

2003

Nonparametric Estimation of the Cumulative Intensity Function for a Nonhomogeneous Service Process

Eric Rozier
College of William and Mary

Follow this and additional works at: <https://scholarworks.wm.edu/honorsthesis>

Recommended Citation

Rozier, Eric, "Nonparametric Estimation of the Cumulative Intensity Function for a Nonhomogeneous Service Process" (2003). *Undergraduate Honors Theses*. Paper 450.
<https://scholarworks.wm.edu/honorsthesis/450>

This Honors Thesis is brought to you for free and open access by the Theses, Dissertations, & Master Projects at W&M ScholarWorks. It has been accepted for inclusion in Undergraduate Honors Theses by an authorized administrator of W&M ScholarWorks. For more information, please contact scholarworks@wm.edu.

**Nonparametric Estimation of the Cumulative Intensity Function for a
Nonhomogeneous Service Process**

Senior Honors Thesis

Presented to

The Faculty of the Department of Computer Science

The College of William & Mary in Virginia

In Partial Fulfillment

Of the Requirements for the Degree

of Bachelor of Science with Honors in Computer Science,

from the College of William and Mary in Virginia

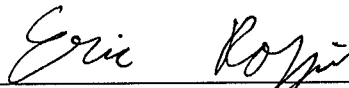
by

Eric William Davis Rozier

2003

Nonparametric Estimation of the Cumulative Intensity Function for a
Nonhomogeneous Service Process

A thesis submitted in partial fulfillment of the requirement for the degree
of Bachelor of Science with Honors in Computer Science,
from the College of William and Mary in Virginia

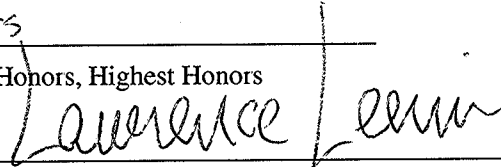


E. W. D. Rozier

Approved, May 2003

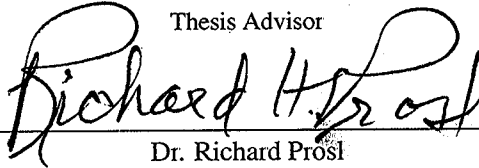
For, Honors

Honors, High Honors, Highest Honors



Dr. Lawrence Leemis

Thesis Advisor



Dr. Richard Prosl



Dr. William Cooke

Department of Physics

Table of Contents

Acknowledgments	vi
List of Figures	vii
List of Algorithms	viii
Abstract	ix
1 Introduction	2
1.1 Simulation and Modeling	2
1.2 Motivation	3
1.3 Methodology	4
2 Background	6
2.1 Nonhomogeneous Poisson Processes	6
2.2 Nonparametric estimation of an NHPP	7
2.3 Overlapping realizations	8
3 Model Formulation for the Exponential Case	11

3.1	Introduction	11
3.2	Properties of NHSP's	13
3.3	Defining and Extracting Idle Regions	14
3.4	A Summary of Forecasting	15
3.5	Estimating Internal Discontinuities	17
3.5.1	Average of Adjacent Service Segments	17
3.5.2	Average of Adjacent Active Regions	18
3.5.3	Weighted Average of Adjacent Active Regions	19
3.5.4	Exponential Smoothing of Adjacent Active Regions	21
3.6	Estimating Initial and Terminal Discontinuities	21
3.7	Estimating $M(t)$ for a Single Realization	23
3.7.1	Constructing the Linear Estimator	23
3.8	Extending $\hat{M}(t)$ for Multiple Realizations	25
3.8.1	Modifying the Linear Estimator	25
3.8.2	Example: Multiple Realizations	25
3.9	Variate Generation	28
4	Areas for Future Research	32
4.1	Including Failure and Repair of the Service Node	32
4.2	Piecewise Methods of Estimation	35
4.3	Model Formulation for Nonexponential Cases	36
4.3.1	General Methods	36
4.3.2	The Erlang Distribution	37

4.3.3	The Chi-square Distribution	37
4.3.4	The Gamma Distribution	38
4.3.5	The Truncated Normal Distribution	39
4.4	Additional Areas	40
5	Summary of Results	42
A	Alternative Methods of Calculation	44
B	Source Code	46
B.1	Introduction	46
B.2	determineIdle.cc	47
B.3	determineIdle-RF.cc	50
B.4	Gamma Variate Generation	53
B.5	parseEventList.pl	56
C	Example Realizations	59
C.1	Realizations for Example 3.8.2	59
	Bibliography	63

ACKNOWLEDGMENTS

I would like to thank my advisor Dr. Lawrence Leemis for his guidance, inspiration, and patience. His unique excitement for this field of study, and constant encouragement made this journey all the more worthwhile.

I would also like to thank Dr. Shandelle Henson for starting me down the long and winding roads of research, Dr. Gianfranco Ciardo and Dr. Evgenia Smirni for teaching me the proverbial ropes, and allowing me the honor of working with them, and my mentors Mrs. Gail Martin and Mrs. Denise Wingfield for all of the support they have given me throughout the long years in all of my endeavors.

Lastly and most importantly, I would like to thank my parents and my loving wife, for their daily support, encouragement, faith, and help in shaping me into the person I am today.

List of Figures

3.1	Initial System	11
3.2	Different Types of Discontinuities	16
3.3	Constructing the Linear Estimator for the Active Regions	23
3.4	Constructing the Linear Estimator for the Idle Regions	24
3.5	The Completed Estimator	24
3.6	Linear Estimator for Example 3.8.2	26
3.7	Linear Estimator for Example 3.8.2	28
3.8	Revised Variate Generation	29
C.1	Number in Node, Realizations Sets 1 and 2	59
C.2	Number in Node, Realizations Sets 3 and 4	60
C.3	Number in Node, Realizations Sets 1 and 2	61
C.4	Number in Node, Realizations Sets 3 and 4	62
C.5	Number in Node, Realization Set 5	62

List of Algorithms

2.1	NHPP Random Variate Generation	8
2.2	NHPP Random Variate Generation Over Multiple Overlapping Realizations	10
3.1	Determining Idle Regions	15
3.2	NHSP Random Variate Generation	30
4.1	Determining Idle Regions with Failures and Repairs	34

ABSTRACT

A nonparametric technique for the estimation of the cumulative intensity function for a nonhomogeneous service process from one or more realizations that may contain idle periods is developed. This technique does not require any arbitrary parameters from the modeler. The estimated cumulative intensity function can be used for the generation of variates for simulation via inversion.

**Nonparametric Estimation of the Cumulative Intensity Function for a
Nonhomogeneous Service Process**

Chapter 1

Introduction

1.1 Simulation and Modeling

Simulation is widely considered to be one of the most useful techniques for the study of complex systems, and has been used for scientific and operational research in almost every field that exists. Biologists often construct computer simulation models to study natural systems, chemists use them to simulate chemical reactions, and physicists build models for everything from virtual circuit boards to astronomical systems. It should not be taken for granted, however, that simulation is only of interest to scientists. In the recent decades an increase in the availability of computers coupled with a decrease in their associated cost has expanded the field of simulation outwards to include such disciplines as economics, international relations, sociology, and many other non-technical fields.

So why is it that simulation has become so popular? One of the primary reasons for the wide use of simulation is the reduction in cost and time that it provides when compared to experiments performed on the real world analogs of artificial systems. Experimentation with actual systems is often prohibitively expensive and can require observations over extremely long periods of time. In addition, performing such studies on actual systems can often be disruptive, often running the risk of causing damage to the system if the proper precautions are not taken (this is especially true in the

case of biological models). In these cases, it is often more advantageous to use known facts about the system in question to construct a model which approximates the actual system, and study this model of the system in lieu of the original system.

While some constructed models can be easily reduced to closed-form solutions, and thus be solved analytically, this is not always the case. Often the solution of a model is extremely difficult, if not impossible to find, at which point we must turn towards a different way of finding the solution, often through simulation. By careful observation of the responses of our model to different inputs, we can estimate properties of the system without having to seek a purely analytical solution.

1.2 Motivation

This study focuses on the analysis of the service times associated with a specific node of a given system. When such a service node is characterized by a stationary service rate, modeling of the service node is often a trivial problem. If the service node were to exhibit a nonstationary rate, however, the model becomes much more complex. The discovery of easily implemented methods for the estimation of these types of service nodes is of great interest to those constructing models which contain nonstationary service rates.

Nonstationary service rates (NSSR) are actually quite common in actual systems, as they exist in any system in which time is a deciding factor for determining the work associated with satisfying a given demand. Causes of nonstationary rates of service can be divided into two categories, job-induced, and node induced.

Job-induced NSSR's indicate a change in the mix of requests arriving at a service node during a given period of time. A good example of this type of effect would be the rush that occurs at restau-

rants during the breakfast, lunch and dinner hours. Other times during the day may be comprised of smaller orders, a drink and a snack, instead of a full course meal, thus taking less processing time for the server. Dinner requests may very well exhibit a trend of an even longer service time as such orders are often more complex than breakfast or lunch.

Node-induced NSSR's are resultant from a change in the characteristic of the service node in question. This sort of cause can be attributed to such effects as fatigue, failure, learning curves, or a variation of the facilities available to the node at a given time. A machine shop whose operators fatigue with time and thus take longer to process jobs at later times during their shift as compared to an earlier and faster rate, is a good example of this sort of cause.

1.3 Methodology

Presented in this thesis is a method for the nonparametric estimation of the cumulative intensity function associated with a process characterized by nonstationary service rates. Probabilistic models are developed for these nonhomogeneous service processes (NHSP), these models are estimated for single and multiple realizations, and algorithms are developed that generate service times based on the estimated cumulative intensity function via inversion.

The models developed in this study indicate possible ways of estimating NHSP's which consist of service times governed by a nonhomogeneous Poisson process. Methods are designed to cope with the specific problems associated with analyzing service times collected from such a process, and examples are presented for the estimation and simulation of systems exhibiting these properties.

Subsequent chapters outline attempts to generalize the technique for more general service times, providing possible methods for the estimation of cumulative intensity functions for NHSP's consist-

ing of rates characterized by distributions more commonly associated with service times.

Chapter 2

Background

2.1 Nonhomogeneous Poisson Processes

While the estimation of service rates which exhibit nonstationary properties has not been extensively studied by the scientific community, a large body of work exists for the estimation of nonstationary arrival rates. As there are numerous similarities between the two processes it is useful to review the probabilistic models and the associated statistical methods. Doing so allows their use them as a foundation for the estimation of NHSP's.

When modeling the time between arrivals, the exponential distribution is often the distribution of choice as it exhibits the unique property of being a memoryless distribution. Thus the forward recurrence time (the time to the next arrival at any given time) is always exponentially distributed. A Poisson processes, can be constructed as a set of iid exponential random variates representing the time between successive arrivals [8].

Nonhomogeneous Poisson processes (NHPPs) are well suited for modeling processes which exhibit a rate that varies over time. Many NHPP estimates involve the creation of piecewise estimators [10] that require the arbitrary selection of points at which to define bins for the collection of arrival statistics.

2.2 Nonparametric estimation of an NHPP

A better approach for the nonparametric estimation of an NHPP was developed in Leemis (1991).

This method is based upon the cumulative intensity function, defined by:

$$\Lambda(t) = \int_0^t \lambda(\tau) d\tau, \quad t > 0,$$

which defines the expected number of events which will have occurred by time t . The intensity function $\lambda(t)$ defines the instantaneous arrival rate at time t . The probability of exactly n events occurring during the interval $(a, b]$ is given by [3]:

$$\frac{[\int_a^b \lambda(t) dt]^n e^{-\int_a^b \lambda(t) dt}}{n!} \quad n = 0, 1, \dots$$

The cumulative intensity function for such a process is estimated using data which represents k realizations of over some interval $(0, S]$, and where n_i , (for $i = 1, 2, \dots, k$) represents the number of observations in the i th realization, and the total number of observations over all realizations is defined as $n = \sum_{i=1}^k n_i$. Furthermore, superpositioning the observations of the k realizations yields the order statistics $t_{(1)}, t_{(2)}, \dots, t_{(n)}$. Denoting the estimator of the cumulative intensity function as $\hat{\Lambda}$, let $\hat{\Lambda}(S) = \frac{n}{k}$ be the average number of events by time S over the k realizations, and $\Lambda(S)$ is the expected population number of events by time S . The piecewise-linear estimator for $\hat{\Lambda}$ is defined as [7]:

$$\hat{\Lambda}(t) = \frac{in}{(n+1)k} + \left[\frac{n(t-t_{(i)})}{(n+1)k(t_{(i+1)}-t_{(i)})} \right], \quad t_{(i)} < t \leq t_{(i+1)}; \quad i = 0, 1, 2, \dots, n.$$

This estimator is able to handle ties within the multiple realizations [7]:

$$\hat{\Lambda}(t_{(m)}) = \hat{\Lambda}(t_{(m+1)}) = \frac{mn}{(n+1)k} \quad \text{and} \quad \lim_{t \downarrow t_{(m+1)}} = \frac{(m+1)n}{(n+1)k}.$$

Based on this estimator, one can easily generate random variates using the process of inversion, transforming the variates generated from a unit Poisson process (E_1, E_2, \dots) into arrivals for the NHPP. The process for generating each event time $T_i = \hat{\Lambda}^{-1}(E_i)$ is described by Algorithm 2.1 [7].

Algorithm 2.1 NHPP Random Variate Generation

```

i ← 1
Ui ∼ U(0, 1)
Ei ← −loge(1 − Ui)
while Ei <  $\frac{n}{k}$  do,
  m ←  $\lfloor \frac{(n+1)kE_i}{n} \rfloor$ 
  Ti ← t(m) + [t(m+1) − t(m)]  $\left( \frac{(n+1)kE_i}{n} - m \right)$ 
  i ← i + 1
  Ui ∼ U(0, 1)
  Ei ← Ei−1 − loge(1 − Ui)
end

```

2.3 Overlapping realizations

The estimation and variate generation method reviewed in Section 2.2 is straightforward for describing NHPPs for whom all realizations are constructed of observations over $(0, S]$, it would be more useful to define a way of constructing such an estimator from realizations on $(0, S]$ with arbitrary

endpoints. This would allow one to construct our estimates based on a wider set of available data, which may be incomplete over portions of the entire interval $(0, S]$.

An extension of [7] was published in 2000, allowing for this case. In this construction the interval $(0, S]$ is partitioned into the minimum number of regions r such that within the bounds of the region $(s_j, s_{j+1}]$ the number of realizations k_{j+1} defined over those values is constant throughout the region. Furthermore n_{j+1} is defined as the number of events which were observed within the region $(s_j, s_{j+1}]$. The union of the order statistics of the superposition of realizations and region boundaries is defined to be $t_{(0)}, t_{(1)}, \dots, t_{(n+r)}$. In order to create a process which estimates the average number of events by time s_{j+1} to be the average number of events during the first $j + 1$ regions, $\hat{\Lambda}$ is defined as [2]:

$$\hat{\Lambda}(s_{j+1}) = \sum_{q=1}^{j+1} \frac{n_q}{k_q}$$

Thus the methods established [7] can be expanded to produce a piecewise-linear estimator for the cumulative intensity function $\hat{\Lambda}$ as such[2]:

$$\hat{\Lambda}(t) = \sum_{q=1}^j \frac{n_q}{k_q} + \frac{(i - \sum_{q=1}^j (n_q + 1))n_{j+1}}{(n_{j+1} + 1)k_j + 1} + \left[\frac{n_{j+1}(t - t_{(i)})}{(n_{j+1} + 1)k_{j+1}(t_{(i+1)} - t_{(i)})} \right],$$

$$t_{(i)} < t \leq t_{(i+1)}; \quad i = 0, 1, \dots, (n + r - 1), \quad s < t \leq s_{j+1}; \quad j = 0, 1, \dots, (r - 1) \quad .$$

In this equation $j + 1$ represents the active region and $i + 1$ represents the active ordered observations.

Variates can be generated from this function in a similar manner to those generated from the earlier approach. Once again we transform a unit Poisson process E_1, E_2, \dots into an NHPP T_1, T_2, \dots via inversion given the superpositioned values $t_{(0)}, t_{(1)}, \dots, t_{(n+r)}$, r, n_1, n_2, \dots, n_r , and k_1, k_2, \dots, k_r as shown in Algorithm 2.2 [2].

Algorithm 2.2 NHPP Random Variate Generation Over Multiple Overlapping Realizations

```

i ← 1
j ← 0
MAX ←  $\sum_{q=1}^r \frac{n_q}{k_q}$ 
Ui ∼ U(0, 1)
Ei ←  $-\log_e(1 - U_i)$ 
while Ei < MAX do,
  while Ei >  $\sum_{q=1}^{j+1} \frac{n_q}{k_q}$  do,
    j ← j + 1
  end
  m ←  $\left\lfloor \frac{(n_{j+1}+1)k_{j+1}(E_i - \sum_{q=1}^j \frac{n_q}{k_q})}{n_{j+1}} \right\rfloor + \sum_{q=1}^j (n_q + 1)$ 
  Ti ← t(m) + [t(m+1) − t(m)]  $\left( \frac{(n_{j+1}+1)k_{j+1}(E_i - \sum_{q=1}^j \frac{n_q}{k_q})}{n_{j+1}} - (m - \sum_{q=1}^j (n_q + 1)) \right)$ 
  i ← i + 1
  Ui ∼ U(0, 1)
  Ei ← Ei-1 −  $\log_e(1 - U_i)$ 
end

```

Chapter 3

Model Formulation for the Exponential Case

3.1 Introduction

The focus of this section will be the adaptation of the techniques discussed in the previous chapter for the modeling of NHSP's. The initial model to be addressed concerns a generalization of a generic system characterized by exponential arrival and service rates as illustrated in 3.1, with λ as the arrival rate and μ as the service rate.



Figure 3.1: Initial System

For this initial system, both the interarrival times and the service times are iid exponential(λ) and iid exponential(μ) random variables respectively. Thus we use the term “exponential case” to describe this model in the title of this chapter. Although not completely descriptive, we also use the title “exponential case” to include time dependant arrival and service rates as described in the next

paragraph.

In order to account for the nonstationary nature of our NHSP, we define μ at any given time t to have the rate $\mu(t)$. The cumulative intensity function for the service process is thus defined by:

$$M(t) = \int_0^t \mu(\tau) d\tau, \quad t > 0,$$

and the probability of the completion of exactly n service events occurring during the interval $(a, b]$ by a single server which is busy during the entire interval $(a, b]$ is given by:

$$\frac{[\int_a^b \mu(t) dt]^n e^{-\int_a^b \mu(t) dt}}{n!}, \quad n = 0, 1, \dots$$

While it is true that most systems are not characterized by exponential service times, the assumption that service rates are exponential is one often made in queuing theory for mathematical tractability [5]. Furthermore it has been shown that queues characterized by exponential service rates often exhibit behavior similar to that exhibited by more complex systems [6]. In addition it has often been noted that systems which make use of a cache (such as a system of proxies, web servers, and clients), often exhibit exponential service times, as when the requested file is currently located in the cache, the service time is zero.

For our initial formulation we will assume that there is only a single realization addressing the issues involved in multiple realizations in Section 3.8.

Throughout this chapter, there is a need to represent lists of order statistics. Each such list is represented by an uppercase variable such as L , and is comprised of the order statistics $(l_{(0)}, l_{(1)}, \dots, l_{(n)})$. The n^{th} ordered element of the list denoted by $L_{(n)}$. Elements are added to the appropriate position in the list via the assignment statement $L \leftarrow L + l_i$, and are removed via the analogous statement

$$L \leftarrow L - l_i.$$

3.2 Properties of NHSP's

NHSP's exhibit several properties which cause them to differ from the previously discussed arrival processes. The first and foremost of these differences is the fact that while an arrival process can never be said to be idle (that is not generating the next arrival event), the same cannot be said of service processes. If at any time the server has no job in service, it can be described as idle, producing a discontinuity in the data. Within a given realization one can define a minimum number of regions r such that within each region (s_j, s_{j+1}) the server is either only idle, or only active.

The cause of these idle regions can be described in many ways, depending on the individual system, but all fall within two general categories: job-induced idle regions, and node-induced idle regions.

Job-induced idle regions occur when the server processes all jobs residing in the queue between two successive arrivals. Because the server is processing jobs faster than they can arrive, an idle period results from the time of the last processed job, to the time of the next job arrival.

Node-induced idle regions occur when the server becomes idle due to a system failure, or other factor which prohibits the node from processing jobs already in the queue. Unfortunately while job-induced idle regions can be identified given data which simply contains arrival and departure information, node-induced idle regions cannot, and require data on the failure and repair times for the system.

When constructing this model, the focus is given primarily to job-induced idle regions. Node-induced idle regions, while beyond the scope of this work, are touched upon briefly in Chapter

4.

While active regions do not prove to be a problem when generating our estimation of the cumulative intensity function \hat{M} , the idle regions do. As no service times were observed over the interval for the idle region in question we cannot make a direct estimation of the service rate in a similar fashion as done in [7] or [2]. These idle regions make it impossible to construct a linear estimator over this idle region. Instead we formulate a way to generate an estimate for the service rate over these idle regions. Doing so is a two-part problem. First we must define a way to extract the idle regions, and second we must define a means by which they can be estimated.

3.3 Defining and Extracting Idle Regions

Algorithm 3.1 defines and extracts idle regions from a given set of data given the ordered arrivals $a_{(0)}, a_{(1)}, \dots, a_{(n)}$, and the ordered departures $d_{(0)}, d_{(1)}, \dots, d_{(n)}$. Observation of the system begins at time 0, and ends at time S .

This algorithm returns the minimal list of all idle regions which occur over the realization. This list is stored in B such that B contains an even number of endpoints, and the ordered event times in B are arranged in such a way that $\forall i, \text{mod}(i, 2) = 0$, the endpoints $B_{(i)}$ and $B_{(i+1)}$ define an idle region over the interval $(B_{(i)}, B_{(i+1)})$.

Once the idle regions are extracted, we find that they fall within two primary categories. Those surrounded by active regions, or internal discontinuities, and those which occur either at the beginning or the end of a realization, defined as initial and terminal discontinuities, as illustrated in Figure 3.2.

Algorithm 3.1 Determining Idle Regions

$A \leftarrow (a_{(1)}, \dots, a_{(n)}),$ $D \leftarrow (d_{(1)}, \dots, d_{(n)}),$ $B \leftarrow \emptyset,$ $n \leftarrow 0,$ $i \leftarrow 0,$ $E \leftarrow A_{(1), \dots, (n)} + D_{(1), \dots, (n)},$ $h \leftarrow 0,$ if ($E_{(0)} \neq 0$), $i \leftarrow 1$ $B \leftarrow B + t_{(0)}$ while ($E \neq \emptyset$) $c \leftarrow E_{(0)},$ $E \leftarrow E - E_{(0)},$ if ($c \in A_{0, \dots, n}$), if ($i = 1$) $B \leftarrow B + c,$ $i \leftarrow 0$ $n \leftarrow n + 1$ else ($c \in D_{0, \dots, n}$), $h \leftarrow c$ $n \leftarrow n - 1$ if ($n == 0$), $B \leftarrow B + c$ $i \leftarrow 1$ if ($c \neq S$), if ($n > 0$), $B \leftarrow B + h$ $B \leftarrow B + t_{(S)}$	Initialize arrival list Initialize departure list Initialize list of idle regions current number in the system test for idle region Initialize the event list last departure event if the first event does not occur at $t_{(0)}$ get the current event Remove current event from list If an arrival Assign right bound of idle region If a departure if server empty if the last event does not occur at $t_{(S)}$ if server isn't empty
---	--

3.4 A Summary of Forecasting

Before identifying the methods which can be used for the estimation of the service rate during a discontinuity, it is helpful to examine some of the basics of the forecasting methods upon which we will base our methods. The discipline of forecasting is primarily concerned with the methods used to make predictions, typically estimating the values of data in the form of a time-series. This definition means that the field of forecasting itself relies not only on quantitative methods, but also qualitative ones as well, requiring the intelligent application of one's judgment to the problem at

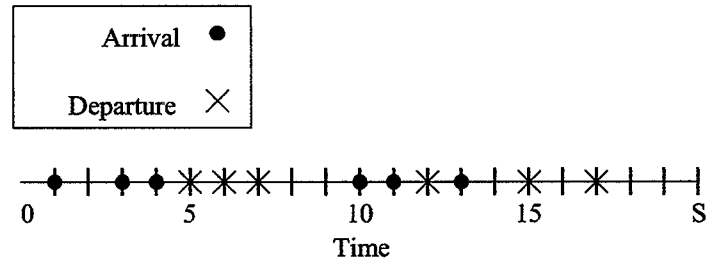


Figure 3.2: Different Types of Discontinuities

hand.

While forecasting is typically concerned with the prediction of the values some function will assume in the future, we have adapted some of these techniques commonly used for such applications, to the estimation of regions of discontinuity. As we wish to have a fully nonparametric approach, we will attempt to eliminate as many of the qualitative aspects of these methods as possible in our adaptations.

When making predictions for a given data set, it is important to realize that many factors play a part in the values which the data takes on. Two primary classifications of these components to the data can easily be established, the first being **signal** comprised primarily of seasonal variations, long-term trends, and cyclical variations. The second is **noise**, which is made up of random variation. It is important to make this distinction, and further more to identify and understand the components which fall into each of these categories, as they provide a useful metric upon which to evaluate our forecasting methods.

The relative level of irregularities within a given time series will alter the effectiveness of forecasts performed on the data. As such it is desirable to perform forecasting on data which has a high **signal to noise ratio**[4].

Based on these fundamentals of forecasting, we have constructed several methods with which to estimate the service rate during an idle period:

1. Average of Adjacent Service Segments
2. Average of Adjacent Active Regions
3. Weighted Average of Adjacent Active Regions
4. Exponential Smoothing of Adjacent Active Regions.

3.5 Estimating Internal Discontinuities

For all of the following methods we create an estimate of the service rate for the given idle region.

This rate, μ_t , can then be used when constructing our linear estimator $\hat{M}(t)$.

3.5.1 Average of Adjacent Service Segments

The first method of estimating μ over an idle region is also the simplest of the methods averaging only the final estimated service rate $\hat{\mu}_a$ performed on the last job before an idle region, and the first estimated service rate $\hat{\mu}_b$ performed on the first job after an idle region. The estimated final service rate $\hat{\mu}_a$ performed on the last job before an idle region is the reciprocal of the final service time. An analogous statement can be made for $\hat{\mu}_b$.

$$\mu_t = \frac{\mu_a + \mu_b}{2}.$$

3.5.2 Average of Adjacent Active Regions

Two faults of the previous method are: (a) it takes into account only two parts of the adjacent active regions and thus may miss long-term trends, and (b) it is unstable due to sampling variability. Our second method is based on the common forecasting technique known as the simple moving average. This method is effective when the service process being evaluated possesses a both a stationary mean, and variance over the evaluated data. Where the simple moving average takes into account past periods, our method is expanded such that it also takes into account the most imminent active region as well, and limits the accounting for past periods to the most recent active region.

Let $\hat{\mu}_a$ be the estimated service rate during the busy period prior to the discontinuity. Let $\hat{\mu}_b$ be the estimated service rate during the busy period after the discontinuity. The service rate during the discontinuity is estimated by $\hat{\mu}$ a weighted average of these estimated service rates. The weights are chosen in such a way that the weight for a single active region is equal to the number of service segments within that active period divided by the total number of service segments which are taken into account.

- n : the number of service segments observed during the previous region
 m : the number of service segments observed during the subsequent region
 S_a : the index of the first arrival during the previous region
 S_b : the index of the first arrival during the subsequent region

$$\hat{\mu}_a = \frac{1}{n} \sum_{i=S_a}^{S_a+n} \mu_i$$

$$\hat{\mu}_b = \frac{1}{m} \sum_{i=S_b}^{S_b+m} \mu_i$$

$$\hat{\mu} = \left(\frac{n}{n+m} \right) \hat{\mu}_a + \left(\frac{m}{n+m} \right) \hat{\mu}_b$$

3.5.3 Weighted Average of Adjacent Active Regions

The next method adds a weighting function to each value of the estimated service rate when estimating the value of an adjacent active period so as to account for the amount of temporal displacement from each observed μ_i to the idle interval, giving more weight to the values of μ_i which are observed nearer to the time at which the idle interval occurs such that

$$w_1 + w_2 + \cdots + w_n = 1$$

and

$$w_0 > w_1 > \cdots > w_n$$

where w_n is the weight for the service rate nearest the idle interval. Thus the new estimate for μ_t becomes:

n : the number of service segments observed during the previous region

m : the number of service segments observed during the next region

$$\begin{aligned}\hat{\mu}_a &= \sum_{i=1}^n \hat{\mu}_i w_i \\ \hat{\mu}_b &= \sum_{i=1}^m \hat{\mu}_i w_{m-i} \\ \hat{\mu} &= \left(\frac{n}{n+m}\right) \hat{\mu}_a + \left(\frac{m}{n+m}\right) \hat{\mu}_b\end{aligned}$$

In order to formulate weights for each of the given service segments, we must first define a metric for comparison of the temporal displacements of each service segment from the idle region. In order to measure this distance, we will define the distance, d_i , as being the distance from a service segment to a given idle region as the distance from the event's initiation to the end of the active region of which the event is a member, which is most proximal to the idle region. Next we must find a way to scale these distances such that:

$$\sum_{i=0}^l d_i' = 1.$$

Where l has the value of either n or m , depending on which adjacent region we are considering. If we define d_{\max} to be the maximum observed value of the unscaled d_i 's we can define an appropriate scaling function as:

$$d_i' = \frac{d_{\max} - d_i + 1}{\sum_{i=0}^l d_{\max} - d_i + 1}$$

using our new scaled distance d_i' as the weight w_i .

3.5.4 Exponential Smoothing of Adjacent Active Regions

This method is a special case of the weighted average method, where the weights of successive points decrease exponentially in magnitude. The value of weight w_i can be computed as follows:

$$w_i = \alpha^i$$

$$0 < \alpha \leq 1$$

where a value of $\alpha = 1$ corresponds to an identical result as the method given in Section 3.5.2.

Choosing a smaller value for α means that the impact of a given service segment will decrease more rapidly as the distance between that service segment and the idle region increases.

n : the number of service segments observed during the previous region

m : the number of service segments observed during the next region

$$\hat{\mu}_a = \frac{1}{1 + \alpha + \alpha^2 + \dots + \alpha^n} \sum_{i=1}^n \hat{\mu}_i \alpha^i$$

$$\hat{\mu}_b = \frac{1}{1 + \alpha + \alpha^2 + \dots + \alpha^m} \sum_{i=1}^m \hat{\mu}_i \alpha^{m-i}$$

$$\hat{\mu} = \left(\frac{n}{n+m} \right) \hat{\mu}_a + \left(\frac{m}{n+m} \right) \hat{\mu}_b$$

3.6 Estimating Initial and Terminal Discontinuities

While thus far we have only taken into account idle regions which are bounded by active regions, two other types of idle regions also exist which can introduce discontinuities into our service model, those that occur at the beginning of the observation period, and those that occur at the end. The

former of these cases of discontinuities is termed an initial discontinuity, while the latter is defined as a terminal discontinuity.

Both of these discontinuities have causes that are similar to those of internal discontinuities, namely that the system is empty, or has failed. In the case of an empty system at start up, it indicates that no jobs were present on the start of the observational period, or that when we began observing service events, a service event was already underway, whose starting time is unknown. In addition it is possible that one might begin observing a system during an interval where it is suffering from a failure. Terminal discontinuities differ from initial discontinuities in that we have full knowledge of service events prior to the discontinuity, rendering the causes identical to the causes of an interior discontinuity.

In this case we cannot interpolate values for these intervals, but must extrapolate them. This means we will only be able to take into account, the most imminent active region (in the case of initial discontinuities), or the most recent active region (in the case of terminal discontinuities). As such our equations for estimation are altered in such a way as to eliminate either $\hat{\mu}_a$ or $\hat{\mu}_b$ respectively. We will call the average service rate which we will consider for our estimations $\hat{\mu}_p$, the average service rate in the most proximal active region.

The methods are identical in all other respects to those used for internal discontinuities, letting $\hat{\mu}_p = \hat{\mu}_a$ for terminal discontinuities, and letting $\hat{\mu}_p = \hat{\mu}_b$ for initial discontinuities.

3.7 Estimating $M(t)$ for a Single Realization

3.7.1 Constructing the Linear Estimator

Once estimates for each idle region have been estimated we can begin construction of the estimator for the cumulative intensity function estimate $\hat{M}(t)$. The estimator $\hat{M}(t)$ is defined as the expected number of events by time t . When constructing our linear estimator, it is useful to first construct separate estimators for each of the active regions in the same manner as was suggested in [7]. Figure 3.3 shows a single realization on $(0, 20]$ with six arrivals, six departures, two busy periods and an initial, internal, and terminal discontinuity. In keeping with the parameter-free approach we use the “average of adjacent active regions” approach when estimating the service rate for the idle periods given in Section 3.5.2.

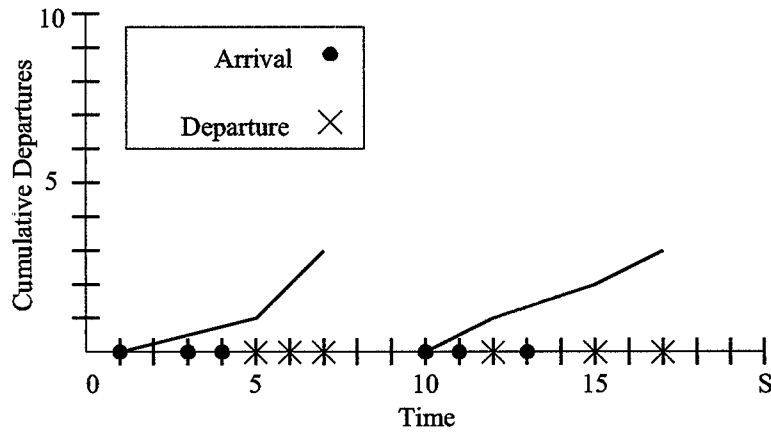


Figure 3.3: Constructing the Linear Estimator for the Active Regions

Once the service rate in these active regions have been estimated we calculate the rate μ_t for each of the idle regions. Given an idle region with end points $(x_i, x_j]$, where $x_i < x_j$, we define the

expected number of events occurring by time x_j to be:

$$\hat{M}(x_j) = \mu_t(x_j - x_i) + \hat{M}(x_i)$$

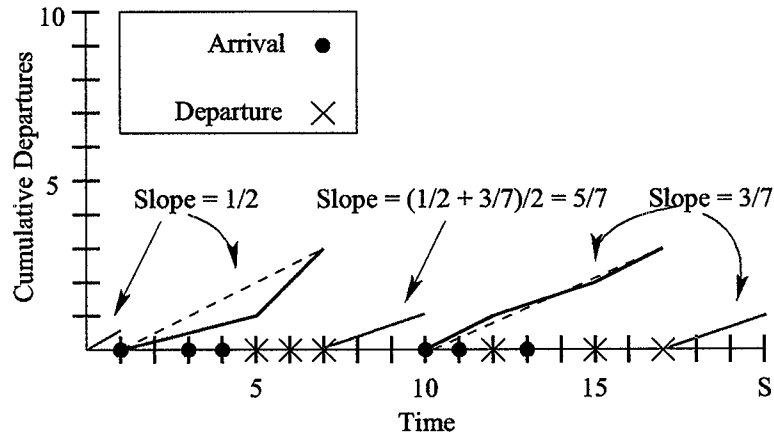


Figure 3.4: Constructing the Linear Estimator for the Idle Regions

As $\hat{M}(x_i)$ will have been previously defined in our estimation of the active regions, we can now completely define the linear estimator of the cumulative intensity function.

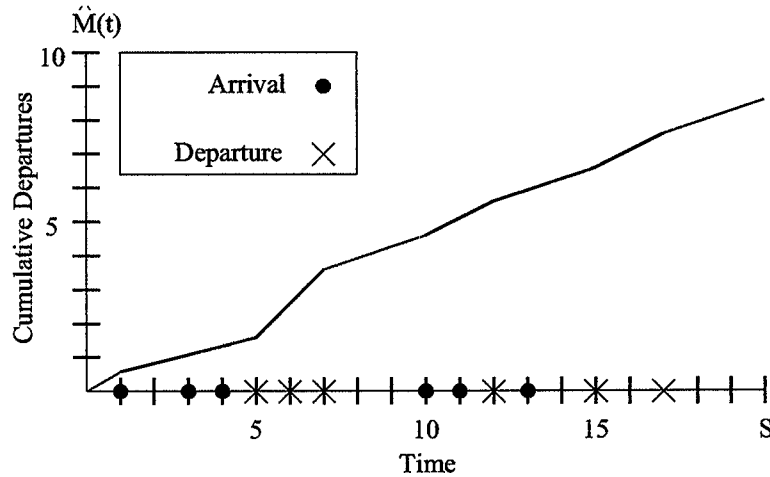


Figure 3.5: The Completed Estimator

3.8 Extending $\hat{M}(t)$ for Multiple Realizations

3.8.1 Modifying the Linear Estimator

Now that an estimator has been defined for a single realization, the extension of this estimator to the case of multiple realizations is straightforward. In order to construct the linear estimator, it is useful to think of each active region within each realization as a separate pseudo-realization in and of itself. In doing so we then superposition the observed events (keeping track of the number of active pseudo-realizations over the various portions of the observed interval $(0, S]$), and then apply Algorithm 3.1 generating the sum of all idle regions over the superpositioned observations. The new active regions are then estimated in accordance to the method described in [2]. That is the value of $\hat{M}(t)$ is the sum of the average number of events for all previous segments of the active region. From this point on construction of the linear estimator is identical to the process followed in section 3.7.1

3.8.2 Example: Multiple Realizations

Two examples are given in this section. The first examines a system with exponential arrivals with rate $\lambda = 1$. The service rate at the node is $\mu(t) = 1$ for $0 \leq t \leq 20$. Event times were generated using a Lehmer random number generator [9]. All exponential random variates for the realizations were generated via thinning. The realizations used for this solution are presented in Appendix B.

The linear estimator was constructed through superpositioning, and it was found that the overlapped multiple realizations featured initial and terminal discontinuities. The cumulative number of departures was estimated for these discontinuities using each of the methods described previously, and all returned nearly the same estimator. It is worthy to note that before performing an estimate

of the service rate over a given active region, it is necessary to sample that region and determine the mean and standard deviation. Due to random sampling and the nature of the exponential distribution, it is likely that some of the rates measured within an active region should be considered outliers. A good test for outliers was found to be the distance from the mean rate for the active region. In the end all rates which were separated from the mean by more than two standard deviations were eliminated from the calculation in an attempt to improve the signal to noise ratio. Figure 3.6 illustrates the estimated cumulative intensity function, based on the four realizations.

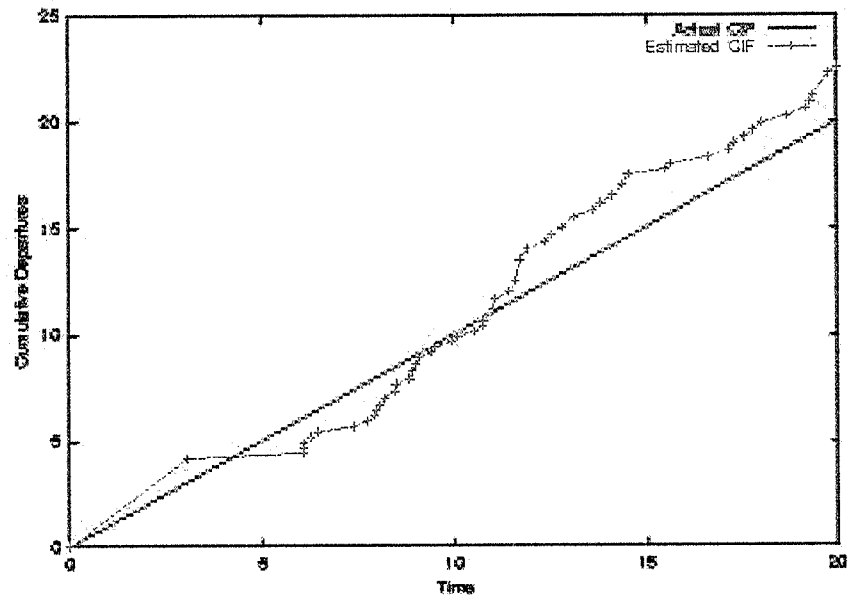


Figure 3.6: Linear Estimator for Example 3.8.2

The second example utilizes a system with exponential arrivals with rate $\lambda = 1$. The service rate at the node is,

$$\mu(t) = \begin{cases} 1 & 0 \leq t < 10, \\ 0.5 & t > 10 \end{cases}.$$

Event times were generated using a Lehmer random number generator [9]. Random event times for the realizations were generated via thinning. The realizations used for this solution are presented in Appendix B. The linear estimator was constructed through superpositioning, and it was found that the overlapped multiple realizations featured initial and terminal discontinuities. The cumulative number of departures was estimated for these discontinuities using each of the methods described previously, and it was found while all methods returned reasonable values, the weighted estimators produced more appropriate values.

Outliers were eliminated in the same fashion as in the previous example. Figure 3.7 illustrates the estimated cumulative distribution function. The intensity functions for the arrival and service process, and the realizations used to construct $\hat{M}(t)$ are given in Appendix B.

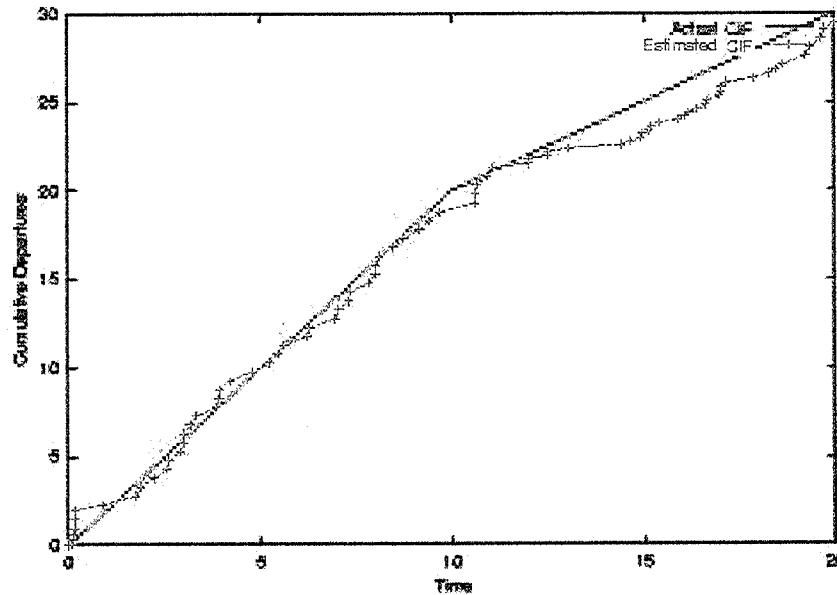


Figure 3.7: Linear Estimator for Example 3.8.2

3.9 Variate Generation

In order to generate random variates through inversion, via the linear estimator of the cumulative intensity function, a similar process is followed as was outlined in [7] and [2]. It is important to note, however, that we cannot generate random variates as if this were a continuous process, as there may be times when the server empties, and thus is idle, generating no service times. Thus the algorithm for variate generation needs to be modified to take into account this possibility of an idle region. This fact is illustrated in Figure 3.8.

While algorithm 2.2 bases the next event time generated from the previous event time, it is clear that something different must be done in this case to account for the idle period during the interval $(I_0, I_1]$. Our next event time must be generated with respect to I_1 , which represents the next arrival

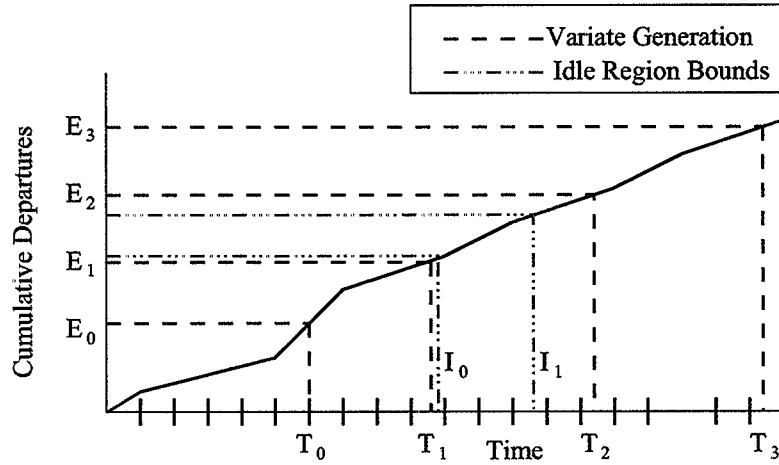


Figure 3.8: Revised Variate Generation

to the system which must be processed, rather than E_1 . To do so it is necessary to define a list of all arrivals on $(0, S]$ as order statistics $A \leftarrow (a_{(1)}, \dots, a_{(n)})$, and pass this as an argument to our algorithm. The previously suggested generators use,

$$E_i = E_{i-1} - \log_e(1 - U_i),$$

to generate a unit Poisson process. We now define $c = A_{(0)}$, that is, the time of the next arrival event, and in the case of an empty system generate the unit Poisson process E_i using the following equation:

$$E_i = \hat{M}(c) - \log_e(1 - U_i).$$

Allowing us to generate our next service time, $T_i = \hat{M}^{-1}(\hat{M}(c) - \log_e(1 - U_i))$ The modified algorithm is presented as algorithm 3.2.

Algorithm 3.2 NHSP Random Variate Generation

```

i ← 1
j ← 0
MAX ←  $\sum_{q=1}^r \frac{n_q}{k_q}$ 
 $U_i \sim U(0, 1)$ 
c ←  $A_{(0)}$  get the first arrival
A ← A − c
n ← 1 start off with the first arrival in the system
 $E_i \leftarrow c - \log_e(1 - U_i)$ 
while ( $E_i < MAX$ )
  while ( $E_i > \sum_{q=1}^{j+1} \frac{n_q}{k_q}$ )
    j ← j + 1
   $m \leftarrow \left\lfloor \frac{(n_{j+1}+1)k_{j+1}(E_i - \sum_{q=1}^j \frac{n_q}{k_q})}{n_{j+1}} \right\rfloor + \sum_{q=1}^j (n_q + 1)$ 
   $T_i \leftarrow t_{(m)} + [t_{(m+1)} - t_{(m)}] \left( \frac{(n_{j+1}+1)k_{j+1}(E_i - \sum_{q=1}^j \frac{n_q}{k_q})}{n_{j+1}} - (m - \sum_{q=1}^j (n_q + 1)) \right)$ 
  n ← n − 1
  c ←  $A_{(0)}$ 
  while (c ≤  $T_i$ )
    A ← A − c
    n ← n + 1
    c ←  $A_{(0)}$ 
  i ← i + 1
   $U_i \sim U(0, 1)$ 
  if (n > 0)
     $E_i \leftarrow E_{i-1} - \log_e(1 - U_i)$ 
  else
    c ←  $A_{(0)}$ 
    A ← A − c
     $E_i \leftarrow \hat{M}(c) - \log_e(1 - U_i)$ 

```

This algorithm is an expansion of algorithm 2.2. It adds functionality for tracking the current number in the node, and maintaining a record of the most imminent arrival in case of an empty node, so that the calculation of the unit Poisson process is modified as previously described to account for these idle periods.

Given this algorithm and the previously described methods for the estimation of $\hat{M}(t)$, it is now possible to construct an estimate of the cumulative intensity functions of nonhomogeneous service processes. Furthermore the methods for the use of these estimates in the generation of random

variates through inversion has been detailed. While the methods provided detail the solution for the exponential case, the next chapter suggests ways to expand this solution to include other types of distributions.

Chapter 4

Areas for Future Research

As with all projects in the sciences, one of the most exciting things about the work itself involves outlining areas for future work. While this study has focused on solving the given problem for the exponential case, many related areas of similar problems suggest themselves as directions for future work.

4.1 Including Failure and Repair of the Service Node

It was suggested in Chapter 3 that one way that the model could be expanded, would be to include the node induced idle regions caused by system failure and repair. While it is beyond the scope of this project to fully examine such a problem, a preliminary algorithm for this purpose can be developed to help encourage further work in this area.

When generating the algorithm it is useful to note that as we are dealing with observed data collected from a single service node, the following properties are true:

- There are exactly four unique types of events which can occur in the system: Arrivals, Departures, Failures, and Repairs.

- Assuming that we begin with an empty working node, we may have no departures until an arrival event has occurred, and likewise may have no repairs until a failure has occurred.
- If a failure has occurred, no more departures may occur until a repair has occurred. Likewise no failure may occur until a repair has occurred.

Algorithm 4.1 utilizes these properties to determine the idle regions over a given observational period.

Algorithm 4.1 Determining Idle Regions with Failures and Repairs

$A \leftarrow (a_{(0)}, \dots, a_{(n)})$ $D \leftarrow (d_{(0)}, \dots, d_{(n)})$ $F \leftarrow (f_{(0)}, \dots, f_{(n)})$ $R \leftarrow (r_{(0)}, \dots, r_{(n)})$ $B \leftarrow 0$ $n \leftarrow 0$ $i \leftarrow 0$ $j \leftarrow 0$ $h \leftarrow 0$ $E \leftarrow A + D + F + R,$ if ($E_{(0)} \neq 0$) $i \leftarrow 1$ $B \leftarrow B + t_{(0)}$ while ($E \neq 0$) $c \leftarrow E_{(0)}$ $E \leftarrow E - E_{(0)}$ if ($c \in A_{0, \dots, n}$) if ($i = 1$) and ($j = 0$) $B \leftarrow B + c$ $i \leftarrow 0$ $n \leftarrow n + 1$ else if ($c \in D_{0, \dots, n}$), $h \leftarrow c$ $n \leftarrow n - 1$ if ($n == 0$) $B \leftarrow B + c$ $i \leftarrow 1$ else if ($c \in F_{0, \dots, n}$) $j \leftarrow 1$ if ($i = 0$) $B \leftarrow B + c$ $i \leftarrow 1$ else if ($c \in R_{0, \dots, n}$) $j \leftarrow 0$ if ($n > 0$) $B \leftarrow B + c$ $i \leftarrow 0$ if ($c \neq S$) if ($n > 0$) $B \leftarrow B + h$ $B \leftarrow B + t_{(S)}$	Initialize arrival list Initialize departure list Initialize failure list Initialize repair list Initialize list of idle regions current number in the system test for idle region test for failure region last departure event Initialize the event list if the first event does not occur at $t_{(0)}$ get the current event Remove current event from list If an arrival Assign right bound of idle region If a departure if server empty If a failure If a repair if the last event does not occur at $t_{(S)}$ if the server isn't empty
--	---

4.2 Piecewise Methods of Estimation

The methods described in Chapter 3 generate a constant rate μ_t for idle regions. It is possible that such a method of estimation could prove less accurate than needed for some systems. In order to refine our estimation procedure, it is also possible to provide a piecewise method of estimating μ_t , by dividing the idle region into a series of equal width time intervals $t_0, \dots, t_{(l-1)}$, and providing a separate estimate of $m\mu_t$ for each of the intervals within the idle region. This varying rate could then be estimated by taking into account a fraction of the service segments in each of the bordering active regions, in a manner similar to that used by the moving average forecasting technique,

n : service times observed during the previous region

m : service times observed during the next region

l : $\min(n, m)$

for $x > 0, x \leq l$

$$\hat{\mu}_a(x) = \frac{1}{l} \sum_{i=n-(l-x)}^n \hat{\mu}_i$$

$$\hat{\mu}_b(x) = \frac{1}{l} \sum_{i=0}^{m-(l-x)} m\hat{\mu}_i$$

$$\hat{\mu}_t(x) = \frac{\hat{\mu}_a(x) + \hat{\mu}_b(x)}{2}$$

This function can be thought of as an analog for the average of adjacent regions which was proposed in Chapter 3, and could thus be expanded upon in a similar manner by including weights for each of the individual μ_i 's.

Further work is necessary to determine if the inclusion of less total service segments in favor of generating a moving average is an acceptable trade off for certain situations.

4.3 Model Formulation for Nonexponential Cases

While the exponential case of service rates is often used in queuing systems, and doing so even for systems which do not exhibit exponential service rates will often produce accurate estimates for the utilization of a service node, the estimates produced for other statistics such as queue length, and response time will often be inaccurate[5]. Ideally we would like a distribution which is closer in behavior to the normal distribution, as it has been shown that service times are often approximately normally distributed [6].

4.3.1 General Methods

When constructing an estimator for nonexponential distributions, a similar approach is used to that of the exponential distribution. The estimate of the cumulative intensity function is constructed in a fashion identical to the methods outlined for the exponential case, the primary difference for such a method occurs during variate generation.

When generating a random variate, our methods use a time transformation from the event times from a unit service process, V_1, V_2, \dots , and transform these times into the event times of an NHSP via $T_i = \hat{M}^{-1}(V_i)$. This means that for our method to work we must be able to define a unit service process for our distribution. The primary requirement for this unit service process is that it has a $\mu = 1$, as this when coupled with $\hat{M}(t)$, will generate an average expected number of events equal to that predicted by our cumulative intensity function. The second requirement is that our unit

service process generate events with the same distribution as the events found in our realizations. The subsequent sections consider the use of several common parametric distributions for producing a more realistic probability model for the service times.

4.3.2 The Erlang Distribution

When modeling service times in a queuing network, and a distribution is required which is closer to real service times than the exponential, the Erlang distribution is often used[5]. Erlang distributions lend themselves to queuing analysis as an Erlang(n, b) can be constructed from a series of n exponential random variables each with a mean of b . Another property of the Erlang which lends itself to our purpose is the fact that the mean and standard deviation of an Erlang random variate are trivial to calculate[10]:

$$\mu = nb$$

$$\sigma = \sqrt{nb}.$$

One of the primary shortcomings of the Erlang distribution for our purposes is the fact that the shape parameter n may only take integer values, restricting our ability to form a unit random variable based on this distribution.

4.3.3 The Chi-square Distribution

Another distribution which initially appears to lend itself to our stated purposes is the Chi-square distribution. For a Chi-square(n) random variable, with $n > 2$, we find that the Chi-square exhibits a bell-shaped probability density function. In addition the mean and standard deviations for a Chi-square random variable are likewise easy to calculate [10]:

$$\mu = n$$

$$\sigma = \sqrt{2n}$$

Unfortunately we find that we cannot formulate a unit Chi-square with the properties we desire. As the mean of a Chi-square(n) equals the parameter n , this would require us to define a unit Chi-square as a Chi-square(1), which has a distribution even worse than the exponential in terms of bell shape. As such our unit Chi-square will not necessarily have a similar distribution to the collected data, should the mean service rate observed within the collected data be greater than one.

4.3.4 The Gamma Distribution

The gamma distribution seems to be more promising than the Erlang and Chi-square distributions, as it is often used to represent service and repair times in queuing models. The gamma distribution is a generalization of the Erlang which allows for noninteger shape parameters. Furthermore, a gamma random variable defined as Gamma(b, c) has an easy to calculate mean and standard deviation[1]:

$$\mu = bc$$

$$\sigma = \sqrt{b^2c}.$$

In addition it seems highly likely, as the gamma overcomes many of the short comings of the Erlang and Chi-square distributions, that one could design a unit gamma random variate with all the properties we have listed.

A unit gamma random variable X conforms to the specifications listed earlier, that it must have $E[X] = 1$, and that the random variates generated by it through inversion must have a similar

distribution when compared to the original observations from which $\hat{M}(t)$ was constructed. The parameterization of the gamma distribution presented here has a probability density function:

$$f(x) = \frac{x^{b-1}e^{-x/c}}{\Gamma(b)c^b}, \quad 0 \leq x < \infty$$

As we wish to have a mean of one, we can initially define the following properties of a unit gamma random variate:

$$bc = 1, \quad c = \frac{1}{b}$$

As such we can define our gamma random variate as a $\text{Gamma}(b, \frac{1}{b})$. When choosing a b with which to parameterize a unit gamma, we need to base our decision on the observations from which $\hat{M}(t)$ was constructed. Therefore we suggest trying to match the coefficient of variation such that:

$$\frac{S}{T} = \frac{\sigma}{\mu}$$

indicating that $b = \left(\frac{S}{T}\right)^2$. Unfortunately the gamma random variable features a mean directly tied to its shape and scale parameters, not unlike the Erlang and Chi-square random variables, meaning that the unit gamma often has a distinctly different distribution than the original service process which it was being used to approximate. Attempts at variate generation with a unit gamma random variate resulted in systems which failed to approximate the distribution of the realizations from which the estimator was constructed.

4.3.5 The Truncated Normal Distribution

While the normal distribution is often used to model the distribution of service times, in order to use the normal distribution for our purposes it must first be truncated to eliminate the negative tail.

If such a distribution were constructed, it would possess many of the attributes which have already been suggested as optimal for the estimation of a cumulative intensity function and the generation of random variates through the inversion of that cumulative intensity function. Most notably it possess an easily calculated mean (as the mean is one of the parameters), and a distribution shape which is independent of the mean. While more work is needed to develop a closed form solution of the truncated normal, a few of the characteristics of this distribution can easily be defined as a starting point for further work. Given the area to be truncated $p(\mu, \sigma)$, and the probability distribution function (pdf) of a normal random variable $f(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$ we can define the pdf of a truncated normal as,

$$g(x) = \frac{f(x)}{1 - p(\mu, \sigma)}, \quad x > 0$$

where $p(\mu, \sigma) = \int_{-\infty}^0 f(x)dx$. In order to adapt this variable for our suggested unit random variable we need to set the expected value, $E[X] = \int_0^{\infty} xg(x)dx$, to one. Furthermore, we can estimate the parameter σ by computing the coefficient of variation $\frac{S}{\bar{X}}$ from the realizations and solve for $\frac{\sqrt{V[X]}}{E[X]} = \frac{S}{\bar{X}}$ where $V[X] = \int_0^{\infty} x^2g(x)dx - E[X]^2$. Doing so should, in theory, provide a unit truncated normal with which to generate service times through inversion using our estimated cumulative intensity function.

4.4 Additional Areas

While the methods presented herein provide estimates for the given types of NHSP's, there is a need for further work investigating the fitting of these distributions to actual data, and the continued expansion of these methods to encompass other distributions. Ideally the methods discussed herein

could be expanded to cover all common service time distributions.

In addition there is a need for more rigorous estimates of the error introduced by each of the procedures outlined in Chapter 3 for the estimation of idle regions. This would further our ability to define error margins for the results obtained using these methods.

Finally it would be advantageous to formulate a method for the decomposition of the estimated cumulative intensity function into job-induced and node-induced nonstationary components. Such a decomposition would give experimenters the ability to alter the effects of the different types of nonstationary components in order to determine their overall effect on the system itself.

Chapter 5

Summary of Results

An extension of the methods given in [7] and [2] is presented that provides a nonparametric estimation of the cumulative intensity function for nonhomogeneous service processes exhibiting exponential service time distributions. Furthermore these estimates allow for multiple realizations, and provide methods for estimating regions for which the realizations do not overlap. Algorithms are presented for variate generation based on the inversion of the cumulative intensity function in a straightforward manner.

It should be noted that as our estimator is derived from realizations which could include a small number of observed service events, simulation via these techniques should be undertaken with an understanding of the actual system, as results are subject to sampling variability and may not be representative of the system on average. As the precision of our estimator $\hat{M}(t)$ increases with an increase in the total realizations taken into consideration, an easy way to obtain results more representative of the modeled system is to calculate the estimates with a large number of realized data.

Work on this project continues along the lines described in the previous chapter. The eventual goal being the solution of this problem for nonexponential cases, broadening the ability of those who use simulation to model nonstationary service processes in a variety of systems with which

they have contact.

Appendix A

Alternative Methods of Calculation

Throughout this study estimations for the service rate over a discontinuity are calculated by averaging the estimated service rates. It is important to note that another approach exists, which is to take the reciprocal of the average of the service times.

Let S_1 be the random service time that occurred before the discontinuity, and S_2 be the random service time that occurred just after the discontinuity. The first approach for estimating the service rate over the discontinuity is to average the estimated service rates over the two intervals:

$$M'_1(t) = \frac{1/S_1 + 1/S_2}{2} = \frac{S_1 + S_2}{2S_1S_2}.$$

A second approach for estimating the service rate over the discontinuity is to take the reciprocal of the average of the service times:

$$M'_2(t) = \frac{1}{(S_1 + S_2)/2} = \frac{2}{S_1 + S_2}.$$

The second approach gives a more conservative estimate of the service rate because, for $S_1 > 0$ and $S_2 > 0$:

$$\begin{aligned}
 M'_1(t) - M'_2(t) &= \frac{S_1 + S_2}{2S_1S_2} - \frac{2}{S_1 + S_2} \\
 &= \frac{(S_1 + S_2)^2 - 4S_1S_2}{2S_1S_2(S_1 + S_2)} \\
 &= \frac{(S_1 - S_2)^2}{2S_1S_2(S_1 + S_2)} \geq 0
 \end{aligned}$$

where equality is achieved when $S_1 = S_2$. Choosing $\hat{M}'_2(t)$ as the estimator yields an unbiased estimate for the expected service time during the discontinuity in the case of a nonhomogeneous Poisson process with an expected time between events of θ . For example:

$$E\left[\frac{S_1 + S_2}{2}\right] = \frac{\theta + \theta}{2} = \theta.$$

Appendix B

Source Code

B.1 Introduction

This chapter serves as a repository for the implementations of certain algorithms described in this text, and other useful implementations of constructs defined herein. Random variate generation was done using the Lehmer random number generator.[9].

B.2 determineIdle.cc

The program `determineIdle.cc` is an implementation in C++ of algorithm 3.1.

```

/*-----
 * determineIdle.cc
 *-----
 * Author : Eric William Davis Rozier
 * Contact: eric@5596.org
 *-----
 * Description:
 * This program reads data from standard input of the following format:
 * <total events> <start time> <end time>
 * <event type 1> <event time 1>
 * ...
 * <event type n> <event time n>
 *
 * And then calculates and outputs the idle regions, and outputs them to
 * them standard output. Comments are outputted with a leading #
 *-----
 * Event types
 *
 * 0 - Arrival, 1 - Departure
 *-----
 */

#include <iostream.h>

#define ARRIVAL 0
#define DEPARTURE 1

int size = 0;
int *eventType;
double *eventTime;

double clock = 0.0;
double stop = 0.0;
double lastDeparture = clock;
int idle = 0;
void dumpEventList() {
    for (int i = 0; i < size; i++) {
        if (eventType[i] == ARRIVAL) {
            cout << "Arrival ";
        } // if arrival
        else if (eventType[i] == DEPARTURE) {

```

```
        cout << "Departure";
    } // if departure
else {
    cout << "ERROR    ";
} // unknown event
cout << "\t" << eventTime[i] << endl;
    } // for each event
} // dumpEventList

int main(void) {
    int type;
    double time;
    cin >> size >> clock >> stop;
    eventType = new int[size];
    eventTime = new double[size];
    int i = 0;
    while (cin >> type >> time) {
eventType[i] = type;
eventTime[i] = time;
i++;
    } // while events to read in

    double currentTime = 0.0;
    double currentType = -1;
    int serverPop = 0;
    cout << "# Idle times for the event list\n";
    i = 0;
    if (eventTime[i] > clock) {
cout << "# Idle at start, left bound\n" << clock << endl;
idle = 1;
    } // if first event not at start
    for (i = 0; i < size; i++) {
currentTime = eventTime[i];
currentType = eventType[i];
if (currentType == ARRIVAL) {
    serverPop++;
    if (idle == 1) {
cout << "# Arrival, right bound\n" << currentTime << endl;
idle = 0;
    } // if idle
    } // if an arrival
else if (currentType == DEPARTURE) {
    serverPop--;
    lastDeparture = currentTime;
    if (serverPop == 0) {
```

```
cout << "# Departure, left bound\n" << currentTime << endl;
idle = 1;
    } // if system emptied
} // if a departure
    } // for each event
    if (eventTime[i] < stop) {
        if (serverPop > 0)
            cout << "# Server occupied at termination, left bound\n"
                << lastDeparture << endl;
cout << "# Idle at end, right bound\n" << stop << endl;
idle = 1;
    } // if first event not at start
    delete eventType;
    delete eventTime;
} // main
```

B.3 determineIdle-RF.cc

The program determineIdle-RF.cc is an implementation in C++ of algorithm 4.1.

```

/*-----
 * determineIdle-RF.cc
 *-----

 * Author : Eric William Davis Rozier
 * Contact: eric@5596.org
 *-----
 * Description:
 * This program reads data from standard input of the following format:
 * <total events> <start time> <end time>
 * <event type 1> <event time 1>
 * ...
 * <event type n> <event time n>
 *
 * And then calculates and outputs the idle regions, and outputs them to
 * them standard output. Comments are outputted with a leading #
 *-----
 * Event types
 *
 * 0 - Arrival, 1 - Departure, 2 - Failure, 3 - Repair
 *-----
 */

#include <iostream.h>

#define ARRIVAL 0
#define DEPARTURE 1
#define FAILURE 2
#define REPAIR 3

int size = 0;
int *eventType;
double *eventTime;

double clock = 0.0;
double stop = 0.0;
int idle = 0;
int fail = 0;
double lastDeparture = clock;
void dumpEventList() {
    for (int i = 0; i < size; i++) {

```

```

if (eventType[i] == ARRIVAL) {
    cout << "Arrival ";
} // if arrival
else if (eventType[i] == DEPARTURE) {
    cout << "Departure";
} // if departure
else if (eventType[i] == FAILURE) {
    cout << "Failure ";
} // if arrival
else if (eventType[i] == REPAIR) {
    cout << "Repair ";
} // if departure
else {
    cout << "ERROR ";
} // unknown event
cout << "\t" << eventTime[i] << endl;
} // for each event
} // dumpEventList

int main(void) {
    int type;
    double time;
    cin >> size >> clock >> stop;
    eventType = new int[size];
    eventTime = new double[size];
    int i = 0;
    while (cin >> type >> time) {
eventType[i] = type;
eventTime[i] = time;
i++;
    } // while events to read in

    double currentTime = 0.0;
    double currentType = -1;
    int serverPop = 0;
    cout << "# Idle times for the event list\n";
    i = 0;
    if (eventTime[i] > clock) {
cout << "# Idle at start, left bound\n" << clock << endl;
idle = 1;
    } // if first event not at start
    for (i = 0; i < size; i++) {
currentTime = eventTime[i];
currentType = eventType[i];
if (currentType == ARRIVAL) {

```

```

serverPop++;
if ((idle == 1) && (fail == 0)){
cout << "# Arrival, right bound\n" << currentTime << endl;
idle = 0;
} // if idle
} // if an arrival
else if (currentType == DEPARTURE) {
lastDeparture = currentTime;
serverPop--;
if (serverPop == 0) {
cout << "# Departure, left bound\n" << currentTime << endl;
idle = 1;
} // if system emptied
} // if a departure
else if (currentType == FAILURE) {
fail = 1;
if (idle == 0) {
cout << "# Failure, left bound\n" << currentTime << endl;
idle = 1;
} // if idle
} // if a failure
else if (currentType == REPAIR) {
fail = 0;
if (serverPop > 0) {
cout << "# Repair, right bound\n" << currentTime << endl;
idle = 0;
} // if the server has queue'd jobs
} // if a repair
} // for each event
if (eventTime[i] < stop) {
if (serverPop > 0)
cout << "# Server occupied at termination, left bound\n"
<< lastDeparture << endl;
cout << "# Idle at end, right bound\n" << stop << endl;
idle = 1;
} // if first event not at start
delete eventType;
delete eventTime;
} // main

```

B.4 Gamma Variate Generation

The following code implements a Gamma Random Variate.

```

/* -----
 * A Monte Carlo simulation to generate one-parameter gamma variates and
 * beta variates --- page 463-465 of Law and Kelton
 *
 * Name           : rvgs-2.c
 * Author          : Larry Leemis
 * Language        : C++
 * Latest Revision : 3-25-03
 * Compile with    : g++ rvgs-2.c rng.c rvgs.c
 *
 * -----
#include <iostream.h>
#include <math.h>
#include <stdlib.h>
#include "rvgs.h"
#include "rngs.h"

/* ===== */
double Gamma(double alpha)
/* ===== */
{
    int flag;
    double b, p, y, v, z, w, a, q, theta, d, x, u1, u2;
    if (alpha == 1.0)
    {
        u1 = Random();
        x = -log(1.0 - u1);
    }
    if (alpha < 1.0)
    {
        flag = 0;
        b = (exp(1) + alpha) / exp(1);
        while (flag == 0)
        {
            u1 = Random();
            p = b * u1;
            if (p > 1.0)
            {
                y = -log((b - p) / alpha);
            }
        }
    }
}

```

```
u2 = Random();
if (u2 <= pow(y, alpha - 1.0))
{
    x = y;
    flag = 1;
}
    }
    else
    {
        y = pow(p, 1.0 / alpha);
u2 = Random();
if (u2 <= exp(-y))
{
    x = y;
    flag = 1;
}
    }
}
if (alpha > 1.0)
{
    flag = 0;
    a = 1.0 / sqrt(2 * alpha - 1.0);
    b = alpha - log(4.0);
    q = alpha + 1.0 / a;
    theta = 4.5;
    d = 1.0 + log(theta);
    while (flag == 0)
    {
        u1 = Random();
        u2 = Random();
        v = a * log(u1 / (1.0 - u1));
        y = alpha * exp(v);
        z = u1 * u1 * u2;
        w = b + q * v - y;
        if (w + d - theta * z >= 0 || w >= log(z))
        {
            x = y;
flag = 1;
        }
    }
}
if (alpha < 0.0)
{
    cout << " error: negative gamma parameter " << endl;
```



```
    return 0;  
  }  
  else  
    return(x);  
}
```

B.5 parseEventList.pl

The following code implements the breaking down of active regions into pseudo-realizations.

```
#!/usr/bin/perl
#-----
# parseEventList.pl
#-----
# Author : Eric William Davis Rozier
# Contact: eric@5596.org
#-----
# Description:
# This program reads data from standard input of the following format:
# <id>
# <total events> <start time> <end time>
# <event type 1> <event time 1>
# ...
# <event type n> <event time n>
#
# It then writes out the pseudo realization decomposition of
# the event list, with estimates for the idle periods found there within
#
#-----
# Event types
#
# 0 - Arrival, 1 - Departure
#-----

$ARRIVAL = 0;
$DEPARTURE = 1;

$clock = 0.0;
$stop = 0.0;

$id = <STDIN>;
chop($id);
@data = <STDIN>;

foreach $i (@data) {
    if ($i == $data[0]) {
        $i =~ /([\S]*)\s*([\S]*)\s*([\S]*)/;
        $size = $1; $clock = $2, $stop = $3;
    } # if first element
    else {
        $i =~ /([\d]*)\s*([\d]*)/;
    }
}
```

```

$eventType[@eventType] = $1;
$eventTime[@eventTime] = $2;
    } # an event to process
} # for each event

$currentTime = 0.0;
$currentType = -1;
$serverPop = 0;
$idle = 0;
if ($eventTime[0] > $clock) {
    $previousActive[@previousActive] = -1;
    $idleLeftBound[@idleLeftBound] = $clock;
    $idle = 1;
} # idle at start
else {
    $activeLeftBound[@activeLeftBound] = $clock;
} # not idle at start
for ($i = 0; $i < $size; $i++) {
    $currentTime = $eventTime[$i];
    $currentType = $eventType[$i];
    if ($currentType == $ARRIVAL) {
$serverPop++;
if ($idle == 1) {
    $activeLeftBound[@activeLeftBound] = $currentTime;
    $nextActive[@nextActive] = @activeLeftBound - 1;
    $idleRightBound[@idleRightBound] = $currentTime;
    $idle = 0;
} # if idle
    } # if an arrival
    elseif ($currentType == $DEPARTURE) {
$departure[@activeLeftBound - 1] =
    "$departure[@activeLeftBound - 1] $currentTime";
$serverPop--;
if ($serverPop < 1) {
    $activeRightBound[@activeRightBound] = $currentTime;
    $previousActive[@previousActive] = @activeRightBound - 1;
    $idleLeftBound[@idleLeftBound] = $currentTime;
    $idle = 1;
} # if system emptied
    } # if a departure
} # for each event
if ($eventTime[$i] < $stop) {
    $activeRightBound[@activeRightBound] = $stop;
    $idleRightBound[@idleRightBound] = $stop;
    $nextActive[@nextActive] = -1;

```

```

    $idle = 1;
} # if first event not at start

open(FILE, ">idle-regions.$sid.idl");
for ($i = 0; $i < @idleLeftBound; $i++) {
    print FILE "$idleLeftBound[$i] $idleRightBound[$i]\n";
} # for each active interval
close(FILE);
`mv idle-regions.$sid.idl idle/.`;

for ($i = 0; $i < @activeLeftBound; $i++) {
    $file_name = "pseudo-realization.active.$i.$sid.rl";
    @times = split(" ", $departure[$i]);
    $lastEvent = $activeLeftBound[$i];
    open(FILE, ">$file_name");
    $count = @times;
    print FILE "$count\t$activeLeftBound[$i]\t$activeRightBound[$i]\n";
    for ($j = 0; $j < @times; $j++) {
        print FILE "1\t$times[$j]\n";
        $rate[$j] = 1 / ($times[$j] - $lastEvent);
        $lastEvent = $times[$j];
    } # for each event in the interval
    close(FILE);
    `mv $file_name data/.`;
} # for each active interval

```

Appendix C

Example Realizations

C.1 Realizations for Example 3.8.2

The following realizations were generated using a Lehmer random number generator[9]. Arrival times were generated using an NHPP via thinning. Service rates were generated as exponentials with a rate of 1.0. The simulation was halted at exactly 20.0 in each case.

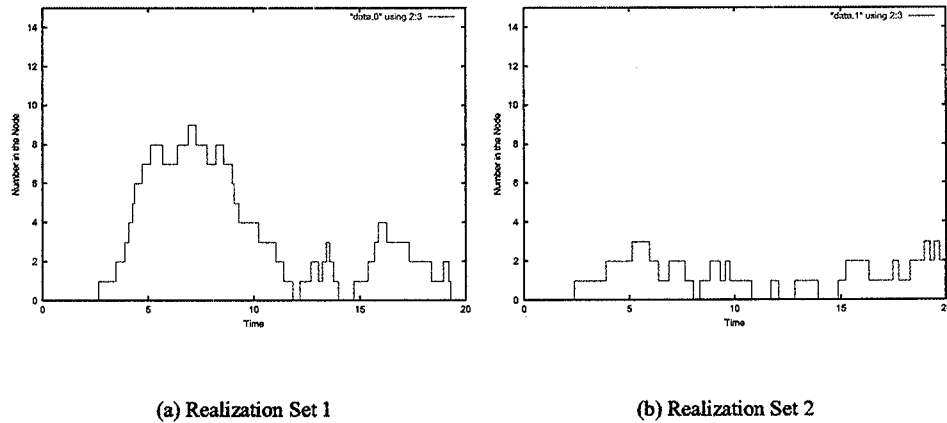
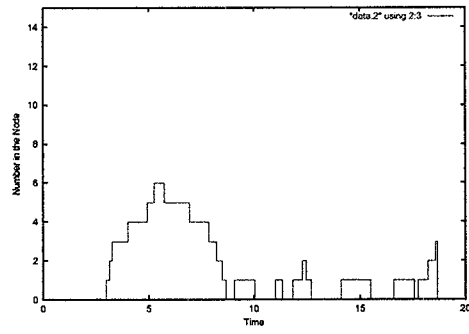
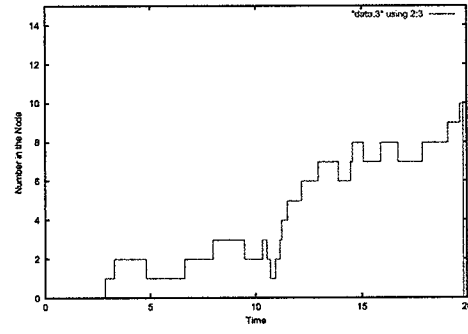


Figure C.1: Number in Node, Realizations Sets 1 and 2



(a) Realization Set 3



(b) Realization Set 4

Figure C.2: Number in Node, Realizations Sets 3 and 4

The following realizations were generated using a Lehmer random number generator[9]. Arrival times were generated using an NHPP via thinning. Service rates were generated as exponentials with a rate of 1.0 when $t < 10$, and 0.5 when $t \geq 10$. The simulation was halted at exactly 20.0 in each case.

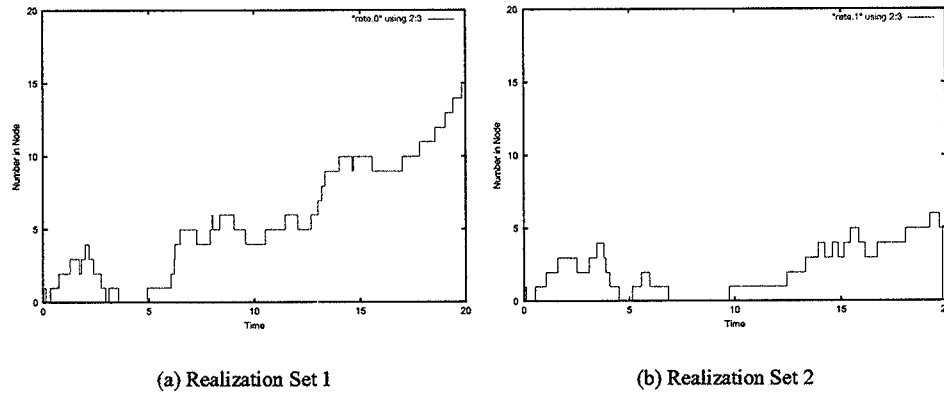


Figure C.3: Number in Node, Realizations Sets 1 and 2

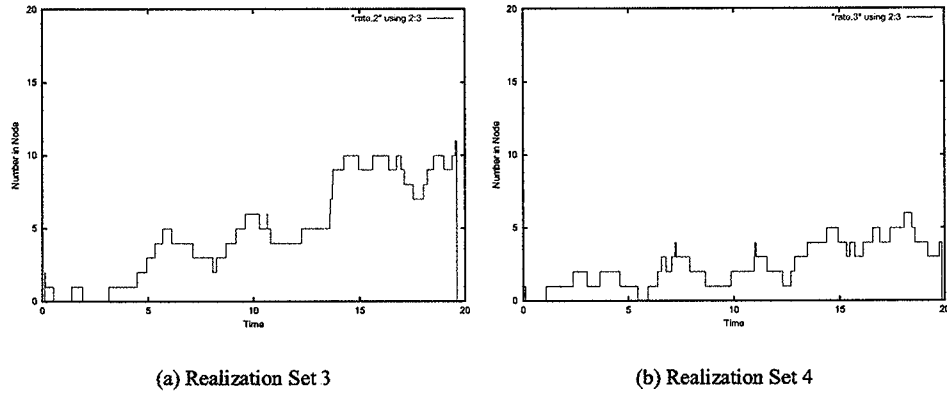


Figure C.4: Number in Node, Realizations Sets 3 and 4

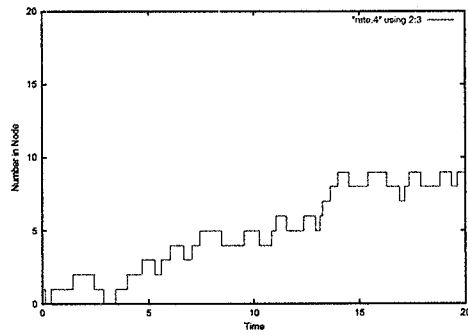


Figure C.5: Number in Node, Realization Set 5

Nonparametric Estimation of the Cumulative Intensity Function for a
Nonhomogeneous Service Process

ABSTRACT

A nonparametric technique for the estimation of the cumulative intensity function for a nonhomogeneous service process from one or more realizations that may contain idle periods is developed. This technique does not require any arbitrary parameters from the modeler. The estimated cumulative intensity function can be used for the generation of variates for simulation via inversion.

Eric William Davis Rozier

Department of Computer Science

The College of William and Mary in Virginia

Advisor: Dr. Lawrence Leemis

Nonparametric Estimation of the Cumulative Intensity Function for a Nonhomogeneous
Service Process

Bibliography

- [1] W. KELTON A. LAW. *Simulation Modeling and Analysis*. McGraw-Hill Science/Engineering/Math, McGraw-Hill Companies, PO Box 182605, Columbus, OH 43218-2605, 1979.
- [2] L. LEEMIS B. ARKIN. Nonparametric estimation of the cumulative intensity function for a nonhomogeneous poisson process from overlapping realizations. *Management Science*, 46, July 2000.
- [3] E. CINLAR. *Introduction to Stochastic Processes*. Prentice-Hall, Englewood Cliffs, NJ, 1975.
- [4] G. JENKINS G. BOX. *Time Series Analysis: Forecasting and Control*. Holden-Day, Inc., 4432 Telegraph Avenue, Oakland, CA, 1976.
- [5] R. JAIN. *The Art of Computer Systems Performance Analysis*. John Wiley and Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, 1991.
- [6] L. KLEINROCK. *Queueing Systems Volume II: Computer Applications*. John Wiley and Sons, Inc., 605 Third Avenue, New York, NY 10158-0012, 1979.
- [7] L. LEEMIS. Nonparametric estimation of the cumulative intensity function for a nonhomogeneous poisson process. *Management Science*, 37, July 1991.
- [8] S. ROSS. *Introduction to Probability Models*. Academic Press, 525 B Street, Suite 1900, San Diego, CA 92101-4495, 1997.
- [9] K. MILLER S. PARK. Random number generators: good ones are hard to find. *Communications of the ACM*, 31, October 1988.
- [10] L. LEEMIS S. PARK. *Discrete-Event Simulation: A First Course*. Revision January 2000, College of William and Mary, Williamsburg, VA 23187, 2000.