CHAPTER -3
Research Methodology

Today, science and technology require high performance hardware and high quality software in order to make improvements and achieve breakthroughs. Software Reliability is an useful measure in planning and controlling the resources during the development process so that high quality software can be developed. Software Reliability was defined as the probability of software not causing failure of a system for a limited duration under specified conditions. Even though the definition looks very simple it constitutes a wide range of research activities with different sub activities. The sub activities are grouped in to number of fields, they are technological assessment of software reliability, quality concern of activity, management activity of project and selection of suitable software.

There are many probabilistic and statistical approaches to modelling software reliability. Software reliability estimates are used for various purposes: during development, to make the release decision; and after the software has been taken into use, as part of system reliability estimation, as a basis of maintenance recommendations, and further improvement, or a basis of the recommendation to discontinue the use of the software. The hardware reliability will continue to change even after the product is delivered however the software reliability continue to change as well as to improved throughout the process development and only before the product is delivered for application. Testing is a common assessment of hardware and software reliability techniques. The results of testing process are used in software growth models to formulate the defect and failure data. They are shown as software reliability measure grouped such as software reliability modelling and software testing.

Non Homogenous Poisson Process (NHPP) is a general class of well developed stochastic process model in reliability engineering. These models are also termed as fault counting models and can be either finite failure or infinite failure models, depending on how they are specified. In these models, the number of failures experienced so far follows the NHPP distribution. The NHPP model class is a close relative of the homogenous Poisson model, the difference is that here the expected
number of failures is allowed to vary with time. Hence, they are useful for both calendar time data as well as for the execution time data. In this thesis, NHPP type of software reliability models and methods for estimating software reliability are used. The NHPP based SRGMs are proved to be quite successful in practical software reliability engineering. Many of the SRGMs assume that each time a failure occurs, the fault that caused it can be immediately removed and new faults are not introduced. It is usually called perfect debugging. The mean value function \( m(t) \) is the characteristic of the NHPP model. Its generalization, the model under consideration in the thesis is the Burr type XII model. There are different approaches to modelling software reliability such as, Reliability growth models, Coverage-based models and Component-based models. In this thesis reliability growth models are focused. There is considerable statistical literature on modelling the reliability growth process of finding and fixing defects in a software product.

In this thesis the failure processes that have occurred in small computer programs were studied and the behaviour of the failures was investigated. The concept of well developed and widely applied to software reliability model to small computer programs. All the different data sets are shown in each Chapter. Statistical analysis was performed to all the date set to investigate the behaviour of the variable. In the Reliability analysis the concept of Burr type XII model were applied on data collected and estimating the parameter of the model was performed using maximum likelihood.

Generally, the SRGMs are classified into two groups. The first group contains models, which use machine execution time (i.e., CPU time) or calendar time as a unit of fault detection/removal period. Such models are called Continuous time models. Hence this type of models are also called failure count models. The second group contains models which use the number of test cases as a unit of fault detection period. Such models are called discrete time models, since the unit of software fault detection period is countable. A large number of models have been developed in the first group while there are fewer in the second group. In this thesis, we explore a broad class of NHPP models based on Continuous distributions. In case of interval domain data, it is consider \( k \) predetermined time intervals, denoted by \([t_{i-1}, t_i)\) for \( i = (1, 2, \ldots, k)\). The failure data consists of the number of failures per time
interval, denoted by \( y_i \) for \( i = (1,2,\ldots,k) \). The total number of failures is denoted by \( n_k \).


In this thesis the researcher considered two methodologies SPC and SPRT to achieve the reliability of software which has been described in respective Chapters. The Software reliability modelling uses statistical models for the previous historical data of similar projects are used for modelling parameters such as fault density; defect density and defect detection rate of the software that are being used. Some well established software reliability models are Musa execution time model, Goel – Okumoto NHPP model, Putnam model, Jelinski-Moranda model and little wood-verall model.

SPC is a methodology that aims to provide process control in statistical terms. It is to determine the study of how best one can describe and analyze the data and then draw conclusion based on available data. It is used to identify and eliminate errors in software development process and also to improve software reliability. The concepts of SPC are used to monitor the performance of a software process over time in order to verify that the process remains in the state of statistical control. It helps in finding assignable causes, long term improvements in the software process. Software quality and reliability can be achieved by eliminating the causes or improving the software process or its operating procedures.

Statistical Process Control (SPC) is about using control charts to manage software development efforts, in order to effect software process improvement. The practitioner of SPC tracks the variability of the process to be controlled. The early detection of software failures will improve the software reliability. The selection of proper SPC charts is essential to effective statistical process control implementation and use. The SPC chart selection is based on data, situation and need. Many factors influence the process, resulting in variability. The causes of process variability can be broadly classified into two categories, viz., assignable causes and chance causes.
Critical business application requires reliable software, but developing reliable software is one of the most difficult problems facing the software industry. Therefore consistence and capability of the software is determined by using statistical and process control methods. This thesis reviews various research work performed about using the SPRT and SPC to measure and analyze the software reliability and the research concentrates on statistical techniques and the usage of SPC in Software Reliability.

The Contents of this Chapter is published in “Elixir Comp. Engg. 79 (2015) 30335-30340”. The reprint is appended toward end of this thesis.

In Sequential Probability Ratio Test (SPRT) the classical hypothesis testing, the data collection is executed without analysis and consideration of the data. After all the data is collected and analyzed, conclusions are drawn using sequential analysis. This is a method of statistical inference whose features is that number of observation required by the procedure is not determined in advance of the experiment. The decision to terminate the experiment depends, at each stage, on the results of the observations which are previously made. A merit of sequential method is applied to test the statistical hypothesis, that can be constructed which require, on the average of substantially smaller number of observation that equally reliable to test the procedure which is based on a predetermined number of observations.

The reliability prediction techniques are useful in the level of the software to be developed at early stages of development life cycle. A major problem of software reliability prediction model is that they fail to predict the reliability accurately. Reason for the same being its limitations to particular organization and particular product. Hence customization of particular model is not possible.

The Contents of this Chapter is published in International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 6 (2), 2015, 1806-1811. The reprint is appended toward end of this thesis.
4.1 Introduction

Software reliability is one of the most important characteristics and is a key part in software quality. Its measurement and management technologies employed during the software life cycle are essential for producing and maintaining quality/reliable software systems. Software reliability is the probability that given software functions without failure in a given environmental condition during a specified time. That is it is the probability of failure-free execution of the software for a specified time in a specified environment. Software reliability can be improved by increasing the testing effort and by correcting detected faults. Reliability tends to change continuously during testing due to the addition of problems in new code or to the removal of problems by debugging errors. There are two important parts to provide reliability: fault detection and fault isolation. The design has to consider both aspects. Since performance requirements influence the selection of data structures and algorithms, it is important to check performance factors at the design phase.

To estimate the performance of the design, the information on usage pattern, design structure, and installation characteristics are needed. The specifications describe the level and what security looks like while design considers its implementation. So, good engineering methods can largely improve software reliability. The study of software reliability can be categorized into three parts: modelling, measurement and improvement. Software reliability modelling has matured to the point that meaningful results can be obtained by applying suitable models to the problem. There are many models exist, one of the well-known and simplest model is our Burr type XII model.

There exist several software reliability growth models which can be used during the testing phase of the software development process to estimate the software reliability. Most software reliability models contain the following parts: assumptions, factors, and a mathematical function that relates the reliability with the factors. The mathematical function is usually higher order exponential or
logarithmic. Software modelling techniques can be divided into two subcategories: prediction modelling and estimation modelling. Both kinds of modelling techniques are based on observing and accumulating failure data and analyzing with statistical inference.

The content of this chapter is published in the following journal.


4.2 Model Formulation

Software reliability models can be classified according to probabilistic assumptions. When a Markov process represents the failure process; the resultant model is called Markovian Model. Second one is fault counting model which describes the failure phenomenon by stochastic process like Homogeneous Poisson Process (HPP), Non Homogeneous Poisson Process (NHPP) and Compound Poisson Process etc. A majority of failure count models are based upon NHPP described in the following lines.

A software system is subjected to failures at random times caused by errors present in the system. Let \{N(t), t > 0\} be a counting process representing the cumulative number of failures by time ‘t’. Since there are no failures at t=0 we have

\[ N(0) = 0 \]

It is to assume that the number of software failures during non-overlapping time intervals do not affect each other. In other words, for any finite collection of times \[ t_1 < t_2 < ... < t_n \]. The ‘n’ random variables \( \{N(t_1) - N(t_i)\},...\{N(t_n) - N(t_{n-1})\} \) are independent. This implies that the counting process \{N(t), t>0\} has independent increments.

Let \( m(t) \) represents the expected number of software failures by time ‘t’. The mean value function \( m(t) \) is finite valued, non-decreasing, non-negative and bounded with the boundary conditions.
\[ m(t) = \begin{cases} 
0, & t = 0 \\
a, & t \rightarrow \infty 
\end{cases} \]

Where ‘a’ is the expected number of software errors to be eventually detected.

Suppose \( N(t) \) is known to have a Poisson probability mass function with parameters \( m(t) \) i.e.,

\[ P\{N(t) = n\} = \frac{[m(t)]^n e^{-m(t)}}{n!}, n = 0,1,2 \ldots \infty \]

then \( N(t) \) is called an NHPP. Thus the stochastic behavior of software failure phenomena can be described through the \( N(t) \) process. Various time domain models have appeared in the literature (Kantam and Subbarao, 2009) which describe the stochastic failure process by an NHPP which differ in the mean value function \( m(t) \).

The proposed mean value function \( m(t) \) of Burr Type XII model is given by

\[ m(t) = a \left[ 1 - (1 + t^c)^{-b} \right] \quad (4.2.1) \]

Where \([m(t)/a]\) is the cumulative distribution function of Burr type XII distribution for the present choice.

\[ p\{N(t) = n\} = \frac{[m(t)]^n e^{-m(t)}}{n!} \]

\[ \lim_{n \rightarrow \infty} P\{N(t) = n\} = \frac{e^{-a} a^n}{n!} \]

This is also a Poisson model with mean ‘a’.

Let \( N(t) \) be the number of errors remaining in the system at time ‘t’.

\[ N(t) = N(\infty) - N(t) \]

\[ E[N(t)] = E[N(\infty)] - E[N(t)] \]

\[ = a - m(t) \]

\[ = a - a \left[ 1 - (1 + t^c)^{-b} \right] \]

\[ = a (1 + t^c)^{-b} \]
Let $S_k$ be the time between $(k-1)^{th}$ and $k^{th}$ failure of the software product. Let $X_k$ be the time up to the $k^{th}$ failure. Let us find out the probability that time between $(k-1)^{th}$ and $k^{th}$ failures, i.e., $S_k$ exceeds a real number ‘s’ given that the total time up to the $(k-1)^{th}$ failure is equal to $x$.

i.e., $P[S_k > \frac{s}{X_{k-1}} = x]$

$$R \frac{S_k}{X_{k-1}}(s/x) = e^{-[m(x+s)-m(s)]}$$ (4.2.2)

This Expression is called Software Reliability.

### 4.3 Illustrating the Maximum Likelihood Estimation

The parameters ‘a’, ‘b’ and ‘c’ are estimated by using Maximum Likelihood method and the values can be computed using iterative method for the given cumulative interval domain data. Using the estimators of ‘a’, ‘b’ and ‘c’ we can compute $m(t)$.

#### Mathematical derivation for parameter estimation

We propose to access the software reliability based on Burr Type XII distribution model. The Burr distribution has a flexible shape and controllable scale and location which makes it appealing to fit to data. It is frequently used to model insurance claim sizes (Hee-cheul Kim, 2013). The mean value function and intensity function of Burr Type XII NHPP model are as follows.

The Cumulative distribution function (CDF) is given by

$$m(t) = \int_0^t \lambda(t) dt = a \left[1 - (1 + t^c)^{-b} \right]$$

$$= a F(t)$$

The Probability Density Function (PDF) of Burr XII distribution are given, respectively by

$$\lambda(t) = a \left[ \frac{cbt^{c-1}}{(1+t^c)^{b+1}} \right] = a f(t)$$
Where $t>0$, $a>0$, $b>0$ and $c>0$ denote the expected number of faults that would be detached given infinite testing time in case of finite failure NHPP models. In order to have an assessment of the software reliability, $a$, $b$ and $c$ are unknown parameters and estimated by using Newton Raphson method. Expressions are now delivered for estimating ‘$a$’, ‘$b$’ and ‘$c$’ for the Burr type XII model.

The Log Likelihood function of Interval domain data is given by:

$$LogL = \sum_{i=1}^{k} (n_i - n_{i-1}) \log \left[ m(t_i) - m(t_{i-1}) \right] - m(t_k)$$  \hspace{1cm} (4.3.1)

Take the mean value function of Burr Type XII is of the form

$$m(t) = a \left[ 1 - (1 + t^c)^{-b} \right]$$  \hspace{1cm} (4.3.2)

By substituting Equation (4.3.2) in the above Equation (4.3.1), we get

$$LogL = \sum_{i=1}^{k} (n_i - n_{i-1}) \log \left\{ a \left[ 1 - (1 + t_i^c)^{-b} \right] - a \left[ 1 - (1 + t_{i-1}^c)^{-b} \right] - a \left[ 1 - (1 + t_k^c)^{-b} \right] \right\}$$

$$= \sum_{i=1}^{k} (n_i - n_{i-1}) \log a \left[ (1 + t_i^c)^{-b} - (1 + t_{i-1}^c)^{-b} \right] - a + a \left[ 1 + t_k^c \right]^{-b}$$

$$Log L = \sum_{i=1}^{k} (n_i - n_{i-1}) \left\{ \log a + \log \left[ (1 + t_i^c)^{-b} - (1 + t_{i-1}^c)^{-b} \right] \right\} - a + a \left[ 1 + t_k^c \right]^{-b}$$  \hspace{1cm} (4.3.3)

The parameter ‘$a$’ is estimated by taking the partial derivative of Log L w.r.t ‘$a$’ and equating to ‘0’. (i.e., $\frac{\partial \log L}{\partial a} = 0$)

$$\frac{\partial \log L}{\partial a} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left\{ \frac{1}{a} + 0 \right\} - 1 + \left(1 + t_k^c\right)^{-b}$$

$$\frac{\partial \log L}{\partial a} = 0$$
By simplifying the Equation (4.3.3), we get

\[ \sum_{i=1}^{k} \frac{(n_i - n_{i-1})}{a} = 1 - \left(1 + t_i^c\right)^{-b} \]

\[ \therefore a = \sum_{i=1}^{k} (n_i - n_{i-1}) \left(1 + t_i^c\right)^b \left(1 + t_i^c\right)^b - 1 \]

(4.3.4)

By simplifying the Equation (4.3.3), we get

\[ \log L = \sum_{i=1}^{k} (n_i - n_{i-1}) \left\{ \log a + \log \left[ \left(1 + t_{i-1}^c\right)^{-b} - \left(1 + t_i^c\right)^{-b} \right] \right\} - a + a\left(1 + t_i^c\right)^b \]

\[ \log L = \sum_{i=1}^{k} (n_i - n_{i-1}) \left\{ \log a + \log \left[ \frac{1}{(1 + t_{i-1}^c)^p} - \frac{1}{(1 + t_i^c)^p} \right] \right\} - a + a\left(1 + t_i^c\right)^b \]

\[ \log L = \sum_{i=1}^{k} (n_i - n_{i-1}) \left\{ \log a + \log \left[ \left(1 + t_i^c\right)^b - \left(1 + t_{i-1}^c\right)^b \right] - b \log(1 + t_{i-1}^c) - b \log(1 + t_i^c) \right\} - a + a\left(1 + t_i^c\right)^b \]

The parameter ‘b’ is estimated by using Newton Raphson iterative Method

\[ b_{n+1} = b_n - \frac{g(b_n)}{g'(b_n)} \]

which is substituted in finding ‘a’. Where \( g(b) \) & \( g'(b) \) are expressed as follows.

Taking the Partial derivative of \( \log L \) w.r.t ‘b’ and equating to ‘0’.

\[ \frac{\partial \log L}{\partial b} = 0 \]

\[ \frac{\partial \log L}{\partial b} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ 0 - \log \left(1 + t_{i-1}^c\right) - \log \left(1 + t_i^c\right) + \frac{\left(1 + t_i^c\right)^b \log(1 + t_i^c) - \left(1 + t_{i-1}^c\right)^b \log(1 + t_{i-1}^c)}{\left(1 + t_i^c\right)^b - \left(1 + t_{i-1}^c\right)^b} \right] + a \frac{1}{(1 + t_i^c)^p} \log \frac{1}{(1 + t_i^c)} \]
Substituting Equation (4.3.4) in the above equation, we get

\[
\frac{\partial \text{Log } L}{\partial b} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ -\log \left( 1 + t_i^c \right) - \log \left( 1 + t_i^e \right) + \frac{\left( 1 + t_i^c \right)^b \log \left( 1 + t_i^c \right) - \left( 1 + t_i^e \right)^b \log \left( 1 + t_i^e \right)}{\left( 1 + t_i^c \right)^b - \left( 1 + t_i^e \right)^b} \right] + \sum_{i=1}^{k} (n_i - n_{i-1}) \cdot \frac{1}{(1 + t_i^c)^b - 1} \cdot \frac{1}{(1 + t_i^c)^b} \cdot \log \frac{1}{1 + t_i^c}
\]

Let \( c=1 \), we get

\[
\frac{\partial \text{Log } L}{\partial b} = g(b) = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ -\log \left( t_i+1 \right) - \log \left( t_i+1 \right) + \frac{(t_i+1)^b \log(t_i+1) - (t_{i-1}+1)^b \log(t_{i-1}+1)}{(t_i+1)^b - (t_{i-1}+1)^b} \right] + \left[ \frac{1}{(t_i+1)^b - 1} \log \frac{1}{1+t_i} \right]
\]

\[
g(b) = \frac{\partial \text{Log } L}{\partial b} = 0
\]

Again taking the Partial derivative of \( g(b) \) w.r.t ‘b’ and equating to ‘0’.

\[
g'(b) = \frac{\partial^2 \text{Log } L}{\partial b^2} = 0
\]
\[
\frac{\partial^2 \text{LogL}}{\partial b^2} = g'(b) = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ \frac{(t_{i-1}+1)^b (t_i+1)^b \log(t_i + 1) \log \left( \frac{t_{i-1}+1}{t_i+1} \right)}{[t_i+1]^b - (t_{i-1}+1)^b]^2} \right] + \\
\sum_{i=1}^{k} (n_i - n_{i-1}) \log(t_i + 1) \left[ \frac{(t_k+1)^b \log(t_k + 1)}{[t_k+1]^b - 1]^2} \right]
\]

\[
\frac{\partial^2 \text{LogL}}{\partial b^2} = g'(b) = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ \frac{2(t_{i-1}+1)^b (t_i+1)^b \log(t_i + 1) \log \left( \frac{t_{i-1}+1}{t_i+1} \right)}{[t_i+1]^b - (t_{i-1}+1)^b]^2} \right] + \\
\sum_{i=1}^{k} (n_i - n_{i-1}) \log(1+t_k) \left[ \frac{(t_k+1)^b \log(t_k + 1)}{[t_k+1]^b - 1]^2} \right]
\] (4.3.6)

The parameter 'c' is estimated using Newton Raphson iterative Method

\[
c_{n+1} = c_n - \frac{g(c_n)}{g'(c_n)}
\]

Where \( g(c) \) and \( g'(c) \) are expressed as follows.

\[
g(c) = \frac{\partial \text{LogL}}{\partial c} = 0
\]

\[
\text{Log L} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ \log a + \log \left( (1+t_i^c) - (1+t_{i-1}^c) \right) - b \log \left( 1+t_{i-1}^c \right) - b \log \left( 1+t_i^c \right) \right]
\]

\[-a + a \left( 1+t_i^c \right)^b\]
\[ \frac{\partial \text{LogL}}{\partial c} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ 0 - \frac{b}{(1+t_{i-1}^c)} \cdot t_{i-1}^c \cdot \log t_{i-1} - \frac{b}{(1+t_{i}^c)} \cdot t_{i}^c \cdot \log t_{i} + \frac{1}{(1+t_{i}^c) - (1+t_{i-1}^c)} \cdot b \left(1+t_{i}^c\right)^{b-1} \cdot t_{i}^c \cdot \log t_{i} - b \left(1+t_{i-1}^c\right)^{b-1} \cdot t_{i-1}^c \cdot \log t_{i-1} \right] \]

Substituting Equation (4.3.4) in the above equation, we get

\[ \frac{\partial \text{LogL}}{\partial c} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ - \frac{b}{(1+t_{i-1}^c)} \cdot t_{i-1}^c \cdot \log t_{i-1} - \frac{b}{(1+t_{i}^c)} \cdot t_{i}^c \cdot \log t_{i} + \frac{1}{(1+t_{i}^c) - (1+t_{i-1}^c)} \cdot b \left(1+t_{i}^c\right)^{b-1} \cdot t_{i}^c \cdot \log t_{i} - b \left(1+t_{i-1}^c\right)^{b-1} \cdot t_{i-1}^c \cdot \log t_{i-1} \right] \]

Again substitute \( b=1 \), we get

\[ \frac{\partial \text{LogL}}{\partial c} = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ - \log t_{i-1} \cdot \frac{t_{i-1}^c}{(1+t_{i-1}^c)} - \log t_{i} \cdot \frac{t_{i}^c}{(1+t_{i}^c)} + \frac{1}{(t_{i}^c - t_{i-1}^c)} \cdot t_{i}^c \cdot \log t_{i} - t_{i-1}^c \cdot \log t_{i-1} \right] \]

\[ - \sum_{i=1}^{k} (n_i - n_{i-1}) \cdot \frac{1}{(1+t_{i}^c)} \cdot \log t_{i} \]

\[ \frac{\partial \text{LogL}}{\partial c} = g(c) = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ - \log t_{i-1} \cdot \frac{t_{i-1}^c}{(1+t_{i-1}^c)} - \log t_{i} \cdot \frac{t_{i}^c}{(1+t_{i}^c)} + \frac{t_{i}^c \cdot \log t_{i} - t_{i-1}^c \cdot \log t_{i-1}}{(t_{i}^c - t_{i-1}^c)} \right] \]

\[ - \sum_{i=1}^{k} (n_i - n_{i-1}) \cdot \frac{\log t_{i}}{(1+t_{i}^c)} \]

(4.3.7)
Taking the partial derivative again w.r.t ‘c’ and equating to ‘0’.

\[ g'(c) = \frac{\partial^2 \log L}{\partial c^2} = 0 \]

\[ \frac{\partial^2 \log L}{\partial c^2} = g'(c) = \sum_{i=1}^{k} (n_i - n_{i-1}) \left[ \left( \log \left( \frac{t_{i+1}}{t_i} \right) \cdot \frac{t_i^c - t_{i+1}^c}{(t_i^c - t_{i-1}^c)^2} \{ \log t_i - \log t_{i-1} \} \right) \right] - \left( \log t_{i+1} \right)^2 \cdot \frac{t_i^c}{(1 + t_i^c)^2} - \left( \log t_i \right)^2 \cdot \frac{t_i^c}{(1 + t_i^c)^2} \]

\[ + \sum_{i=1}^{k} (n_i - n_{i-1}) (\log t_i)^2 \cdot \frac{t_i^c}{(1 + t_i^c)^2} \]

(4.3.8)

4.4 Data Analysis

Datasets Phase 1 and Phase 2 from Pham (2005)

A set of failure data taken from Pham (2005) given in Table 4.4.1 and 4.4.2.

Datasets Release #1, #2, #3 and #4 from Alan Wood Tandem Computers (1996)

A set of failure data taken from Wood (1996) given in Table 4.4.3 to 4.4.6 consists of the observation time(week), CPU Hours and the number of failures detected per week :defects found.
<table>
<thead>
<tr>
<th>Week Index</th>
<th>Exposure Time (cum. System test hours)</th>
<th>Fault</th>
<th>Cumulative Fault</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>356</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>712</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1068</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>1424</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>1780</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>2136</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>7</td>
<td>2492</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>2848</td>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>3204</td>
<td>1</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>3560</td>
<td>2</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>3916</td>
<td>2</td>
<td>13</td>
</tr>
<tr>
<td>12</td>
<td>4272</td>
<td>2</td>
<td>15</td>
</tr>
<tr>
<td>13</td>
<td>4628</td>
<td>4</td>
<td>19</td>
</tr>
<tr>
<td>14</td>
<td>4984</td>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>15</td>
<td>5340</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>16</td>
<td>5696</td>
<td>0</td>
<td>22</td>
</tr>
<tr>
<td>17</td>
<td>6052</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>18</td>
<td>6408</td>
<td>1</td>
<td>24</td>
</tr>
<tr>
<td>19</td>
<td>6764</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>20</td>
<td>7120</td>
<td>0</td>
<td>24</td>
</tr>
<tr>
<td>21</td>
<td>7476</td>
<td>2</td>
<td>26</td>
</tr>
<tr>
<td>Week Index</td>
<td>Exposure Time (cum. System test hours)</td>
<td>Fault</td>
<td>Cumulative Fault</td>
</tr>
<tr>
<td>------------</td>
<td>---------------------------------------</td>
<td>-------</td>
<td>------------------</td>
</tr>
<tr>
<td>1</td>
<td>416</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>832</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>1248</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>1664</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>5</td>
<td>2080</td>
<td>2</td>
<td>9</td>
</tr>
<tr>
<td>6</td>
<td>2496</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>2912</td>
<td>1</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>3328</td>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>9</td>
<td>3744</td>
<td>4</td>
<td>17</td>
</tr>
<tr>
<td>10</td>
<td>4160</td>
<td>2</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>4576</td>
<td>4</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>4992</td>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>13</td>
<td>5408</td>
<td>5</td>
<td>30</td>
</tr>
<tr>
<td>14</td>
<td>5824</td>
<td>2</td>
<td>32</td>
</tr>
<tr>
<td>15</td>
<td>6240</td>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>16</td>
<td>6656</td>
<td>1</td>
<td>37</td>
</tr>
<tr>
<td>17</td>
<td>7072</td>
<td>2</td>
<td>39</td>
</tr>
<tr>
<td>18</td>
<td>7488</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>19</td>
<td>7904</td>
<td>0</td>
<td>39</td>
</tr>
<tr>
<td>20</td>
<td>8320</td>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>21</td>
<td>8736</td>
<td>1</td>
<td>43</td>
</tr>
</tbody>
</table>
Table 4.4.3. Data Set Release #1 (Alan Wood Tandem Computers -1996)

<table>
<thead>
<tr>
<th>Test Week</th>
<th>CPU Hours</th>
<th>Percent CPU Hours</th>
<th>Defects Found</th>
<th>Predicted Total Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>519</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>968</td>
<td>-</td>
<td>24</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1,430</td>
<td>-</td>
<td>27</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1,893</td>
<td>-</td>
<td>33</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>2,490</td>
<td>-</td>
<td>41</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>3,058</td>
<td>-</td>
<td>49</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>3,625</td>
<td>-</td>
<td>54</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>4,422</td>
<td>-</td>
<td>58</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>5,218</td>
<td>-</td>
<td>69</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>5,823</td>
<td>58</td>
<td>75</td>
<td>98</td>
</tr>
<tr>
<td>11</td>
<td>6,539</td>
<td>65</td>
<td>81</td>
<td>107</td>
</tr>
<tr>
<td>12</td>
<td>7,083</td>
<td>71</td>
<td>86</td>
<td>116</td>
</tr>
<tr>
<td>13</td>
<td>7,487</td>
<td>75</td>
<td>90</td>
<td>123</td>
</tr>
<tr>
<td>14</td>
<td>7,846</td>
<td>78</td>
<td>93</td>
<td>129</td>
</tr>
<tr>
<td>15</td>
<td>8,205</td>
<td>82</td>
<td>96</td>
<td>129</td>
</tr>
<tr>
<td>16</td>
<td>8,564</td>
<td>86</td>
<td>98</td>
<td>134</td>
</tr>
<tr>
<td>17</td>
<td>8,923</td>
<td>89</td>
<td>99</td>
<td>139</td>
</tr>
<tr>
<td>18</td>
<td>9,282</td>
<td>93</td>
<td>100</td>
<td>138</td>
</tr>
<tr>
<td>19</td>
<td>9,641</td>
<td>96</td>
<td>100</td>
<td>135</td>
</tr>
<tr>
<td>20</td>
<td>10,000</td>
<td>100</td>
<td>100</td>
<td>133</td>
</tr>
</tbody>
</table>
Table 4.4.4. Dataset Release #2 (Alan Wood Tandem Computers -1996)

<table>
<thead>
<tr>
<th>Test Week</th>
<th>CPU Hours</th>
<th>Percent CPU Hours</th>
<th>Defects Found</th>
<th>Predicted Total Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>384</td>
<td>-</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>1,186</td>
<td>-</td>
<td>18</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1,471</td>
<td>-</td>
<td>26</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>2,236</td>
<td>-</td>
<td>34</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>2,772</td>
<td>-</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>2,967</td>
<td>-</td>
<td>48</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>3,812</td>
<td>-</td>
<td>61</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>4,880</td>
<td>-</td>
<td>75</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>6,104</td>
<td>-</td>
<td>84</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>6,634</td>
<td>65</td>
<td>89</td>
<td>203</td>
</tr>
<tr>
<td>11</td>
<td>7,229</td>
<td>70</td>
<td>95</td>
<td>192</td>
</tr>
<tr>
<td>12</td>
<td>8,072</td>
<td>79</td>
<td>100</td>
<td>179</td>
</tr>
<tr>
<td>13</td>
<td>8,484</td>
<td>83</td>
<td>104</td>
<td>178</td>
</tr>
<tr>
<td>14</td>
<td>8,847</td>
<td>86</td>
<td>110</td>
<td>184</td>
</tr>
<tr>
<td>15</td>
<td>9,253</td>
<td>90</td>
<td>112</td>
<td>184</td>
</tr>
<tr>
<td>16</td>
<td>9,712</td>
<td>95</td>
<td>114</td>
<td>183</td>
</tr>
<tr>
<td>17</td>
<td>10,083</td>
<td>98</td>
<td>117</td>
<td>182</td>
</tr>
<tr>
<td>18</td>
<td>10,174</td>
<td>99</td>
<td>118</td>
<td>183</td>
</tr>
<tr>
<td>19</td>
<td>10,272</td>
<td>100</td>
<td>120</td>
<td>184</td>
</tr>
<tr>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
### Table 4.4.5. Dataset Release #3 (Alan Wood Tandem Computers - 1996)

<table>
<thead>
<tr>
<th>Test Week</th>
<th>CPU Hours</th>
<th>Percent CPU Hours</th>
<th>Defects Found</th>
<th>Predicted Total Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>162</td>
<td>-</td>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>499</td>
<td>-</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>715</td>
<td>-</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1,137</td>
<td>-</td>
<td>20</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>1,799</td>
<td>-</td>
<td>28</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>2,438</td>
<td>-</td>
<td>40</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>2,818</td>
<td>-</td>
<td>48</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>3,574</td>
<td>71</td>
<td>54</td>
<td>163</td>
</tr>
<tr>
<td>9</td>
<td>4,234</td>
<td>84</td>
<td>57</td>
<td>107</td>
</tr>
<tr>
<td>10</td>
<td>4,680</td>
<td>93</td>
<td>59</td>
<td>93</td>
</tr>
<tr>
<td>11</td>
<td>4,955</td>
<td>98</td>
<td>60</td>
<td>87</td>
</tr>
<tr>
<td>12</td>
<td>5,053</td>
<td>100</td>
<td>61</td>
<td>84</td>
</tr>
</tbody>
</table>

### Table 4.4.6. Dataset Release #4 (Alan Wood Tandem Computers - 1996)

<table>
<thead>
<tr>
<th>Test Week</th>
<th>CPU Hours</th>
<th>Percent CPU Hours</th>
<th>Defects Found</th>
<th>Predicted Total Defects</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>254</td>
<td>-</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>788</td>
<td>-</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1054</td>
<td>-</td>
<td>8</td>
<td>-</td>
</tr>
<tr>
<td>4</td>
<td>1393</td>
<td>-</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>5</td>
<td>2216</td>
<td>-</td>
<td>11</td>
<td>-</td>
</tr>
<tr>
<td>6</td>
<td>2880</td>
<td>-</td>
<td>16</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>3593</td>
<td>-</td>
<td>19</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>4281</td>
<td>-</td>
<td>25</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>5180</td>
<td>-</td>
<td>27</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>6003</td>
<td>53</td>
<td>29</td>
<td>84</td>
</tr>
<tr>
<td>11</td>
<td>7621</td>
<td>67</td>
<td>32</td>
<td>53</td>
</tr>
<tr>
<td>12</td>
<td>8783</td>
<td>78</td>
<td>32</td>
<td>44</td>
</tr>
<tr>
<td>13</td>
<td>9604</td>
<td>85</td>
<td>36</td>
<td>45</td>
</tr>
<tr>
<td>14</td>
<td>10064</td>
<td>89</td>
<td>38</td>
<td>46</td>
</tr>
<tr>
<td>15</td>
<td>10560</td>
<td>93</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>16</td>
<td>11008</td>
<td>97</td>
<td>39</td>
<td>48</td>
</tr>
<tr>
<td>17</td>
<td>11237</td>
<td>99</td>
<td>41</td>
<td>50</td>
</tr>
<tr>
<td>18</td>
<td>11243</td>
<td>99</td>
<td>42</td>
<td>51</td>
</tr>
<tr>
<td>19</td>
<td>11305</td>
<td>100</td>
<td>42</td>
<td>52</td>
</tr>
</tbody>
</table>
4.5 Method of Performance Analysis

The performance of SRGM is judged by its ability to fit the software failure data. The term goodness of fit denotes the question of “How good does a mathematical model fit to the data?” In order to validate the model under study and to assess its performance, experiments on a set of actual software failure data have been performed. The performance evaluation of software reliability growth model is generally measured with sum of square errors (SSE) and correlation index of regression curve equation (R-square). Among them, the model performance is better when SSE is smaller and R-square is close to 1.

SSE is used to describe the distance between actual and estimated number of faults detected totally, which is defined as

$$SSE = \sum_{i=1}^{n} (y_i - m(t_i))^2$$

Where n denotes the number of failure samples in failure data set, $y_i$ denotes the number of faults observed to the moment $t_i$, and $m(t_i)$ denotes the estimated number of faults detected to the time $t_i$ according to the proposed model. The model can provide a better goodness-of-fit when the value of SSE is smaller.

The equation of calculating the value R-square is written as:

$$R - square = \frac{\sum_{i=1}^{n} (\bar{y} - m(t_i))^2}{\sum_{i=1}^{n} (\bar{y} - y_i)^2}$$

Where $\bar{y}$ denotes the mean value of faults detected. The model can provide a better goodness-of-fit when the value of R-square is close to 1.

Solving equations in Section 4.3 by Newton Raphson Method (N-R) method for all the data sets, the iterative solutions for MLEs of a, b, c and the reliabilities of given software failure datasets are shown in Table 4.5.1.

The estimator of the Reliability function from the Equation (4.2.2) at any time x is given by
The Reliability of the Phase 1 System Test data is given by

\[ R_{S_{12}}/X_{11} = e^{-[m(7476/50)]} \]

\[ = e^{-[m(7526)-m(7476)]} \]

\[ = e^{-[25.99202697-25.99201268]} \]

\[ = e^{-[1.42921E-05]} \]

\[ = 0.999985708 \]

The Reliability of the Phase 2 System Test data is given by

\[ R_{S_{5}}/X_{4} = e^{-[m(8736/50)]} \]

\[ = e^{-[m(8786)-m(8736)]} \]

\[ = e^{-[41.58771804-41.58770144]} \]

\[ = e^{-[1.66063E-05]} \]

\[ = 0.999983394 \]

The Reliability of the Release #1 System Test data is given by

\[ R_{S_{12}}/X_{11} = e^{-[m(10000/50)]} \]

\[ = e^{-[m(10050)-m(10000)]} \]

\[ = e^{-[87.52816482-87.52813803]} \]

\[ = e^{-[2.67897E-05]} \]

\[ = 0.999973211 \]
(iv) The Reliability of the Release #2 System Test data is given by
\[ R_{S_{14}}/X_{13}(10272/50) = e^{-[m(50+10272)-m(10272)]} \]
\[ = e^{-[m(10322)-m(10272)]} \]
\[ = e^{-[111.7720845-111.7720517]} \]
\[ = e^{-[0.000033]} \]
\[ = 0.999967139. \]

(v) The Reliability of the Release #3 System Test data is given by
\[ R_{S_{9}}/X_{8}(5053/50) = e^{-[m(50+5053)-m(5053)]} \]
\[ = e^{-[m(5103)-m(5053)]} \]
\[ = e^{-[59.36639277-59.36629509]} \]
\[ = e^{-[0.000098]} \]
\[ = 0.999902327. \]

(vi) The Reliability of the Release #4 System Test data is given by
\[ R_{S_{9}}/X_{8}(11305/50) = e^{-[m(50+11305)-m(11305)]} \]
\[ = e^{-[m(11355)-m(11305)]} \]
\[ = e^{-[42.82880885-42.8287985]} \]
\[ = e^{-[0.0010]} \]
\[ = 0.999989656 \]
Table 4.5.1. Parameter Estimations and Reliabilities of the Software Failure data

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. Of Samples</th>
<th>Estimated Parameters</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>a</td>
<td>b</td>
<td>c</td>
<td></td>
</tr>
<tr>
<td>Pham (2005) Phase 1 Data</td>
<td>21</td>
<td>25.994042</td>
<td>0.978993</td>
<td>1.083116</td>
<td></td>
</tr>
<tr>
<td>Pham (2005) Phase 2 Data</td>
<td>21</td>
<td>41.590454</td>
<td>0.978993</td>
<td>1.083119</td>
<td></td>
</tr>
<tr>
<td>Wood (1996) Release #1 Dataset</td>
<td>20</td>
<td>87.533224</td>
<td>0.978352</td>
<td>1.082376</td>
<td></td>
</tr>
<tr>
<td>Wood (1996) Release #2 Dataset</td>
<td>19</td>
<td>111.77847</td>
<td>0.977674</td>
<td>1.08129</td>
<td></td>
</tr>
<tr>
<td>Wood (1996) Release #3 Dataset</td>
<td>12</td>
<td>59.376054</td>
<td>0.971698</td>
<td>1.051525</td>
<td></td>
</tr>
<tr>
<td>Wood (1996) Release #4 Dataset</td>
<td>19</td>
<td>42.831021</td>
<td>0.977674</td>
<td>1.081287</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5.2. The results on different datasets

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Reliability (t_{n+50})</th>
<th>SSE</th>
<th>R-Square</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pham (2005) Phase 1 Data</td>
<td>0.999985708</td>
<td>2603.5148</td>
<td>1.2859</td>
</tr>
<tr>
<td>Pham (2005) Phase 2 Data</td>
<td>0.999983394</td>
<td>10015.7216</td>
<td>1.3703</td>
</tr>
<tr>
<td>Wood (1996) Release #1 Dataset</td>
<td>0.999973211</td>
<td>20913.2148</td>
<td>0.3464</td>
</tr>
<tr>
<td>Wood (1996) Release #2 Dataset</td>
<td>0.999967139</td>
<td>42882.6562</td>
<td>0.9822</td>
</tr>
<tr>
<td>Wood (1996) Release #3 Dataset</td>
<td>0.999902327</td>
<td>8491.8095</td>
<td>0.8617</td>
</tr>
<tr>
<td>Wood (1996) Release #4 Dataset</td>
<td>0.999989656</td>
<td>5918.2451</td>
<td>1.1753</td>
</tr>
</tbody>
</table>

From the Table 4.5.2 it can be seen that the value of SSE is smaller and the value of R-square is more close to 1. The results indicate that our NHPP Burr type XII model based on fault detection rate fits the data in the given datasets, best and predicts the number of residual faults in software most accurately.
4.6 Conclusion

Software reliability is an important quality measure that quantifies the operational profile of computer systems. This model is primarily useful in estimating and monitoring software reliability, viewed as a measure of software quality. In this thesis the fault detection rate is calculated with the number of faults remaining in the software. Considering the two factors jointly the fault detection rate is more realistic and accurate. we have discussed the performances of 6 datasets by using our new Burr type XII SRGM. The experiment result shows that the Phase 1 data set can provide a better goodness-of-fit compared with other datasets. The reliability of the model over Release #4 data is high among the data sets which were considered. This is a simple method for model validation and is very convenient for practitioners of software reliability.
Program to find unknown parameters $a$, $b$ and $c$ of Burr Type XII using Newton Rapson Method for Interval domain data

```c
#include<stdio.h>
#include<conio.h>
#include<math.h>
#define N 38

double g(double b,int s[],int n[],int sn);
double gdash(double b,int s[],int n[],int sn);
double gc(double c,int s[],int n[],int sn);
double gcdash(double c,int s[],int n[],int sn);

main()
{
    int i,j,k,sk;
    int n[N]={9,13,20,26,31,34,36,41,45,47,51,58,58,63,66,69,72,76,86,89,90,
    int s[N]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,
              26,27,28,29,30,31,32,33,34,35,36,37,38};
    double savg,g1,g2,g3,g4,a;
    double b[25],c[25];
    double f1=0.0,f2,f3,z;
    clrscr();
    //sk=0;
    printf("n********Newton Rapson Method********");
    c[0]=b[0]=1.0;
    i=-1;
    do
    {
        //printf("B iteration");
        i=i+1;
        g1=g(b[i],s,n,s[N-1]);
        g2=gdash(b[i],s,n,s[N-1]);
        b[i+1]=b[i]-(g1/g2);
        printf("\n\nb[%d]=%f b[%d]=%f",i,b[i],i+1,b[i+1]);
    }
```

printf("n\n\n%\t\t b[%d]-b[%d]=%f",i+1,i,fabs(b[i+1]-b[i]));
}
while(fabs(b[i+1]-b[i])>=0.1));
//printf("n Final i value=%d",i);

j=-1;
do{
//printf("n C Iteration");
j=j+1;
g3=gc(c[j],s,n,s[N-1]);
g4=gcldash(c[j],s,n,s[N-1]);
c[j+1]=c[j]-(g3/g4);
printf("n\n\%d=%f",j,c[j],j+1,c[j+1]);
printf("n\n\%d-c[%d]=%f",j+1,j,fabs(c[j+1]-c[j]));}
while(fabs(c[j+1]-c[j])>=0.1);
for(k=1;k<N;k++){

f1=f1+(n[k]-n[k-1]);
}
z=(1+pow(s[N-1],c[j+1]));
f2=f1*pow(z,b[i+1]);
f3=pow(z,b[i+1])-1;
a=f2/f3;
printf("n\n\nb[%d]=%f is the MLE of b=%f",i+1,b[i+1],b[i+1]);
printf("n\n\nc[%d]=%f is the MLE of c=%f",j+1,c[j+1],c[j+1],a);
printf("n\n***************");
getch();}

double g(double b,int s[N],int n[N],int sn)
{
int i;
double d1,d2,e=0.0,gval,d;

double g(double b,int s[N],int n[N],int sn)
double c1,c2,c3,c4,c5=0.0,c6,c7,e1,e2;
for(i=1;i<N;i++)
{
  e=e+(n[i]-n[i-1]);
}
for(i=1;i<N;i++)
{
  c1=(double)(s[i]+1);
  c2=(double)(s[i-1]+1);
  c3=pow(c1,b);
  c4=pow(c2,b);
  c5=c5+((-log(c2)-log(c1))+(((c3*log(c1))-(c4*log(c2)))/(c3-c4)));
}
d1=(double)sn+1;
d2=pow(d1,b);
c6=(log(1/d1)*(1/(d2-1)));
gval=(e*c5)+(e*c6);
printf("ngval=%f",gval);
return gval;
}

double gdash(double b,int s[N],int n[N],int sn)
{
  int i;
  double gdval,c1,c2,c3,c4,c5=0.0;
  double d1,d2,d3,d4,e;
  for(i=1;i<N;i++)
  {
    e=e+(n[i]-n[i-1]);
  }
  for(i=1;i<N;i++)
  {
    c1=(double)(s[i]+1);
    c2=(double)(s[i-1]+1);
c3 = pow(c1,b);
c4 = pow(c2,b);
c5 = c5 + ((2 * (c4 * c3 * log(c2/c1) * log(c1))) / pow((c3-c4),2));
}

d1 = (double)(sn+1);
d2 = pow(d1,b);
d3 = (d2 * log(d1)) / pow((d2-1),2);
d4 = e * log(d1) * d3;
gdval = (e * c5) + d4;
printf("ngdval=%f",gdval);
return gdval;

}

double gc(double c, int s[N], int n[N], int sn)
{
    int i;
    double gcval, c1 = 0.0, c2, c3, c4, c5, c6 = 0.0, c7, p, q, r;
    for (i=1; i<N; i++)
    {
        c1 = c1 + (n[i] - n[i-1]);
    }
    for (i=1; i<N; i++)
    {
        c2 = (double)log(s[i-1]);
        c3 = (double)pow(s[i], c);
        c4 = (double)pow(s[i-1], c);
        c5 = (double)log(s[i]);
        p = (((c3 * c5) - (c4 * c2)) / (c3 - c4));
        q = ((c2 * c4) / (1 + c4));
        r = ((c5 * c3) / (1 + c3));
        c6 = c6 + ((p - q) - r);
    }
    c7 = log(sn) / (1 + pow(sn, c));
    gcval = c1 * (c6 - c7);
double gcdash(double c, int s[N], int n[N], int sn)
{
    int i;
    double gcdval,c1=0.0,c2,c3,c4,c5,c6,c7,c8,c9,c10,c11,c12,t=0.0,mk;
    for(i=1;i<N;i++)
    {
        c1=c1+(n[i]-n[i-1]);
    }
    for(i=1;i<N;i++)
    {
        c2=log(s[i-1])/log(s[i]);
        c3=pow(s[i],c);
        c4=pow(s[i-1],c);
        c5=(c3*c4)/((c3-c4)*(c3-c4));
        c6=log(s[i])-log(s[i-1]);
        c7=log(s[i-1])*log(s[i-1]);
        c8=c4/((1+c4)*(1+c4));
        c9=log(s[i])*log(s[i]);
        c10=c3/((1+c4)*(1+c4));
        t=t+(c2*c5*c6)-(c7*c8)-(c9*c10);
    }
    c11=pow(log(sn),2);
    mk=(1+pow(sn,c)*(1+pow(sn,c)));
    c12=pow(sn,c)/mk;
    gcdval=(c1*t)+(c11*c12);
    printf("\ngc=%f",gcdval);
    return gcdval;
}