter estimates and linear function hereof, such as tests of fixed effects.

The COVRATIO statistic relates the determinants of the covariance matrices of the full data and reduced data estimates. The yardstick of no influence is a value of one. Values larger than one indicate increased precision in the full-data case, and values smaller than one indicate higher precision for the reduced data estimates. Such an interpretation in terms of increase or decrease in precision is not possible with the COVTRACE statistic. The disadvantage of the COVRATIO statistic is primarily computational. When the $\boldsymbol{X}$ matrix is of less than full rank, special manipulations are required to compute the determinants of the non-singular components of the variance matrices.

Once we have checked the influential analysis for the two datasets, the next step is residual analysis to check the assumption that random effects and error term follow normal distribution.

3.7 Residual Analyses

In the mixed model of form [3.41] with the random effects and errors, the conditional distribution of $\boldsymbol{Y|u}$ has mean $\boldsymbol{X\beta + Zu}$ and variance $\boldsymbol{R}$. The marginal distribution of $\boldsymbol{Y}$ has mean $\boldsymbol{X\beta}$ and variance $\boldsymbol{ZGZ' + R}$. Consequently, we define the conditional residuals as

$$\boldsymbol{r_c = Y - X\hat{\beta} - Z\tilde{u}} \qquad [3.65]$$

that measures the deviations from the conditional mean and the marginal residuals as

$$\boldsymbol{r_m = Y - X\hat{\beta}} \qquad [3.66]$$

that measures the deviations from the overall mean. Residual analyses indicate the outliers, homoscedasticity, normality etc. SAS is used to analyze the linear mixed model.

4 Findings and Discussion

4.1 Leveling of Variables as Random and Fixed

In some situations, it is believed that the decision as to whether certain variables are fixed or random is not immediately obvious. In endeavoring to decide whether a set of effects is fixed or random, the context of the data, the manner in which they were gathered and the environment from which they come are determinant factors. In considering these points the important question is that of inference: are inferences going to be drawn from these data about just these levels of the factor? *Yes*, then the variables are to be considered as fixed effects. *No*, then, presumably, inference will be made not just about the levels occurring in the data but about some population of levels of the variable from which those in the data are presumed to have come: and so the variables are considered as being random (West *et al*. 2007; Searle 1997)

Thus, when inferences are going to be confined to the effects in the model, the effects are considered as fixed and when inferences will be made about a population of effects from which those in the data are considered to be a random sample, the variables are considered as random.

Variables	Random	Fixed
Number of Parity		√
Herd or Farm	√	
Season of Calving		√
Period of Calving		√
Daily Milk Yield		√
Lactation Length		√
Lactation Milk Yield	√	
Genetic Group	√	

Table 4.1: Leveling of effects as random and fixed

4.2 Model Fitting and Diagnostic Checking

This study has two datasets namely, a Monte Carlo dataset generated by Monte Carlo simulation and a bootstrap dataset generated based on bootstrap simulation. Since we need to fit a linear mixed model for these datasets, we must first fit the model and then check whether they satisfy the basic assumptions of linear mixed model (Appendix E and D).

Based on the concept stated in section 3.4.1.1 of chapter three, the final model selected as best for the two datasets is

$$LM_{ijklm} = \mu + F_i + S_j + C_k + GG_l + \beta_1 P_m + \beta_2 D_m + \beta_3 T_m + \beta_4 LL_m + \beta_5 (PS)_{mj} + (GGD)_{lm} + (LLGG)_{ml} + (GGP)_{lm} + \varepsilon_{ijklm} \quad [4.1]$$

$k, l = 1, 2, \cdots, 5$, $i = 1, 2$ and $j = 1, 2, 3$, $m = 1, 2, \cdots, b$ where b is the size of the respective datasets, μ is the general mean, F_i is the i^{th} farm effect, C_k is the k^{th} period effect, GG_l is the effect of the l^{th} genetic group, S_j is the effect of the j^{th} season. D_m, P_m, LL_m are the m^{th} daily milk yield, parity and lactation length, respectively. $(PS)_{mj}$ is the interaction effect of the m^{th} parity and the j^{th} season. $(GGD)_{lm}$ is the interaction effect of the l^{th} genetic group and the m^{th} daily milk yield. $(LLGG)_{ml}$ is the interaction effect of the l^{th} genetic group and the m^{th} lactation length. $(GGP)_{lm}$ is the interaction effect of the l^{th} genetic group and the m^{th} parity. ε_{ijklm} is the random disturbance term. LM_{ijklm} is the m^{th} lactation milk yield of the i^{th} farm, the k^{th} period, the j^{th} season and the l^{th} genetic group.

Table 4.2: Goodness-of-fit test for the Monte Carlo simulation dataset

Source	DF	Squares	Mean square	F value	Pr>F
Model	11	646999736.3	58818157.8	20903.4	<0.0001
Error	3321	9344651.7	2813.8		
Corrected total	3332	656344387.9			

Table 4.3: Goodness-of-fit test for the bootstrap simulation dataset

Source	DF	Squares	Mean square	F value	Pr>F
Model	11	705843584.8	64167598.6	21019.9	<0.0001
Error	3403	10388385.2	3052.7		
Corrected total	33414	716231970.0			

The above tables show that the model fits the data very well.

4.3 Comparisons Among Datasets

The process of comparison is gone in two phases: The first is comparing fixed effect estimates and their significance. The second technique is testing some basic contrasts.

4.3.1 Fixed Effect Estimates

A. Monte Carlo Simulation Dataset

Table 4.4: Fixed effect estimates for Monte Carlo dataset

Effect	Season	Period	Estimate	Standard error	DF	t value	Pr > |t|
Intercept			-1814.97	102.97	1	-17.63	0.0361
Parity			-1.9362	0.9332	3301	-2.07	0.0381
Season	Light rain		-18.7881	5.2639	7	-3.57	0.0091
Season	Main rain		22.3197	7.6812	7	2.91	0.0228
Season	Dry season		0	.	.	.	.
Period		1990-92	-3.2845	3.7346	3301	-0.88	0.3792
Period		1993-95	-1.2386	2.8070	3301	-0.44	0.6591
Period		1996-98	-6.9695	2.5403	3301	-2.74	0.0061
Period		1999-01	-3.6514	2.5284	3301	-1.44	0.1488
Period		2002-04	0	.	.	.	.
Lactation Length			5.8837	0.2630	4	22.37	<0.0001
Daily Milk Yield			309.13	9.3237	4	33.16	0.0003
Parity*Season	Light rain		3.7552	1.0491	3301	3.58	<0.0001
Parity*Season	Main rain		-7.0003	1.2704	3301	-5.51	<0.0001
Parity*Season	Dry season		0	.	.	.	.

Based on the above table, parity has a significant effect on lactation milk yield. Moreover, by holding the other factors' effect constant, lactation milk yield reduced by 1.9362 litters when the number of live-born calves delivered by cow is increased by one.

Lactation length and daily milk yield have a prominent positive effect on lactation milk yield. Furthermore, by holding the effect of other factors constant, lactation milk yield increased by 5.8837 and 309.13 litters as per a unit increment in lactation length and daily milk yield, respectively.

Lactation milk yield is also affected by the interaction of number of live-born calves and season of calving. That is, by holding the effect of other factors constant, a unit increment of live-born calf makes the lactation milk yield raised and abridged by 3.76 and 7 litters in light and main rain seasons, respectively. Moreover, season of calving is significant at 5% and 10% level of significances.

Period of calving is significant at 5% and 10% level of significances (Table 4.5). Specially, the third (1996-1998) period of calving is significant at 10% level of significance.

Table 4.5: Fixed effect tests for Monte Carlo dataset

Effect	Num DF	Den DF	F value	Pr > F
Parity	1	3301	15.35	<0.0001
Season	2	7	7.65	0.0173
Period	4	3301	2.47	0.0429
Lactation length	1	4	500.62	<0.0001
Daily milk yield	1	4	1099.30	<0.0001
Parity*Season	2	3301	34.98	<0.0001

B. Bootstrap Simulation Dataset

Table 4.6: Fixed effect estimates for bootstrap dataset

Effect	Season	Period	Estimate	Standard error	DF	t value	Pr > |t|
Intercept			-1456.20	81.3603	1	-17.90	0.0355
Parity			-0.6815	1.2884	3382	-0.53	0.5969
Season	Light rain		-0.5972	4.5141	8	-0.13	0.8980
Season	Main rain		-5.6003	4.4266	8	-1.27	0.2414
Season	Dry season		0	.	.	.	.
Period		1990-92	-8.0144	3.9104	3382	-2.05	0.0405
Period		1993-95	-3.8829	3.3042	3382	-1.18	0.2400
Period		1996-98	-0.7420	3.0096	3382	-0.25	0.8053
Period		1999-01	-1.3629	2.8501	3382	-0.48	0.6325
Period		2002-04	0	.	.	.	.
Lactation Length			4.8536	0.2050	4	23.67	<0.0001
Daily Milk Yield			300.46	4.7460	4	63.31	<0.0001
Parity*Season	Light rain		0.2371	1.4653	3382	0.16	0.8715
Parity*Season	Main rain		1.0154	1.4451	3382	0.70	0.4823
Parity*Season	Dry season		0	.	.	.	.

Based on the above table, lactation length and daily milk yield are significant and they increase lactation milk yield by 4.8536 and 300.46 litters as per a unit increment of them, holding the effect of other factors constant, respectively.

Even thought period of calving, in general, isn't significant, the first period (1990-1992) of calving is significant at 5% and 10% level of significances.

Table 4.7: Fixed effect tests for bootstrap dataset

Effect	Num DF	Den DF	F value	Pr > F
Parity	1	3382	0.08	0.7841
Season	2	8	0.95	0.4253
Period	4	3382	1.30	0.2661
Lactation length	1	4	560.38	<0.0001
Daily milk yield	1	4	4007.88	<0.0001
Parity*Season	2	3382	0.27	0.7626

4.3.2 Contrast Testing

One of the hypothesis testing is about contrasts.

A. Farm Effect

The null hypothesis for the two datasets is that Debre Zeit and Holeta farm have the same average effect on lactation milk yield. In other words, on average the farm management practices in these two dairy farms have the same effect.

Table 4.8: Farm effect test

Dataset	Label	Num DF	Den DF	F value	Pr>F
Monte Carlo	Farm effect	1	4.47	311.26	<0.0001
Bootstrap	Farm effect	1	4.01	323.35	<0.0001

As we see in the above table, there is a significant difference between the two farms. The farm effect is significant at 5% and 10% of significance levels. The farm management practice in Debre Zeit and Holeta don't have the same effect on lactation milk yield on average. That is, the farm management practices in these farms are the determinant factor for the difference in the average milk yield per lactation.

B. Genetic Group Effect

The null hypothesis is that the five genetic groups have the same average effect on lactation milk yield. In other words, the average yield of milk in a lactation period is the same across the five genetic groups or put differently their percentage of Holstein inheritance doesn't have any effect on average yield of milk per lactation period.

Table 4.9: Genetic group effect

Dataset	Label	Num DF	Den DF	F value	Pr>F
Monte Carlo	Genetic effect	1	38.3	1125.15	<0.0001
Bootstrap	Genetic effect	1	39.6	1450.76	<0.0001

Based on the above table, all five genetic groups don't have the same effect on the average milk yield per lactation period. That is, there are significantly different effects among genetic groups.

As a result, the next interest lies in identifying on the identification which genetic group is significantly different regarding the average milk yield per lactation. Therefore, multiple (pair-wise) comparisons are examined below.

Table 4.10: Multiple comparisons of genetic group for the Monte Carlo dataset

Label	Num DF	Den DF	F value	Pr>F
50% and 62.5% Holstein	1	4.89	240.71	<0.0001
50% and 75% Holstein	1	5.24	124.36	<0.0001
50% and 87.5% Holstein	1	4.87	238.52	<0.0001
50% and 0% Holstein	1	4.76	220.33	<0.0001
62.5% and 75% Holstein	1	5.17	165.31	<0.0001
62.5% and 87.5% Holstein	1	4.79	296.62	<0.0001
62.5% and 0% Holstein	1	4.69	276.33	<0.0001
75% and 87.5% Holstein	1	6.22	381.11	<0.0001
75% and 0% Holstein	1	5.59	176.50	<0.0001
87.5% and 0% Holstein	1	4.8	276.05	<0.0001

Based on the Tables 4.10 and 4.11, all pair-wise genetic group comparison have significant average effects at any significance level.

Table 4.11: Multiple comparisons of genetic group for the bootstrap dataset

Label	Num DF	Den DF	F value	Pr>F
50% and 62.5% Holstein	1	4.55	321.55	<0.0001
50% and 75% Holstein	1	4.53	292.36	<0.0001
50% and 87.5% Holstein	1	4.53	298.42	<0.0001
50% and 0% Holstein	1	4.51	494.30	<0.0001
62.5% and 75% Holstein	1	4.53	274.88	<0.0001
62.5% and 87.5% Holstein	1	4.53	280.73	<0.0001
62.5% and 0% Holstein	1	4.51	471.43	<0.0001
75% and 87.5% Holstein	1	4.49	311.53	<0.0001
75% and 0% Holstein	1	4.43	153.73	<0.0001
87.5% and 0% Holstein	1	4.46	503.99	<0.0001

5 Conclusion

5.1 Conclusion

In the case of Monte Carlo simulation, season and period of calving are significant. The same result was reported in many researches (Goshu 2005; Raheja 1994; Mukherjee 2005). In contrast, season of calving is not significant in case of bootstrap simulation. This result coincides with the work of Haile *et al.* (2009), Gebeyehu *et al.* (2005) and Fayeye and Ayorinde (2010). The work of Haile *et al.* (2009) sows that unlike season of calving, period of calving is significant. Whatever the optimal feeding and management conditions attained and maintained there is always the effect of either season or period of calving on lactation milk yield (Goshu 2005; Million and Tadelle 2003).

Unlike bootstrap simulation, parity is significant in case of Monte Carlo simulation. Such kind of result was reported by Gebeyehu *et al.* (2005), Gebeyehu *et al.* (2007) and Haile *et al.* (2009). In contrast to bootstrap simulation, the interaction effect of parity by season of calving is significant in Monte Carlo simulation. This result was reported by Ray *et al.* (1992).

The lactation length and daily milk yield are significant in both simulation results. The coefficients are almost similar. There is no significant difference between these simulations in this regard (Million and Tadelle 2003).

For contrast testing both simulation techniques equally approximate the real figure. Haile *et al.* (2009) reported that, due to lack of consistent variation in the two herds, farm effect is insignificant. Many research works assure that farm has significant effect on lactation milk yield even under similar and standard farm management practices (Jadhav *et al.* 1991; Mukherjee 2005).

This study shows that there are amazing approximations with the results of both simulations. However, bootstrap simulation comes up with unexpected finding (neglecting environmental effect) that is not supported by previously done works. In contrast, Monte Carlo simulation

produces results which are highly in line with the previously accomplished researches in the field. Therefore, this study reveals that Monte Carlo simulation provides a better approximation to previous works in the field.

References

[1] Alemu, G., Mengistu, A., Sendros, D. and Alemu, T. (1998), "Status of Dairy Research in Ethiopia," *Proceedings of the Smallholder Dairy Development of Project,* Addis Ababa: Ministry of Agriculture, pp. 73-81.

[2] Banks, J. (ed) (1998), *Handbook of simulation: principles, methodology, advances, applications, and practice,* Canada: Wiley.

[3] Box, G.E.P and Muller, M.E. (1958), "A Note on the Generation of Random Normal deviates," *Ann. Math. Statist.* **29** (2), 610-611.

[4] Cheng, R.C.H. and Feast, G.M. (1979), "Some Simple Gamma Variate Generators," *Appli. Statist.* **28**(3), 290-295.

[5] Danell, B. (1982), "Studies on Lactation Yield and Individual Test Day Yields of Swedish Red Cows," *Acta Agriculryrae Scandinavica.* **32**, 65-81.

[6] Davison, A.C., Hinkley, D.V. and Schechtman, E. (1986), "Efficient Bootstrap Simulation," *Biometrika* **73**(3), 555-566.

[7] Efron, B. (1982). "The Jackknife, the Bootstrap, and other Resampling Plans," *Conf. Series in Appl. Math.*, No. 38. Philadelphia: SIAM.

[8] Efron, B. (1983), "Estimating the error rate of a prediction rule: improvement on cross-validation," *J. Am. Statist. Assoc*. **78**, 316-331.

[9] Efron, B. and Tibshirani, R.J. (1993), *An Introduction to Bootstrap.* Monographs on Statistics and Applied Probability, New York: Chapman &Hall.

[10] Fayaye, T.R. and Ayorinde, K.L. (2010), " Effects of season, generation, number of mating , parity and Doe number of teat on Doe and litter birth characteristics in domestic Rabbit, " *Research Journal of Animal and Veterinary science.* **5**, 6-9.

[11] Fernandez, G. (2007), "Model selection in PROC MIXED-A user friendly SAS® Macro Application," SAS Global Forum2007-Statistics and Data Analysis, Paper 197-2007.

[12] Fishman, S.G. (1973), *Concepts and Methods in Discrete Event Simulation*, New York: Wiley.

[13] Fishman, S.G. (1976), "Sampling from Gamma Distribution on a Computer," *Management Science/Operation Research,* 407-409.

[14] Frey, H.C and Burmaster, D.E. (1999), "Methods for Characterizing Variability and Uncertainty: Comparison of Bootstrap Simulation and Likelihood-Based Approaches," *Risk Analysis* **19**(1), 109-130.

[15] Gardner, F.M. and Baker, J.D. (1997), *Simulation Techniques: Models of Communication Signals and Processes,* New York: Wiley.

[16] Gebeyehu, G., Asmare, A. and Asseged, B. (2005), "Reproductive performances of Fogera cattle and their Friesian crosses in Andassa ranch, Northwestern Ethiopia," *Livestock Research for Rural Development. Volume 17, Article #131.* http://www.lrrd.org/lrrd17/12/gosh17131.htm [Accessed June 15, 2010].

[17] Gebeyehu, G., Belihu, K. and Berihun, A. (2007), **"**Effect of parity, season and year on reproductive performance and herd life of Friesian cows at Stella private dairy farm, Ethiopia," *Livestock Research for Rural Development. Volume 19, Article #98.* http://www.lrrd.org/lrrd19/7/gosh19098.htm [Accessed May 15, 2010].

[18] Goshu, G. (2005), **"**Breeding efficiency, lifetime lactation and calving performance of Friesian-Boran crossbred cows at Cheffa farm, Ethiopia," *Livestock Research for Rural Development. Volume 17, Article #73.* http://www.lrrd.org/lrrd17/7/gosh17073.htm [Accessed May 15, 2010]

[19] Haile, A., Joshi, B. K., Ayalew, W., Tegegne, A. and Singh, A. (2009), "Genetic evaluation of Ethiopian Boran Cattle and their crosses with Holstein Friesian in central Ethiopia: Milk production traits," *Animal* **3**(4), 486-493.

[20] Hizkias, K. (1998), "Dairy development in Ethiopia," *Proceedings of the Smallholder Dairy Development of Project,* Addis Ababa: Ministry of Agriculture, pp. 26-39.

[21] Jadhav, K.L., Tripathi, V.N., Taneja, V.K. and Kale, M.M. (1991), "Performance of various Holstein × Sahiwal grades for first lactation reproduction and production traits," *Indian Journal of Dairy Science* **44**, 209-215.

[22] Kleijinen, J. and Groenendaal, W.V. (1992), *Simulation: A Statistical Perspective,* England: Wiley.

[23] Law, A.M. and Kelton, W.D. (2000), *Simulation Modeling and Analysis,* New York: *McGraw*-Hill.

[24] Law, A.M. (2007), *Simulation Modeling and Analysis,* New York: McGraw-Hill.

[25] Lehmer, D.H. (1951), "Mathematical Methods in Large-Scale Computing Units," *Ann. Comput. Lab. Harvard univ.* **26**, 141-146.

[26] Lewis, P.A.W & Orav, E.J (1989), *Simulation methodology for statistician, operations analysts and engineers*, California: Wadsworth &Brooks.

[27] Littell, R.C, Milliken, G.A., Stroup, W.W., Wolfinger, R.D. and Schabenberger, O. (2006), *SAS For Mixed Models*, North Carolina: SAS Institute Inc.

[28] Marsaglia, G. and Bray, T.A. (1965), "A Convenient Method for Generating Normal Variables," *SIAM Rev.***6** (3), 260-264.

[29] Million Tadesse and Tadelle Dessie (2003), "Milk production performance of Zebu, Holstein Friesian and their crosses in Ethiopia," *Livestock Research for Rural Development (15) 3.* http://www.lrrd.org/lrrd15/3/Tade153.htm [Retrieved May 18, 2010].

[30] Mohamed, A.M.A., Simeon, E. and Yemesrach, A. (2004), "Dairy development in Ethiopia," Environmental and production technology division (EPTD) discussion paper No.123. International Food Policy Research Institute Washington, USA.

[31] Mood, A.M., Graybill, F.A. and Boes, D.C. (1974), *Introduction to the Theory of Statistics*, Singapore: McGraw-Hill.

[32] Mukerjee, S. (2005), "Genetic evaluation of Frieswal cattle," PhD, National Research institute, Karnal, India.

[33] Muskasa-Mugerwa, E., Bekele, E. & Tessema, T. (1989), "Type and productivity of indigenous cattle in central Ethiopia," *Trop. Anim Hith Prod*. **21**, 120.\

[34] Ofcansky, T.P. & Berry, L. (eds) (1991), *Ethiopia: A country study*, Washington: GPO for the library of congress.

[35] Patterson, H.D. and Thompson, R. (1971), "Recovery of inter-block information when block sizes are unequal," *Biometrica* **58**, 545-554.

[36] Perros, H. (2009), "Computer Simulation Techniques: The Definitive Introduction" http://www.csc.ncsu.edu/faculty/perros//simulation.pdf [Accessed 05 May 2010].

[37] Raheja, K.L.(1994), "Genetic parameters for first lactation and lifetime production traits in Friesian–Hariana and Friesian–Sahiwal halfbreds estimated by multiple trait maximum likelihood procedures. *The Indian Journal of Animal Science* **64**, 616–621.

[38] Raspe, R.E. (2007), *The Surprising Adventures of Baron Munchausen,* USA: Wilder Publication.

[39] Ray,D.E., Halbach, T.J. and Armstrong, D.V. (1992), "Season and lactation number effects on milk production and reproduction of dairy cattle in Arizona, " *Journal of Animal Science* **75**(11), 2976-2983.

[40] Rosbenbluth, A. and Wiener, N. (1945), "The Role of Models in Science", *philos. Sci.* **12**(4), 316-321.

[41] Rubinstein, R.Y. (1981), *Simulation and the Monte Carlo method,* New Jersey: Wiley.

[42] Saxena, M.M., Katpatal, B.G. and Pandey, H.S. (1997), "Study of Milk Constituents and Their Yield in Holstein–Friesian Cows," *Indian Journal of Animal Production and Management* **13**(3), 127–130.

[43] Schabenberger, O. (2004), "Mixed Model Influence Diagnostics," SUGI29-Statistics and Data Analysis, Paper 189-29.

[44] Searle, S. R. (1997), *Linear Models,* (2nd Ed.) New York: Wiley.

[45] Searle, S. R. (2006a), *Linear Models for Unbalanced Data,* New Jersey: Wiley.

[46] Searle, S.R., Casella, G., & McCulloch, C.E. (2006b), *Variance components,* New Jersey: Wiley.

[47] Sobol, I.M. (1975), *The Monte Carlo Method,* Moscow: Mir Publisher

[48] West, B.T., Welch, K.B., and Galecki, A.T. (2007), *Linear Mixed Models: A practical Guide Using Statistical Software,* USA: Chapman &Hall/CRC.

[49] Wikipedia (2009), "Debre Zeyit", http://en.wikipedia.org/wiki/Debre_Zeyit [Accessed 20 December 2009].

[50] Wilmink, J.B.M. (1987), "Adjusted of test day milk, fat and protein yield for age, season and stage of lactation", Livestock Production Science 16, 321-334.

[51] Wood, P.D.P. (1969), "Factors affecting the shape of the lactation curve in cattle", *Animal Production.* **11**, 307-316.

Appendix A

C++ Code for Monte Carlo Simulation

```
/**********************************Program-1*********************************
*  Author:           YAbebal Ayalew                                          *
*  Starting Date:    March 16, 2010                                          *
*  End Date:         April 26+2, 2010                                        *
*  Title:            Monte Carlo Simulation                                  *
*****************************************************************************/
#include <iostream>
#include <fstream>
#include <conio.h>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <ciso646>
using namespace std;
#define  PI 3.141592654
#define  e 2.718281828
//Defining constants
const int  ALL_TOTAL=4000;
ofstream fout("Monte Carlo Simulation.xls"); //Opening file
ifstream fin("Monte Carlo Simulation.xls");
//Global variables declaration
int CowId[ALL_TOTAL];
int v; // used to increase the second componenet of test day milk yield in daily milk
yield function
int z; // used in leanear search
int counter=0;
```

```
int i=0; // Used to cheak stanNorm function called even or odd time

int d; // Holds a random number between 7 and 10

int calvingYear[ALL_TOTAL]; // Holds the year between 1990 up to 2004

int herd[ALL_TOTAL]; // hold the farm choice

int parity[ALL_TOTAL]; // Holds dry period of cow of id counter

int geneticType[ALL_TOTAL]; // Hold the genetic group

double even; // Used in generation of standard normal variate for even number call

double sum; // used as a container in the daily milk yield

double TestDayMilk[5][10]; // hold test day milk on a defined time interval to
calculate 305 day milk yield

// Function declaration

void Print(double); //Prints the output

void Cleaner(); // Cleanes the value of Test day Milk array

double DailyMilkYield(int, int, int, int, int, int); // Determine the average daily
milk yield

double LactationMilkYield(); // Estimate lactation milk yield

double ThreeLactation(); // estimate 305 day milk yield

double UniformRandom(); //Generates Uniform Random number between 0 and 1

double StanNormal(); //generates normal variate from N(0,1)

double Normal(double, double); //generates normal variate from N(mu, Sigma)

double JohnsonSB(double, double, double, double); //Generates Johnson random variate

double LogLogistic(double, double, double); //Generates Log Logistic with 3
parameters

double Gamma(double, double); //Generates gamma function

double PearsonType6(double, double, double, double); //Generates Pearson type VI
random variate

double Dagum(double, double, double, double); // Generates Dagum with three and four
variate

double Cauchy(double, double); // Generates Cauchy distribution

double GambelMin(double, double); // Genetrate Gambel Min

double Laplace(double, double); // laplace distribution
```

```
double HyperSecant(double, double);
double GenPareto(double, double, double);
double GenExtremeValue(double, double,double);
double GammaSpecial(double, double);
double LactationMilkYield(double, int);
int LactationLength(int, int, int, int, int); // generates lactation length
int ParityNumber(); // Determine the parity number of a cow with id number
int CalvingSeason(); // Season of calving
int CowNumber(); // Generate random cow identification number
int CalvingYear(); //Generate year of calving
int GeneticGroup(); // Generates genetic group
int Herd();// Generates farm
int Bernoulli(float); // generates Bernoulli random variate
int LenearSearch(); // Searches the existance of cow id
int LenearSearch2();
int randomNumberGenerator(int,int); // Generates random number between max and min
//Function construction
void Cleaner(){
       for(int h=0; h<5; h++){
              for(int f=0; f<10;f++)
                     TestDayMilk[h][f]=0;
       }
}
void Print(double yabebal){
              fout<<yabebal<<"\t";
}
int CalvingYear(){
        int m;
        switch(counter){
```

```
            case 0:
                  m=randomNumberGenerator(1990, 1995);
                  return calvingYear[counter]=m;
                  break;
            default:
                  if(LenearSearch()==-1){
                        m=randomNumberGenerator(1990, 1995);
                        return calvingYear[counter]=m;
                  }
                  else{
                       if(calvingYear[z]+1==2004)
                  m= 2004;
                 else if (calvingYear[z]+10<2004)
                       m=randomNumberGenerator(calvingYear[z]+1, 1997);
                else
                       m=randomNumberGenerator(calvingYear[z]+1, 2004);
                  return calvingYear[counter]=m;
        }
                  break;
        }
}
int CalvingSeason(){
      return randomNumberGenerator(1, 3);
}
int GeneticGroup(){
    int Ggroup;
      switch(counter){
            case 0:
           Ggroup=randomNumberGenerator(1, 5);
```

```
                    geneticType[counter]=Ggroup;
                    return geneticType[counter];
                    break;
             default:
                    if(LenearSearch()!=-1)
                           return geneticType[counter]=geneticType[z];
                    else{
                        Ggroup=randomNumberGenerator(1, 5);
                        geneticType[counter]=Ggroup;
                           return geneticType[counter];
            }
                           break;
    }
}
int ParityNumber(){
       switch(counter){
             case 0:
                    parity[counter]=1;
                    return parity[counter];
                    break;
             default:
                    if(LenearSearch()==-1)
                  return parity[counter]=1;
                    else
                           return parity[counter]=parity[z]+1;
                           break;
    }
}
int Herd(){
```

```
       switch(counter){
              case 0:
                     herd[counter]=Bernoulli(0.5)+1;
                     return herd[counter];
                     break;
              default:
                     if(LenearSearch()!=-1)
                            return herd[counter]=herd[z];
                     else
                            return herd[counter]=Bernoulli(0.5)+1;
                            break;
    }
}
int LenearSearch(){
              for(z=counter-1; z>=0;z--){
                     if(CowId[z]==CowId[counter])
                            return z;
    }
    return -1;
}
int LenearSearch2(){
              for(z=counter-1; z>=0;z--){
                     if(calvingYear[z]==2004 && CowId[counter]==CowId[z])
                            return z;
       }
    return -1;
}
int CowNumber(){
       switch(counter){
```

```
		case 0:
			CowId[counter]=randomNumberGenerator(1, 888);
			return CowId[counter];
			break;
		default:
			CowId[counter]=randomNumberGenerator(1, 888);
			if( LenearSearch2()!=-1){
				CowNumber();
		}
			return CowId[counter];
			break;
	}
}
int Bernoulli(float p){
	double u;
	u=UniformRandom();
	if(u<=p)
		return 1;
	else
		return 0;
}
int LactationLength(int cSeason, int gGroup, int cYear, int parity, int farm){
	double num1, num2, num3,num4, num5;
	switch(cSeason){
		case 1:
			num1=randomNumberGenerator(3, 797);
			break;
		case 2:
			num1=Cauchy(311.42, 50.509 );
```

```
                break;
        case 3:
                num1=Laplace(0.01053, 298.99);
                break;
}
switch(gGroup){
        case 1:
                num2= randomNumberGenerator(1, 215);
                break;
        case 2:
                num2=LogLogistic(22.401, 1324.2, -994.2);
                break;
        case 3:
                num2=LogLogistic(44.751, 2586.2 , -2248.9);
                break;
        case 4:
                num2=Dagum(0.36771, 11.586, 440.15, -37.47);
                break;
        case 5:
                num2=Normal(210.36, 118.13 );
                break;
}
switch(cYear){
        case 1990:
        case 1991:
        case 1992:
                num3=Normal(315.24, 140.59 );
                break;
        case 1993:
```

```
        case 1994:
        case 1995:
                num3=Laplace(0.01075, 323.24);
                break;
        case 1996:
        case 1997:
        case 1998:
                num3= Laplace(0.00969, 295.67);
                break;
        case 1999:
        case 2000:
        case 2001:
                num3=Laplace(0.0126, 297.85);
                break;
        case 2002:
        case 2003:
        case 2004:
                num3=randomNumberGenerator(1,587);
                break;
}
switch(parity){
        case 1:
                num4= randomNumberGenerator(1,797);
                break;
        case 2:
                num4=Cauchy(316.73, 53.682);
                break;
        case 3:
                num4=Laplace(0.0119, 299.35);
```

```
                        break;
                case 4:
                        num4=Cauchy(298.74, 49.156);
                        break;
                case 5:
                        num4=Cauchy(292.23, 51.085);
                        break;
                case 6:
                        num4=Laplace(0.01088, 286.26);
                        break;
                case 7:
                case 8:
                case 9:
                case 10:
                        num4=HyperSecant(271.48, 131.87);
                        break;
        }
        switch(farm){
                case 1:
                        num5=HyperSecant(358.23, 83.1);
                        break;
                case 2:
                        num5=randomNumberGenerator(1, 797);
                        break;
        }
        return (int)ceil((num1+num2+num3+num4+num5)/5);
}
int randomNumberGenerator(int min,int max){
        return (rand() % (max - min + 1)) + min;
```

```
}
double Gamma(double alpha, double beta){
        double a, b, c,d, U1, U2, W;
        a=alpha-1;
        b=(alpha-pow((6*alpha), -1.0))/a;
        c=2/a;
        d=c+2;
        do{
           U1= UniformRandom();
           U2= UniformRandom();
           W=b*U1/U2;
            if((c*U2-d+W+pow(W,-1.0))<=0)
            break;
        }while(c*log(U2)-log(W)+W-1>=0);
        return (float)a*W*beta;
}
double GammaSpecial(double alpha, double beta){
        double b, U1, U2, P, Y;
        b=(e+alpha)/e;
        RE:
        U1=UniformRandom();
        P=b*U1;
        if(P>1){
           Y=-log((b-P)/alpha);
           U2=UniformRandom();
           if(U2<=pow(Y, alpha-1))
              return (float)fabs(beta*Y);
           else
               goto RE;
```

```
            }
        else{
        Y=pow(P, 1/alpha);
        U2=UniformRandom();
        if(U2<=pow(e, -Y))
          return (float)fabs(beta*Y);
        else
        goto RE;
           }
}
double PearsonType6(double alpha1, double alpha2, double beta, double gamma){
        double Y1, Y2;
        if (alpha1>=1)
           Y1=Gamma(alpha1, beta);
        else
            Y1=GammaSpecial(alpha1, beta);
        if (alpha2>=1)
             Y2=Gamma(alpha2, 1);
        else
            Y2=GammaSpecial(alpha2, 1);
           return (float)(Y1/Y2)+gamma;
}
double Dagum(double k, double alpha, double beta, double gamma){
          double U=UniformRandom();
          return (float)beta*pow((pow(1/U, 1/k)+1), -1/alpha)+gamma;
}
double Cauchy(double mu, double sigma){
          double U = UniformRandom();
        double S = pow(PI,2.0);
```

```
          double G = tan (S*(U-0.5)/180);
          return (float)((sigma*G)+mu);
}
double GambelMin(double mu, double sigma){
          double U = UniformRandom();
          return (float)sigma*log(log(1/(1-U)))+mu;
}
double Laplace(double lamda, double mu){
          double U= UniformRandom();
          double D= (-log(2*U)/lamda)+mu;
          if (D<=mu)
                return (float)D;
          else
                 return (float)-log(2*(1-U))/lamda+mu;
}
double HyperSecant(double mu, double sigma){
       double U=UniformRandom();
          double Nj=(PI*U)/2;
          double G= tan((PI*Nj)/180);
          double Q=(2*sigma*G/PI)+mu;
       return (float)Q;
    }
double GenPareto(double k, double sigma, double mu){
          double U= UniformRandom();
          if(k==0)
              return (float)fabs(sigma*log(1-U)+mu);
          else
              return (float)fabs((sigma*(1-pow(1/(1-U), k))/k)+mu);
}
```

```
double GenExtremeValue(double k, double sigma,double mu){
          double U=UniformRandom();
          if (k==0)
                return (float)fabs(sigma*log(-1/log(U))+mu);
          else
                 return (float)fabs((sigma/k)*(pow(-1/log(U), k)-1)+mu);
}
double UniformRandom(){
          return rand()/(double) RAND_MAX;
}
double Normal(double mean, double stanDeviation){
          return (float)mean+stanDeviation*StanNormal();
}
double JohnsonSB(double alpha1, double alpha2, double a, double b){
          double Z=StanNormal();
          double Y=exp((Z-alpha1)/alpha2);
          return (float)fabs((a+b*Y)/(Y+1));
}
double LogLogistic(double alpha, double beta, double gamma){
          double U=UniformRandom();
          return (float)beta*pow((U/(1-U)),1/alpha)+gamma;
}
double StanNormal(){
          i++;
          double W,W2, W1, U1,U2;
          if (i%2!=0){
          do{
                   U1=UniformRandom();
                   U2=UniformRandom();
```

```
                    W1=2*U1-1;
                    W2=2*U2-1;
                    W=pow(W1, 2)+pow(W2,2);
                 }while(W>1);
           double C=sqrt(-2*log(W)/W);
           even=C*W2;
           return C*W1;
              }
         else
               return even;
}
double DailyMilkYield(int cSeason, int gGroup, int cYear, int parity, int farm, int
lLength){
          double average;
          double average1;
          double average2;
          double average3;
          double average4;
          double norm;
          d= (rand() % (10 - 7 + 1)) + 7;
    switch(gGroup){
              case 1:
                    sum=0.0;
                v=0;
                   for(int j=0;j<lLength; j++){
                          norm=(float)fabs(PearsonType6(28.539, 2285.3, 1058.4, -
7.1189));
                          sum=sum+norm;
                          if(j%(lLength/d)==0){
```

```
                        TestDayMilk[0][v]=norm;
                        ++v;
                }
        }
        if(d<10){
                for(int t=d+1;t<10;t++){
                        TestDayMilk[0][v]=0;
                }
        }
average=sum/lLength;
                        break;
 case 2:
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
                norm=(float)Dagum(0.26446, 7.8268, 6.1838, 1.0983);
                sum=sum+norm;
                if(j%(lLength/d)==0){
                        TestDayMilk[0][v]=norm;
                        ++v;
                }
        }
        if(d<10){
                for(int t=d+1;t<10;t++){
                        TestDayMilk[0][v]=0;
                }
        }
        average=sum/lLength;
                        break;
```

```
case 3:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float)JohnsonSB(1.4386,  1.7067, 20.559, -0.15636);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                     TestDayMilk[0][v]=norm;
                     ++v;
              }
       }
       if(d<10){
              for(int t=d+1;t<10;t++){
                     TestDayMilk[0][v]=0;
              }
       }
    average=sum/lLength;
              break;
case 4:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float)Dagum(0.45116, 6.6657, 8.0292, 0);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                     TestDayMilk[0][v]=norm;
                     ++v;
              }
       }
```

```
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[0][v]=0;
                        }
                }
                average=sum/lLength;
                        break;
        case 5:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float)fabs(JohnsonSB(2.7365,  2.0056, 13.784, -
1.2566));
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[0][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[0][v]=0;
                        }
                }
                average=sum/lLength;
                        break;
        }
        switch(cSeason){
                case 1:
```

```
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float)Dagum(0.13027, 8.3812, 8.771, -0.01187);
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[1][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[1][v]=0;
                        }
                }
                average1=sum/lLength;


                                break;
        case 2:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float) fabs(JohnsonSB(1.3141,  2.0892, 28.173, -
4.9311));
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[1][v]=norm;
                                ++v;
                        }
```

```
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[1][v]=0;
                        }
                }
                average1=sum/lLength;
                                break;
        case 3:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=randomNumberGenerator(0,18);
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[1][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[1][v]=0;
                        }
                }
                average1=sum/lLength;
                                break;
}
switch(cYear){
        case 1990:
```

```
case 1991:
case 1992:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float)Dagum(0.15126, 9.5144, 6.0532, 0);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                     TestDayMilk[2][v]=norm;
                     ++v;
              }
       }
       if(d<10){
              for(int t=d+1;t<10;t++){
                     TestDayMilk[2][v]=0;
              }
       }
       average2=sum/lLength;
                     break;
case 1993:
case 1994:
case 1995:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float)Normal(4.8649, 2.2371);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                     TestDayMilk[2][v]=norm;
```

```
                ++v;
            }
        }
        if(d<10){
            for(int t=d+1;t<10;t++){
                TestDayMilk[2][v]=0;
            }
        }
        average2=sum/lLength;
                break;
case 1996:
case 1997:
case 1998:
        sum=0;
        v=0;
        for(int j=0;j<lLength; j++){
            norm=(float) fabs(GenPareto(-1.0082, 9.5899, 0.14904));
            sum=sum+norm;
            if(j%(lLength/d)==0){
                TestDayMilk[2][v]=norm;
                ++v;
            }
        }
        if(d<10){
            for(int t=d+1;t<10;t++){
                TestDayMilk[2][v]=0;
            }
        }
        average2=sum/lLength;
```

```
                    break;
case 1999:
case 2000:
case 2001:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float) fabs(GenPareto(-0.88185, 8.6544, 0.32174));
              sum=sum+norm;
              if(j%(lLength/d)==0){
                     TestDayMilk[2][v]=norm;
                     ++v;
              }
       }
       if(d<10){
              for(int t=d+1;t<10;t++){
                     TestDayMilk[2][v]=0;
              }
       }
       average2=sum/lLength;
                     break;


case 2002:
case 2003:
case 2004:
       sum=0.0;
       v=0;
       for(int j=0;j<lLength; j++){
              norm=(float) fabs(GenPareto(-0.68479, 9.7055, 0.53803));
```

```
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[2][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[2][v]=0;
                        }
                }
                average2=sum/lLength;
                                break;
}
switch(farm){
        case 1:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float)Dagum(0.20095, 7.7595, 8.6542, 0.07957);
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[3][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[3][v]=0;
```

```
                        }
                }
                average3=sum/lLength;
                                break;
        case 2:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float)Dagum(0.1103, 7.5545, 8.3896, 0);
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[3][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[3][v]=0;
                        }
                }
                average3=sum/lLength;
                                break;
}
switch(parity){
        case 1:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float)Dagum(0.09566,9.7951, 8.5568, 0);
```

```
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[4][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[4][v]=0;
                        }
                }
                average4=sum/lLength;
                                break;
        case 2:
                sum=0.0;
                v=0;
                for(int j=0;j<lLength; j++){
                        norm=(float) fabs(GenExtremeValue(-0.16671, 2.6149,
3.782));
                        sum=sum+norm;
                        if(j%(lLength/d)==0){
                                TestDayMilk[4][v]=norm;
                                ++v;
                        }
                }
                if(d<10){
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[4][v]=0;
                        }
```

```
        }
        average4=sum/lLength;
                    break;
case 3:
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
              norm=(float)Dagum(0.16814, 6.7382, 8.9983, -0.00384);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                    TestDayMilk[4][v]=norm;
                    ++v;
              }
        }
        if(d<10){
              for(int t=d+1;t<10;t++){
                    TestDayMilk[4][v]=0;
              }
        }
        average4=sum/lLength;
                    break;
case 4:
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
              norm=(float)Dagum(0.11991,  8.4256, 9.425, 0);
              sum=sum+norm;
              if(j%(lLength/d)==0){
                    TestDayMilk[4][v]=norm;
```

```
                ++v;
            }
        }
        if(d<10){
            for(int t=d+1;t<10;t++){
                TestDayMilk[4][v]=0;
            }
        }
        average4=sum/lLength;
                break;
    case 5:
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
            norm=(float)GenPareto(-0.79619, 8.6659, 0.07912);
            sum=sum+norm;
            if(j%(lLength/d)==0){
                TestDayMilk[4][v]=norm;
                ++v;
            }
        }
        if(d<10){
            for(int t=d+1;t<10;t++){
                TestDayMilk[4][v]=0;
            }
        }
        average4=sum/lLength;
                break;
    case 6:
```

```
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
                norm=GenExtremeValue(-0.26522, 2.845, 3.9457);
                sum=sum+norm;
                if(j%(lLength/d)==0){
                        TestDayMilk[4][v]=norm;
                        ++v;
                }
        }
        if(d<10){
                for(int t=d+1;t<10;t++){
                        TestDayMilk[4][v]=0;
                }
        }
        average4=sum/lLength;
                        break;
default:
        sum=0.0;
        v=0;
        for(int j=0;j<lLength; j++){
                norm=Dagum(0.15372, 8.2967, 8.8713, -0.01103);
                sum=sum+norm;
                if(j%(lLength/d)==0){
                        TestDayMilk[4][v]=norm;
                        ++v;
                }
        }
        if(d<10){
```

```
                        for(int t=d+1;t<10;t++){
                                TestDayMilk[4][v]=0;
                        }
                }
                average4=sum/lLength;
                        break;
        }
        return (average+average1+average2+average3+average4)/5;
}
double LactationMilkYield(double milk, int lLength){
        return (float)milk*lLength;
        }
double ThreeLactation(){
        int num;
        double sum;
        double summation=0;
        double container[10];
        for(int se=0;se<=10;se++){
                sum=0;
                num=5;
                for(int be=0;be<4;be++){
                        if(TestDayMilk[be][se]==0) // Cheaking empty array;
                                num=num-1;
                        sum =sum+TestDayMilk[be][se]; // sum test day milk yield colunm
wise
                }
                container[se]=sum/num;
        }
        for(int k=0; k<10;k++)
```

```
            summation=summation+container[k]; //sum the single matrix
       return 30.5*summation;
}

int main(){
    srand(time(0));
    int COB, HerD, Q, RT, M, PO, times;
    double PUT;
    cout << "\n\n\n\n\n\t\t\tHow many iteration do you need?\n\n\t\t\t\t ";
    cin>>times;
    if (cin.good() == false) {
    cout << "\n\n\n\n\n\t\t\tThat was not an integer." << endl;
    cout<<"\n\n\n\t\t\tPress any key to EXIT";
    getch();
    return 0;
  }
    cout << "\n\n\t\t\tIt is working Please wait ...";
    if(!fout) {
    cout << "Cannot open output file.\n";
    return 1;
      }
    fout<<"CowId\tParity\tFarm\tGeneticG.\tYear\tSeason";
    fout<<"\tLLength\tDMY\t305MY\tLMilk"<<endl;
    for(counter=0; counter<times;counter++){
            Print(CowNumber());
            COB=ParityNumber();
            Print(COB);
            HerD=Herd();
            Print(HerD);
```

```
            Q=GeneticGroup();
            Print(Q);
            RT=CalvingYear();
            Print(RT);
            M=CalvingSeason();
            Print(M);
            PO=LactationLength(M, Q, RT, COB, HerD);
            Print(PO);
            PUT= DailyMilkYield(M, Q, RT, COB, HerD, PO);
            Print(PUT);
            Print(ThreeLactation());
            Print(LactationMilkYield(PUT, PO));
            Cleaner();
            fout<<endl;
    }
     fout.close();
     fin.close();
       return 0;
}
```

Appendix B

C++ Code for Bootstrap Simulation

```
/**********************************Program-2*********************************
*  Author:           YAbebal Ayalew                                          *
*  Starting Date:   May 02, 2010                                            *
*  End Date:        May 04+1, 2010                                          *
*  Title:            Bootstrap Simulation                                    *
*****************************************************************************/
#include <iostream>
#include <fstream>
#include <conio.h>
#include <cmath>
#include <ctime>
#include <cstdlib>
#include <algorithm>
#include <ciso646>
#include "BootstrapVariable.h"
using namespace std;
ofstream fout("Bootstrap Simulation.xls");
//ifstream fin("Bootstrap Simulation.xls");
// Declaring constatnts
//const int TOTAL=204;
const int ALL_TOTAL=8744;
// Global variable declaration
long int z, counter = 0;
long int calvingYear[ALL_TOTAL], CowId[ALL_TOTAL];
int geneticType[ALL_TOTAL], times, *L=NULL;
int herd[ALL_TOTAL];
```

```
int parity[ALL_TOTAL];


//Function declaration
double DailyMilkYield(int, int, int, int, int);
double LactationMilkYield(int, double);
double ThreeMilkYield(double);
int LactationLength(int, int, int, int, int);
int CalvingYear();
int CalvingSeason();
int GeneticGroup();
int Herd();
int CowNumber();
int randomNumberGenerator(int, int);
int LenearSearch();
int LenearSearch2();
void Printer(double);
void RandomPermutation(int);


void Print(double yabebal){
              fout<<yabebal<<"\t";
}
void RandomPermutation(int size){
       long int K, j;
       K=(size*times);
       L=new int [K];
       for(int m=O; m <K; m++)
              L[m]=(m%size)+1;
       for(int i= K; i<=2; i--){
              j =randomNumberGenerator(1, i);
```

```
            int temp=L[i];
            L[j]=L[i];
            L[i]=L[j];
      }
}
int randomNumberGenerator(int min,int max){
      return (rand() % (max - min + 1)) + min;
}
int ParityNumber(){
      switch(counter){
            case 0:
                  parity[counter]=1;
                  return parity[counter];
                  break;
            default:
                  if(LenearSearch()==-1)
               return parity[counter]=1;
                  else
                        return parity[counter]=parity[z]+1;
                        break;
    }
}
int Herd(){
      switch(counter){
            case 0:
                  herd[counter]=randomNumberGenerator(1,2);
                  return herd[counter];
                  break;
            default:
```

```
                    if(LenearSearch()!=-1)
                           return herd[counter]=herd[z];
                    else
                           return herd[counter]=randomNumberGenerator(1,2);
                           break;
    }
}
int CowNumber(){
       switch(counter){
              case 0:
                    CowId[counter]=randomNumberGenerator(1,888);
                    return CowId[counter];
                    break;
              default:
                    CowId[counter]=randomNumberGenerator(1,888);
                    if( LenearSearch2()!=-1){
                             CowNumber();
             }
                    return CowId[counter];
                    break;
       }
}
int CalvingYear(){
         int m;
        switch(counter){
              case 0:
                    m=randomNumberGenerator(1990, 1995);
                    return calvingYear[counter]=m;
                    break;
```

```
            default:
                  if(LenearSearch()==-1){
                        m=randomNumberGenerator(1990, 1995);
                        return calvingYear[counter]=m;
                  }
                  else{
                       if(calvingYear[z]+1==2004)
                  m= 2004;
                else if (calvingYear[z]+10<2004)
                      m=randomNumberGenerator(calvingYear[z]+1, 1997);
               else
                      m=randomNumberGenerator(calvingYear[z]+1, 2004);
                  return calvingYear[counter]=m;
        }
                  break;
        }
}
int CalvingSeason(){
       return (rand()%3)+1;
}
int LenearSearch(){
    for(z=counter-1; z>=0;z--){
        if(CowId[z]==CowId[counter])
            return z;
    }
    return -1;
}
int LenearSearch2(){
             for(z=counter-1; z>=0;z--){
```

```
                    if(calvingYear[z]==2004 && CowId[counter]==CowId[z])
                           return z;
       }
    return -1;
}
int GeneticGroup(){
       int Ggroup;
       switch(counter){
              case 0:
             Ggroup=randomNumberGenerator(1, 5);
                    geneticType[counter]=Ggroup;
                    return geneticType[counter];
                    break;
              default:
                    if(LenearSearch()!=-1)
                           return geneticType[counter]=geneticType[z];
                    else{
                        Ggroup=randomNumberGenerator(1, 5);
                        geneticType[counter]=Ggroup;
                           return geneticType[counter];
             }
                           break;
    }
}
double LactationMilkYield(int LL, double DYM){
       return LL*DYM;
}
double DailyMilkYield(int cSeason, int cYear, int farm, int parity, int gGroup){
          double av1, av2, av3, av4, av5;
```

```
   int size,j;
switch(cSeason){
      case 1:
            size=sizeof(DYDSeason1)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av1=DYDSeason1[L[j]];
            delete [] L;
            L=NULL;
            break;
      case 2:
            size=sizeof(DYDSeason2)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av1=DYDSeason1[L[j]];
            delete [] L;
            L=NULL;
            break;
      case 3:
            size=sizeof(DYDSeason3)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av1=DYDSeason1[L[j]];
            delete [] L;
            L=NULL;
            break;
}
   switch(cYear){
      case 1990:
```

```
case 1991:
case 1992:
       size=sizeof(DYDYear1)/8;
       RandomPermutation(size);
       j=randomNumberGenerator(0, size*times);
       av2=DYDYear1[L[j]];
       delete [] L;
       L=NULL;
       break;
case 1993:
case 1994:
case 1995:
       size=sizeof(DYDYear2)/8;
       RandomPermutation(size);
       j=randomNumberGenerator(0, size*times);
       av2=DYDYear2[L[j]];
       delete [] L;
       L=NULL;
       break;
case 1996:
case 1997:
case 1998:
       size=sizeof(DYDYear3)/8;
       RandomPermutation(size);
       j=randomNumberGenerator(0, size*times);
       av2=DYDYear3[L[j]];
       delete [] L;
       L=NULL;
       break;
```

```
    case 1999:
    case 2000:
    case 2001:
           size=sizeof(DYDYear4)/8;
           RandomPermutation(size);
           j=randomNumberGenerator(0, size*times);
           av2=DYDYear4[L[j]];
           delete [] L;
           L=NULL;
           break;
    case 2002:
    case 2003:
    case 2004:
           size=sizeof(DYDYear5)/8;
           RandomPermutation(size);
           j=randomNumberGenerator(0, size*times);
           av2=DYDYear5[L[j]];
           delete [] L;
           L=NULL;
           break;
}
switch(farm){
    case 1:
           size=sizeof(DYDFarm1)/8;
           RandomPermutation(size);
           j=randomNumberGenerator(0, size*times);
           av3=DYDFarm1[L[j]];
           delete [] L;
           L=NULL;
```

```
            break;
        case 2:
            size=sizeof(DYDFarm2)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av3=DYDFarm2[L[j]];
            delete [] L;
            L=NULL;
            break;
    }
    switch(parity){
        case 1:
            size=sizeof(DYDParity1)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=DYDParity1[L[j]];
            delete [] L;
            L=NULL;
            break;
        case 2:
            size=sizeof(DYDParity2)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=DYDParity2[L[j]];
            delete [] L;
            L=NULL;
            break;
        case 3:
            size=sizeof(DYDParity3)/8;
```

```
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=DYDParity3[L[j]];
        delete [] L;
        L=NULL;
        break;
case 4:
        size=sizeof(DYDParity4)/8;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=DYDParity4[L[j]];
        delete [] L;
        L=NULL;
        break;
case 5:
        size=sizeof(DYDParity5)/8;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=DYDParity5[L[j]];
        delete [] L;
        L=NULL;
        break;
case 6:
        size=sizeof(DYDParity6)/8;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=DYDParity6[L[j]];
        delete [] L;
        L=NULL;
```

```
            break;
    case 7:
    case 8:
    case 9:
    case 10:
    case 11:
            size=sizeof(DYDParity7)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=DYDParity7[L[j]];
            delete [] L;
            L=NULL;
            break;
}
switch(gGroup){
    case 1:
            size=sizeof(DYDgGroup1)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av5=DYDgGroup1[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 2:
            size=sizeof(DYDgGroup2)/8;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av5=DYDgGroup2[L[j]];
            delete [] L;
```

```
                L=NULL;
                break;
            case 3:
                size=sizeof(DYDgGroup3)/8;
                RandomPermutation(size);
                j=randomNumberGenerator(0, size*times);
                av5=DYDgGroup3[L[j]];
                delete [] L;
                L=NULL;
                break;
            case 4:
                size=sizeof(DYDgGroup4)/8;
                RandomPermutation(size);
                j=randomNumberGenerator(0, size*times);
                av5=DYDgGroup4[L[j]];
                delete [] L;
                L=NULL;
                break;
            case 5:
                size=sizeof (DYDgGroup5)/8;
                RandomPermutation(size);
                j=randomNumberGenerator(0, size*times);
                av5=DYDgGroup5[L[j]];
                delete [] L;
                L=NULL;
                break;
        }
    return (float)((av1+av2+av3+av4+av5)/5);
}
```

```
double ThreeMilkYield(double DYD){
       return 30.5*(10*DYD);
}
int LactationLength(int cSeason, int cYear, int farm, int parity, int gGroup){
          int av1, av2, av3, av4, av5;
          int size, j;
          switch(cSeason){
              case 1:
                     size=sizeof(LLSeason1)/4;
                     RandomPermutation(size);
                     j=randomNumberGenerator(0, size*times);
                     av1=LLSeason1[L[j]];
                     delete [] L;
                     L=NULL;
                     break;
              case 2:
                     size=sizeof(LLSeason2)/4;
                     RandomPermutation(size);
                     j=randomNumberGenerator(0, size*times);
                     av1=LLSeason2[L[j]];
                     delete [] L;
                     L=NULL;
                     break;
              case 3:
                     size=sizeof(LLSeason3)/4;
                     RandomPermutation(size);
                     j=randomNumberGenerator(0, size*times);
                     av1=LLSeason3[L[j]];
                     delete [] L;
```

```
            L=NULL;
            break;
}
   switch(cYear){
       case 1990:
       case 1991:
       case 1992:
              size=sizeof(LLYear1)/4;
              RandomPermutation(size);
              j=randomNumberGenerator(0, size*times);
              av2=LLYear1[L[j]];
              delete [] L;
              L=NULL;
              break;
       case 1993:
       case 1994:
       case 1995:
              size=sizeof(LLYear2)/4;
              RandomPermutation(size);
              j=randomNumberGenerator(0, size*times);
              av2=LLYear2[L[j]];
              delete [] L;
              L=NULL;
              break;
       case 1996:
       case 1997:
       case 1998:
              size=sizeof(LLYear3)/4;
              RandomPermutation(size);
```

```
            av2=LLYear3[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 1999:
    case 2000:
    case 2001:
            size=sizeof(LLYear4)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av2=LLYear4[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 2002:
    case 2003:
    case 2004:
            size=sizeof(LLYear5)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av2=LLYear5[L[j]];
            delete [] L;
            L=NULL;
            break;
}
switch(farm){
    case 1:
            size=sizeof(LLFarm1)/4;
            RandomPermutation(size);
```

```
            j=randomNumberGenerator(0, size*times);
            av3=LLFarm1[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 2:
            size=sizeof(LLFarm2)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av3=LLFarm2[L[j]];
            delete [] L;
            L=NULL;
            break;
}
switch(parity){
    case 1:
            size=sizeof(LLParity1)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=LLParity1[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 2:
            size=sizeof(LLParity2)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=LLParity2[L[j]];
            delete [] L;
```

```
        L=NULL;
        break;
case 3:
        size=sizeof(LLParity3)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=LLParity3[L[j]];
        delete [] L;
        L=NULL;
        break;
case 4:
        size=sizeof(LLParity4)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=LLParity4[L[j]];
        delete [] L;
        L=NULL;
        break;
case 5:
        size=sizeof(LLParity5)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av4=LLParity5[L[j]];
        delete [] L;
        L=NULL;
        break;
case 6:
        size=sizeof(LLParity6)/4;
        RandomPermutation(size);
```

```
            j=randomNumberGenerator(0, size*times);
            av4=LLParity6[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 7:
    case 8:
    case 9:
    case 10:
    case 11:
            size=sizeof(LLParity7)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av4=LLParity7[L[j]];
            delete [] L;
            L=NULL;
            break;
}
switch(gGroup){
    case 1:
            size=sizeof(LLgGroup1)/4;
            RandomPermutation(size);
            j=randomNumberGenerator(0, size*times);
            av5=LLgGroup1[L[j]];
            delete [] L;
            L=NULL;
            break;
    case 2:
            size=sizeof(LLgGroup2)/4;
```

```
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av5=LLgGroup2[L[j]];
        delete [] L;
        L=NULL;
        break;
case 3:
        size=sizeof(LLgGroup3)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av5=LLgGroup3[L[j]];
        delete [] L;
        L=NULL;
        break;
case 4:
        size=sizeof(LLgGroup4)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av5=LLgGroup4[L[j]];
        delete [] L;
        L=NULL;
        break;
case 5:
        size=sizeof(LLgGroup5)/4;
        RandomPermutation(size);
        j=randomNumberGenerator(0, size*times);
        av5=LLgGroup5[L[j]];
        delete [] L;
        L=NULL;
```

```
                    break;
          }
         return (int)ceil((av1+av2+av3+av4+av5)/5);
}
int main(){
    srand(time(0));
       int COB, Q, RT, M, PO,HerD;
       double PUT;
    cout << "\n\n\n\n\n\t\t\tHow many iteration do you need?\n\n\t\t\t\t ";
    cin>>times;
    if (cin.good() == false) {
    cout << "\n\n\n\n\n\t\t\tThat was not an integer." << endl;
    cout<<"\n\n\n\n\n\t\t\tPress any key to EXIT";
    getch();
    return 0;
    }
    cout << "\n\n\t\t\tIt is working Please wait ...";
    if(!fout) {
    cout << "Cannot open output file.\n";
    return 1;
       }
    fout<<"CowId\tParity\tFarm\tGeneticG.\tYear\tSeason";
    fout<<"\tLLength\tDMY\t305MY\tLMilk"<<endl;
    for(counter=0; counter<times;counter++){
              Print(CowNumber());
              COB=ParityNumber();
              Print(COB);
              HerD=Herd();
              Print(HerD);
```

```
            Q=GeneticGroup();
            Print(Q);
            RT=CalvingYear();
            Print(RT);
            M=CalvingSeason();
            Print(M);
            PO=LactationLength(M, RT, HerD, COB, Q);
            Print(PO);
            PUT= DailyMilkYield(M, RT, HerD, COB, Q);
            Print(PUT);
            Print(ThreeMilkYield(PUT));
            Print(LactationMilkYield(PO, PUT));
            fout<<endl;
    }
     fout.close();
    // fin.close();
    return 0;
}
```

Appendix C

Iteration Number Calculation

The calculation of total iteration number for bootstrap and Monte Carlo simulation based on the results obtained using six iteration presented as follow:

Statistic	Bootstrap	Monte Carlo
Mean	4.862065	6.79235
Standard deviation	1.128608	1.559416
t value	2.57058	
Percentage error	1%	

Based on equation 3.38 the iteration number for bootstrap becomes 3564 and 3483 for Monte Carlo simulation.

Appendix D

Influence Analysis

D.1 Influence Analysis for Monte Carlo Simulation

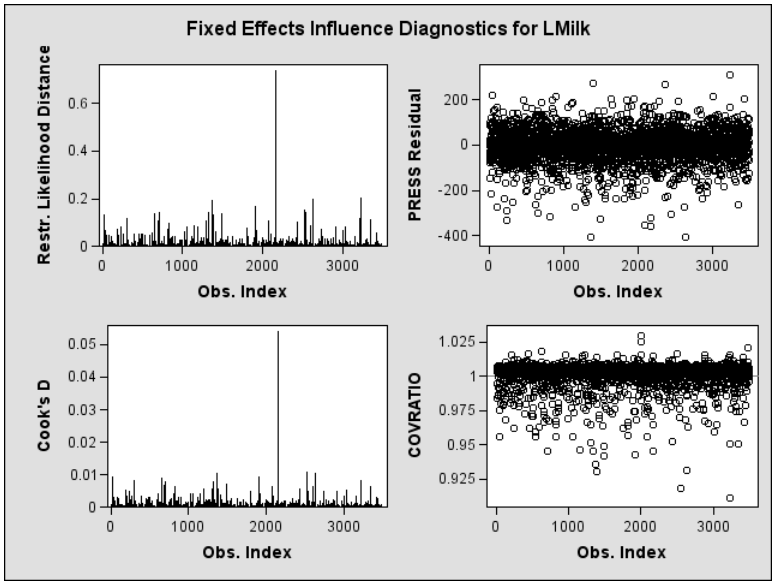


Figure D.1: Influence diagnostic for lactation milk yield

D.2 Influence Analysis for Bootstrap Simulation

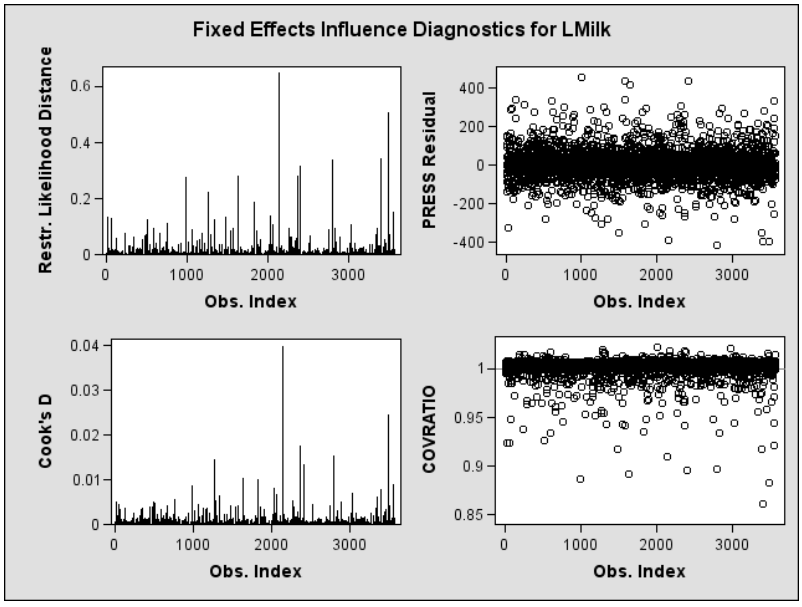
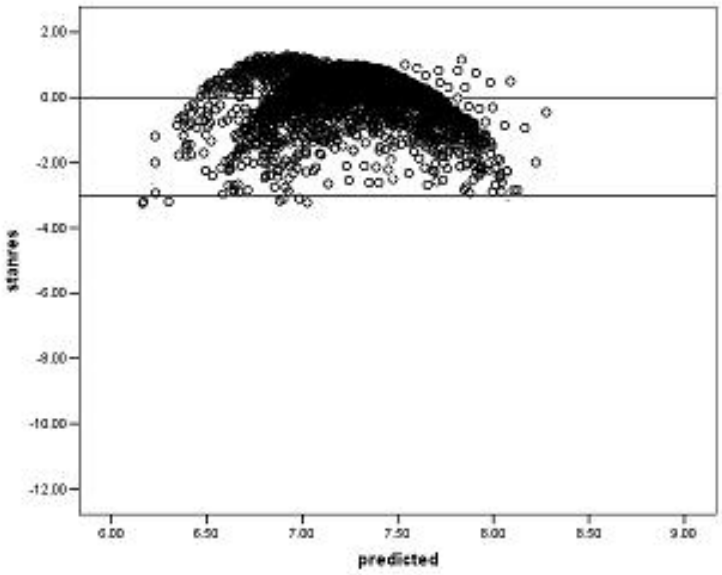


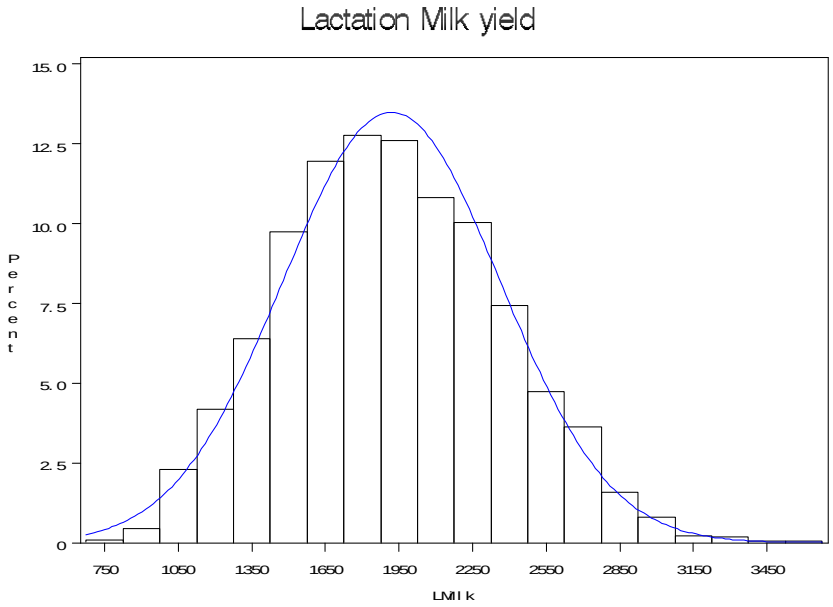
Figure D.2: Influence diagnostic for lactation milk yield

Appendix E

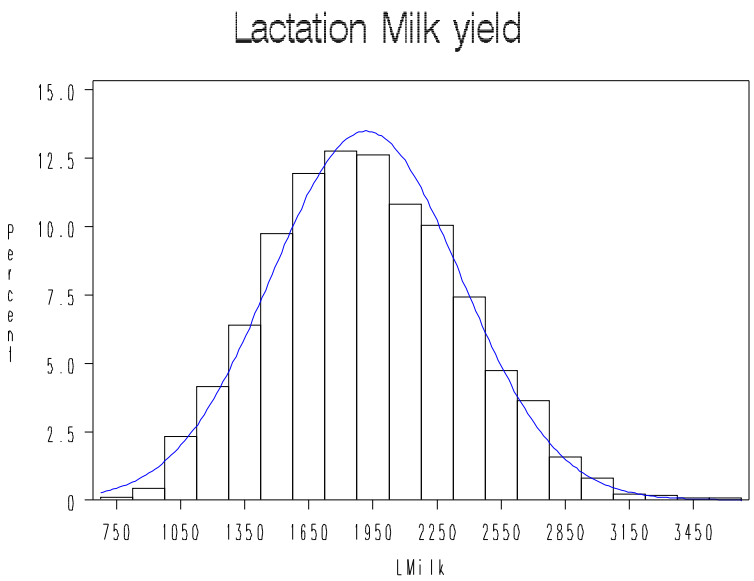
Residual Analysis

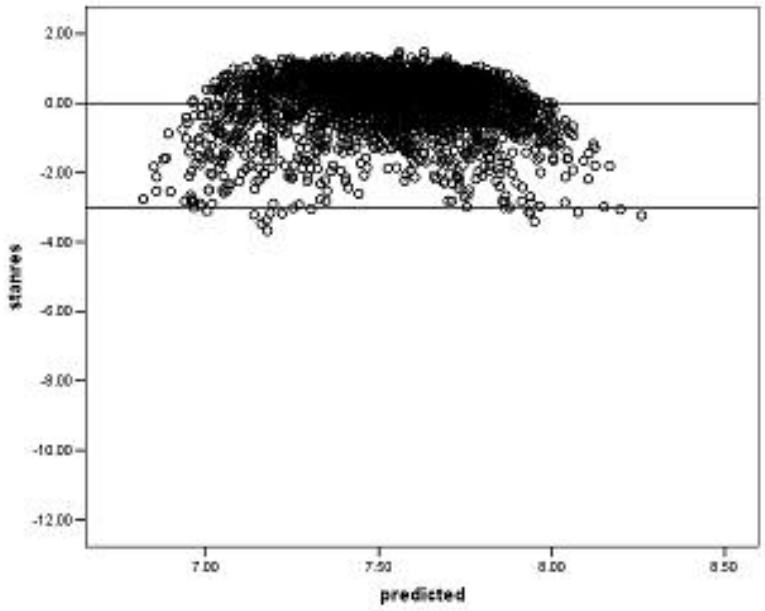
E.1 Residual analysis for Bootstrap Simulation





E.2 Residual Analysis for the Monte Carlo Simulation





Declaration

I, the undersigned, hereby declare that this thesis entitled, "Comparison of Simulation Techniques on Milk Yield Data: A linear mixed model approach" is my own work, and that all the sources I have used or quoted have been indicated or acknowledge by means of completed references.

Name	Signature and date