2016

Analysis and extension of a flexible stochastic differential equation model to estimate the amount of nanoconjugates required to initiate remission of cancer

Nicholas Joseph Skradski

ABSTRACT

ANALYSIS AND EXTENSION OF A FLEXIBLE STOCHASTIC DIFFERENTIAL EQUATION MODEL TO ESTIMATE THE AMOUNT OF NANOCONJUGATES REQUIRED TO INITIATE REMISSION OF CANCER

Nicholas Skradski, M.S.
Division of Statistics
Northern Illinois University, 2016
Nader Ebrahimi, Director

This thesis examines a definition of reliability that can be applied to targeted nanoconjugate chemotherapy in cancer and also simulates a stochastic differential equation model to attempt to predict the estimated time until remission when using targeted nanoconjugate chemotherapy utilizing a recursive algorithm to attempt to find the dose of nanoconjugates that will result in remission by an expected time *t*. Areas of research that can be used to obtain model parameter values are cited and values for the dose of nanoconjugates, given as number of nanoconjugates, are predicted for several different model parameters. The model is then extended to include a probabilistic framework that allows for a prior distribution to be placed on parameters where the exact value is not known but a distribution of the parameters may be known. Following that, examples of the extended model are given assuming that all of the model parameters are random. It is then suggested that the model undergoes evaluation with empirical data in order to examine the predictions of the model under certain parameters and some examples of further model extension are given that may allow for better predictions by relaxing some assumptions. Lastly the code for the program, written in R, is given in the appendix.

NORTHERN ILLINOIS UNIVERSITY
DE KALB, ILLINOIS

DECEMBER 2016

ANALYSIS AND EXTENSION OF A FLEXIBLE STOCHASTIC DIFFERENTIAL EQUATION MODEL TO ESTIMATE THE AMOUNT OF NANOCONJUGATES REQUIRED TO INITIATE REMISSION OF CANCER

BY

NICHOLAS JOSEPH SKRADSKI

A THESIS SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE
MASTER OF SCIENCE

DIVISION OF STATISTICS

Thesis Director:
Nader Ebrahimi

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1
INTRODUCTION

1.1 - Background

In the past several decades there has been a significant increase in the research into nanoparticles and their possible applications. These applications are numerous, ranging from use in electronics, biological experiments (De, Ghosh, & Rotello, 2008; Miranda, Creran, & Rotello, 2010), medicine (Biju, Mundayoor, Omkumar, Anas & Ishikawa, 2010; Lu, Doane, Zhu, & Burda, 2012), and many other fields. One specifically interesting area of study is the use of nanotechnology in biomedical applications. Nanomedicine, the approach of science and engineering at the nanometer scale toward biomedical applications, is the area of study in which biomedical applications and medicine intersect (Lee, Solanki, Kim, & Jung, 2009). The biomedical applications are also numerous, ranging from medical imaging (Jain, Huang, El-Sayed, & El-Sayed, 2008; Saha, Agasti, Kim, Li, & Rotello, 2012), photodynamic therapy (Celli et al., 2010; Mansoori, Brandenburg, & Shakeri-Zadeh, 2010), targeted drug delivery (Cai et al., 2016; Kalomiraki, Thermos, & Chaniotakis, 2016; Keyhanian, Mansoori & Rahimpour, 2010), bone grafting and regeneration (Murgan & Ramakrishna, 2007; Yang et al., 2016), gene therapy (Majidi et al., 2016), and multifunctional therapies (Lim et al., 2014). Nanoparticles, through their novel properties, can produce and improve therapeutics in a variety of different ways (Doane & Burda., 2012). These ways include improved bioavailability (Aditya et al., 2013) and improved toxicity to cancer cells, and, as mentioned above, nanoparticles can be used to provide medicine that can target specific areas of the body. The use of nanoparticles to target specific

areas of the body is achieved through several methods; these methods include taking advantage of pH differences in targeted tissues, upregulation or downregulation of receptors in the cell membranes in targeted tissues, differences in temperature in targeted tissues, and hydrophobicity changes in targeted tissues (Calixto, Bernegossi, Fonseca-Santos, & Chorilli, 2014; Doane & Burda, 2012). Nanoparticles give the opportunity to utilize more than one of these methods of targeting at the same time, possibly resulting in much more efficient and efficacious treatments. The next several paragraphs will provide examples of research that have utilized nanomedicine targeted treatment methods.

One method of targeted drug delivery utilizes the targeting methods of pH differences and targeting of upregulated folic receptors in cancer cells, demonstrating that nanomedicine is not limited to only one method of targeting tissue. For instance, Long et al. (2016) synthesized a micelle-based targeted treatment by loading cross-linked micelles with doxorubicin and conjugating folic acid to the fused polymers which would be reactive in lower pH solutions. The particles were then tested in vitro on mouse fibroblast cells and human cervical carcinoma cells. It was found that there was significantly more doxorubicin release at a pH of 5.8 (less than 20% released after 160 hours of incubation) than at a pH of 7.4 (over 50% release at an incubation time of 160 hours), where a pH of 5.8 is closer to what is exhibited in lysosomes and endosomes and a pH of 7.4 is what is exhibited in healthy human tissue generally. This suggests that doxorubicin release by this micelle would be lower when in non-tumor tissue, as tumor tissue often exhibits lower pH levels than healthy tissue. It was also found that the conjugation of folic acid to the polymer assisted in the uptake of the micelles into the cells, further demonstrating that this targeting mechanism showed benefits from both the pH targeting, as there would be less healthy tissue killed, and from folic acid conjugation, as there was more uptake when the micelle

was conjugated with folic acid. Overall, there is potential for this particular method of targeted treatment to deliver chemotherapies to cancer cells.

Another targeted treatment involved conjugating folic acid to PEGylated nanoparticles with quercetin and testing them on mice in vivo with HeLa cells and with IGROV-1 cells; both are types of ovarian cancer found in humans (El-Gogary et al., 2014). Both cell lines were also tested in vitro. It was found, through luminescence in vivo and testing with free folic acid and no free folic acid in vitro, that folic acid improved uptake of the nanoparticles. Furthermore, the conjugated quercetin showed as much cytotoxicity in the cancer cells as free quercetin, and release of quercetin was low in serum, which suggests that much of the quercetin would not be released until entering a cell or entering cancerous tissue. More research on the use of PEGylated folic acid nanoconjugates is needed to determine whether they are an effective carrier of chemotherapeutic agents, yet this study demonstrates potential for another effective targeted therapy.

These examples, though not anywhere close to exhaustive, serve as a foundation for one of the current problems that exists in nanotechnology research. That problem is the lack of any models to demonstrate the reliability of nanotechnology drug delivery systems. Currently it is not possible to tell, in an empirical manner, which nanoparticles should be pursued beyond the early stages of testing and which ones are superior drug delivery mechanisms. The next section will describe this problem in further detail and describe one suggested attempt to solve the problem. Chapter 2 will describe the method proposed in greater detail.

1.2 – Description of the Problem

There is a lack of consensus on what a reliable nanoparticle-based cancer treatment system is. This lack of consensus results in problems with designing, assessing, and

implementing nanoparticle-based cancer treatments. Recently, attempts to define reliability and establish ways to model it have been implemented (Ebrahimi & Mansoori, 2014; Ebrahimi, Mansoori, & Skradski, 2016). Specifically, Ebrahimi et al. (2016) suggest that a definition of reliability for cancer targeting nanoconjugate drugs over time should be a probability function interpreted as the probability of killing all cancer cells in a targeted tissue at some time t, where the higher the value of the function at time t, the more reliable the therapy is at time t. This definition of reliability has yet to be applied by researchers, but it does suggest that the faster a drug kills all of the cancer cells, the more reliable the drug is. Intuitively, this definition makes sense; however, there may be confusion because it differs from the definition of reliability generally used in survival analysis.

Barring possible semantic complaints about the definition of reliability in this context, modeling the probability of successfully treating cancer at certain points in time is a beneficial endeavor. To model the time taken to achieve death of all cancer cells, Ebrahimi et al. (2016) also proposed a stochastic differential equation model for the probability of killing all cancer cells in a certain amount of time. This model has many parameters and may benefit from a probabilistic framework.

In this thesis, the model proposed by Ebrahimi et al. (2016) will be examined at different values. There is no data to use to test this model's parameters, so examples will need to be used. It should be noted that the model proposed by Ebrahimi et al. (2016) is a time-dependent extension of the model proposed by Ebrahimi and Mansoori (2014).

Since it is possible to not know what one or more parameters is, a probabilistic framework will be incorporated into the model. Specifically, in situations where the parameters are unknown, it will be assumed that each of the unknown parameters will be a random quantity.

The outline of the thesis is as follows: In Chapter 2, the model proposed by Ebrahimi et al. (2016) is going to be introduced in detail. Assumptions, postulates, and examples of the model will be shown. The goal of Chapter 2 will be to test the model under different parameters in order to determine how the model should function under certain conditions. This will provide an estimate on how the nanoparticles should behave if the model is able to successfully make predictions. In Chapter 3, the model will be extended by making the assumption that the unknown parameters each follow certain distributions. This will then be tested in order to determine how the distributions affect the probability of killing cancer cells by some time t. Furthermore, examples will also be given in order to demonstrate how the model is expected to work under these conditions.

CHAPTER 2
PROPOSED PHARMACODYNAMIC MODEL

2.1 – Introduction to the Model

Table (2.1) lists the notation used, also given in by Ebrahimi et al. (2016), throughout this thesis. In table (2.1) nanoconjugate is defined as a molecule tailored from nanoparticles that has at least one biologically active module. A nanoparticle is simply defined as a particle with diameter between 1nm and 100nm.

The notation in table (2.1) and the criteria postulated by Ebrahimi et al (2016) results in the following definition of reliability in drug delivery utilizing nanomedicine:

$$\text{Let } T = Inf\{t : I_1(t) - kN(t) \geq 0\},\ P(t) = P(T \leq t).$$

This suggests that T is the time at which all cancer cells will be killed, a description of the functions mentioned and the parameter k is in the next section. A higher value of $P(t)$ implies that a drug delivery system is more reliable. Specifically, a drug delivery system A is more reliable than a drug delivery system B if $P_A(t) > P_B(t)$, where $P_A(t)$ and $P_B(t)$ are the reliability functions for system A and system B respectively. In general this definition does not necessitate that a particular drug delivery system is more reliable at some particular time t, or even for all t; it does, however, necessitate that the faster the drug delivery system kills all of the cancer cells, the more reliable it is. This should be sufficient for a basic model of the reliability of a drug delivery system.

Table 2.1. Notation Used in the Pharmacodynamic Model.

C	The number of nanoconjugates injected into the bloodstream containing the drug to be delivered
$S(t)$	The number of nanoconjugates that are at the site of administration at time t
$S_1(t)$	The number of nanoconjugates that are in non-targeted tissue at time t
$I_1(t)$	The number of nanoconjugates that are in cancerous cells at time t
$I_2(t)$	The number of nanoconjugates that are in healthy cells at time t
$R(t)$	The number of nanoconjugates that are removed from the body at time t
d_1	The average number of nanoconjugates that penetrate targeted tissues per unit time
d_2	The average number of nanoconjugates that penetrate healthy cells in the targeted tissue per unit time
d_3	The average number of nanoconjugates that penetrate non-targeted tissue per unit time
d_4	The average number of nanoconjugates that are eliminated from non-targeted tissues per unit time
d^*	$d_1 + d_2 + d_3$
γ_1	The average number of nanoconjugates that are eliminated from cancerous cells in the targeted tissue per unit time
γ_2	The average number of nanoconjugates that are eliminated from healthy cells in the targeted tissue per unit time
λ	The average number of cancerous cells that are killed per unit time
λ_1	The average number of cancerous cells proliferation per unit time
d_1^*	The average number of cancerous cells being eliminated per unit time ($\lambda - \lambda_1$)
k	Number of nanoconjugates carrying the drug needed to destroy a single cancerous cell in the targeted tissue
$N(t)$	The number of cancerous cells in the targeted tissue at time t
n	The number of cancerous cells in the tissue at time 0
T	The first time at which all cancerous cells are killed
$P(t)$	The reliability of the nanodrug system $(P(T \le t))$
$E(T)$	The expected time required to kill all cancer cells
$B_j(t)$	Standard Brownian motion, $j = 1,2,3,4,5$
$\sigma_j(t)$	Diffusion coefficient, $j = 1,2,3,4,5$
a_j	Intensity parameter, $j = 1,2,3,4,5$

2.2 – Proposed Pharmacodynamic Model

The model proposed by Ebrahimi et al. (2016) suggests that a pharmacodynamic model can be used to predict the reliability of the nanotechnology drug delivery system. This model is based on the number of nanoconjugates that are injected, the behavior of the nanoconjugates in the body, and the number of nanoconjugates, on average, required to kill a cancerous cell.

The model is a single compartment model, which is to say that the model proposes an all-encompassing model in which all tissues are treated as a single system. There is, however, a parameter to describe the rate at which nanoparticles enter targeted tissues and a parameter to describe the rate at which the nanoparticles enter non-targeted tissue.

The model, based on a first rate elimination process, is mathematically complex due to the use of a series of stochastic differential equations. Much of the derivation of the equations can be seen by reading Ebrahimi et al. (2016).

The model begins by assuming that the following four differential equations are used to model the elimination of nanoconjugates from the body, as given by Ebrahimi et al. (2016):

$$dS(t) = (-d_1 S(t) - d_2 S(t) - d_3 S(t))dt, \tag{2.1}$$

$$dS_1(t) = (d_3 S(t) - d_4 S_1(t))dt, \tag{2.2}$$

$$dI_i(t) = (d_i S(t) - \gamma_i I_i(t))dt, i = 1, 2, \tag{2.3}$$

$$dR(t) = (\gamma_1 I_1(t) + \gamma_2 I_2(t) + d_4 S_1(t))dt. \tag{2.4}$$

Utilizing basic differential equations concepts, and then setting $S(0) = C$ the following solutions can be found for Eqs. (2.1)-(2.4):

$$S(t) = S(0)\exp(-d^* t), \tag{2.5}$$

$$S_1(t) = \frac{Cd_3}{d^* - d_4}(\exp(-d_4 t) - exp(-d^* t)), \tag{2.6}$$

$$I_i(t) = \frac{Cd_i}{d^* - \gamma_i}(\exp(-\gamma_i t) - exp(-d^* t)), \tag{2.7}$$

$$\begin{aligned} R(t) = C[1 - \frac{d_1}{d^* - \gamma_1}\exp(-\gamma_1 t) - \frac{d_2}{d^* - \gamma_2}\exp(-\gamma_2 t) - \frac{d_3}{d^* - d_4}\exp(-d_4 t) - \\ (1 - \frac{d_1}{d^* - \gamma_1} - \frac{d_2}{d^* - \gamma_2} - \frac{d_3}{d^* - d_4})\exp(-d^* t)]. \end{aligned} \tag{2.8}$$

Since variation is always observed in effectiveness of medical treatments, in order to model the rates of the nanoconjugates being removed from the body, stochastic perturbation terms were introduced into Eqs. (2.1), (2.2), and (2.3). The resulting equations are as follows:

$$dS(t) = (d^* S(t))dt + \sigma_1(t)dB_1(t), \tag{2.9}$$

$$dS_1(t) = (d_3 S(t) - d_4 S_1(t))dt + \sigma_2 B_2(t), \tag{2.10}$$

$$dI_i(t) = (d_i S(t) - \gamma_i I_i(t))dt + \sigma_{i+2}(t)dB_{i+2}(t), i = 1, 2. \tag{2.11}$$

The solutions to Eqs. (2.9), (2.10), and (2.11) are as follows:

$$S(t) = C\exp(-d^* t) + \int_0^t \exp(-d^*(t-\mu))\sigma_1(\mu)d\mu \tag{2.12}$$

$$\begin{aligned} S_1(t) = \frac{Cd_3}{d^* - d_4}(\exp(-d_4 t) - exp(-d^* t)) + \\ \frac{d_3}{d^* - d_4}[\int_0^t exp(-d_4(t-\mu) - \exp(-d^*(t-\mu))\sigma_1(\mu)dB_1(\mu)] + \\ \int_0^t \exp(-d_4(t-\mu))\sigma_2(\mu)dB_2(\mu), \end{aligned} \tag{2.13}$$

$$I_i(t) = \frac{Cd_i}{d^* - \gamma_i}(\exp(-\gamma_i t) - exp(-d^* t)) +$$

$$\frac{d_i}{d^* - \gamma_i}[\int_0^t (exp(-\gamma_i(t-\mu)) - exp(-d^*(t-\mu))\sigma_1(\mu)dB_1(\mu)] + \tag{2.14}$$

$$\int_0^t exp(-\gamma_i(t-\mu))\sigma_{i+2}(\mu)dB_{i+2}(\mu) i = 1, 2.$$

Since $C = R(t) + S(t) + S_1(t) + I_1(t) + I_2(t),$ it is unnecessary to model R(t), as it can be found by modeling the other parameters. Also, the amount of nanoconjugates in non-targeted tissue is not important for this model, so $S_1(t)$ and $I_2(t)$ do not need to be modeled. Ebrahimi et al. (2016) described the following stochastic differential equation for $N(t)$ as:

$$dN(t) = -\lambda N(t) + \lambda_1 N(t) + \sigma_5(t)dB_5(t). \tag{2.15}$$

The solution for $N(t)$ derived utilizing stochastic calculus concepts is:

$$N(t) = n\exp(-d_1^* t) + \int_0^t \exp(-d_1^*(t-\mu))\sigma_5(\mu)d\mu, \tag{2.16}$$

$$Cov(N(t), N(t+s)) = \exp(-d_1^* s)\int_0^t \sigma_5^2(\mu)exp(-2d_1^*(t-\mu))d\mu. \tag{2.17}$$

If $i = 1,$ then the equation for $I_i(t)$ becomes:

$$I_1(t) = \frac{Cd_1}{d^* - \gamma_1}(\exp(-\gamma_1 t) - \exp(-d^* t))$$

$$+\frac{d_1}{d^* - \gamma_1}[\int_0^t (\exp(-\gamma_1(t-\mu)) - \exp(-d^*(t-\mu))\sigma_1(\mu)dB_1(\mu)] \tag{2.18}$$

$$+\int_0^t (\exp(-\gamma_1(t-\mu))\sigma_3(\mu)dB_3(\mu).$$

Throughout this chapter, for simplicity, it is assumed that $\sigma_j(t) = exp(-a_j t)\sigma_j,\ \ j = 1, 2, 3, 4, 5.$

This results in the following covariance for $I_1(t)$:

$$Cov(I_1(t), I_1(t+s)) = (\frac{\sigma_3^2}{2(a_3-\gamma_1)})\exp(-\gamma_1 s)(\exp(-2\gamma_1 t) - \exp(-2a_3 t)) + (\frac{d_1\sigma_1}{d^*-\gamma_1})^2[\frac{\exp(-2d^* t) - \exp(-2a_1 t)}{2(a_1-d^*)}\exp(-d^* s) + \frac{\exp(-2\gamma_1 t) - \exp(-2a_1 t)}{2(a_1-\gamma_1)}\exp(-\gamma_1 s) - (\exp(-d^* s) + \exp(-\gamma_1 s))\frac{\exp(-(d^*-\gamma_1)t) = exp(-2a_1 t)}{2a_1 - (d^*+\gamma_1)}]. \quad (2.19)$$

The assumption $\sigma_5(t) = exp(-a_5 t)\sigma_5$ reduces Eq. (2.9) to the following:

$$Cov(N(t), N(t+s)) = \sigma_5^2 \frac{\exp(-2d_1^* t) - exp(-2a_5 t)}{2(a_5 - d_1^*)}\exp(-d_1^* s). \quad (2.20)$$

These functions taken together suggest that $I_1(t) - kN(t)$ is a Gaussian process with mean

$$A(t) = \frac{Cd_1}{d^*-\gamma_1}(\exp(-\gamma_1 t) - \exp(-d^* t)) - kn\exp(-d_1^* t), \quad (2.21)$$

and covariance

$$B(t, t+s) = Cov(I_1(t), I_1(t+s)) + k^2 Cov(N(t), N(t+s)). \quad (2.22)$$

The estimated probability density function is then:

$$p(t) = g(t)(2\pi B(t,t))^{-1/2}\exp(\frac{A^2(t)}{2B(t,t)}), \quad (2.23)$$

where $g(t)$ is the following:

$$
\begin{aligned}
g(t) = -A(t)(B(t,t))^{-1}\Bigg\{ & k^2 \frac{\sigma_5^2}{2a_5 - 2d_1^*}[(2a_5 - d_1^*)\exp(-2a_5 t) - d_1^* \exp(-2d_1^* t)] + \\
& \frac{\sigma_3^2}{2a_3 - 2\gamma_1}[(2a_3 - \gamma_1)\exp(-2a_3 t) - \gamma_1 \exp(-2\gamma_1 t)] + \\
& \left(\frac{d_1\sigma_1}{d^* - \gamma_1}\right)^2 \frac{1}{2a_1 - 2d^*}[(2a_1 - d^*)\exp(-2a_1 t) - d^* \exp(-2d^* t)] + \\
& \left(\frac{d_1\sigma_1}{d^* - \gamma_1}\right)^2 \frac{1}{2a_1 - 2\gamma_1}[(2a_1 - \gamma_1)\exp(-2a_1 t) - \gamma_1 \exp(-2\gamma_1 t)] - \\
& \left(\frac{d_1\sigma_1}{d^* - \gamma_1}\right)^2 \frac{1}{2a_1 - (d^* + \gamma_1)}[(2a_1 - d^*)\exp(-2a_1 t) - \gamma_1 \exp(-(d^* + \gamma_1)t)] - \\
& \left(\frac{d_1\sigma_1}{d^* - \gamma_1}\right)^2 \frac{1}{2a_1 - (d^* + \gamma_1)}[(2a_1 - \gamma_1)\exp(-2a_1 t) - d^* \exp(-(d^* + \gamma_1)t)]\Bigg\} + \frac{d}{dt}(A(t)).
\end{aligned}
\tag{2.24}
$$

For further details regarding proofs, see Ebrahimi et al. (2016) and the sources referenced. With this, the probability that all of the cancer cells are killed at a certain time *t* can be obtained by integrating the probability density function:

$$
P(t) = \int_0^t g(x)(2\pi B(x,x))^{-1/2} \exp\left(\frac{-A^2(x)}{2B(x,x)}\right)dx \tag{2.25}
$$

The expected value of the function follows intuitively by integrating $\int_{-\infty}^{\infty} xf(x)I_{[0,\infty)}(x)dx$, where I($x$) is an indictor function, to obtain:

$$
E(T) = \int_0^{\infty} xg(x)(2\pi B(x,x))^{-1/2} \exp\left(\frac{-A^2(x)}{2B(x,x)}\right)dx. \tag{2.26}
$$

The integral is very complex and has no closed form. Thus, numerical integration techniques will have to be used.

With these equations it is possible to determine an algorithm to compute the expected number of nanoconjugates (C) needed to kill all cancer cells by some time *t*. Ebrahimi et al. (2016) suggest a simple iterative algorithm that adds or subtracts from C until it finds a C that has the closest expected value $E(t) \leq \alpha$ where α is chosen by the experimenter. Here an algorithm is presented that converges on the same number but obtains the required number faster, especially if the value of C required to kill cancer cells by time alpha is very far from the starting point. More specifically, the algorithm proposed adds some number *v* to the number of nanoconjugates if $E(T) > \alpha$ and subtracts some number, *v*, of nanoconjugates if $E(T) < \alpha$, where *v* is assumed to be a power of 10. After a number is found such that C-*v* and *C* satisfy $E(T) > \alpha$ and $E(T) < \alpha$ *v* is then divided by 10 and then the algorithm is run repeatedly until the algorithm produces $E(T) = \alpha$ and $v = 1$. Finally, the value for C is given when this is finished. The main idea is that this algorithm will be useful in predicting the estimated number of nanoconjugates, C, required to kill all cancer cells by time *t* by testing different values for C. The implementation of this algorithm in R is given in the appendix.

Due to of a lack of explicit research focusing on some of the parameters, there may currently be no justification for specific values of the parameters. However, some research does exist that may give some information about the parameters. For instance, there are studies on rats that found the relative amount of nanoconjugates distributed to different organs in vivo and ex vivo (El-Gogary et al., 2014; Patra et al., 2008, Tang et al., 2013). These studies are usually limited to testing postmortem, testing using fluorescence, or testing using ex vivo methods, but they may give some insight into the rates at which nanoconjugates are entering untargeted and targeted tissues. Furthermore, cancer proliferation and growth has been studied extensively

(Hanahan & Weinberg, 2000; Hannhan & Weinberg, 2011; Jennings, Winer-Muram, Tann, Ying, & Dowdeswell, 2006; Stein et al., 2010). The next few paragraphs will argue whether or not values for the parameters can be justified with the current research that is available.

The first parameter that may be justifiable with current research is the number of cancer cells. According to DeVita et al. (1975), the number of cancer cells in a 1-cm^3 tumor is about one billion cells. This number has been accepted by most medical researchers but has been questioned in editorials (Del Monte, 2009; Narod, 2012). Overall, since it is known that the number of cancer cells scales with tumor size, given the ability to measure average sizes of cancer cells, this parameter would be able to be found by estimating the size of the tumor with traditional imaging techniques and either accepting the statement by DeVita et al. (1975) as 1 billion cells per cubic centimeter or by analyzing average sizes of different cancers and using those sizes to predict the number of cells in cancerous tissue.

The second parameter value that may be able to be justified with current research is growth rate of the cancer being treated. There are studies that attempt to determine growth rates of different cancers. For instance, Stein et al. (2010) determined an average growth rate of pancreatic cancers of 0.6cm/year. Also Tseng et al. (2005) determined different growth rates of prostate cancers prior to treatment and during treatment. Utilizing similar studies to Stein et al. (2010) and Tseng et al. (2005), it is possible to calculate the growth rates to use in the model.

One objection to the constant growth rate assumption is different models of cancer growth result in different parameters, and these growth rates are likely not constant (Ulmschneider & Searson, 2015). For example, Norton (1988) utilizes a Gompertzian model to model the growth rates of breast cancer, which suggests that the rate of growth is not constant

over time. This problem can be alleviated in the model proposed by Ebrahimi et al. (2016) by using the fastest growth rate predicted by the models for the growth rate parameter.

The next parameter to examine is the rate at which cancer cells are killed. In vitro tests of nanoconjugates suggest that the rate of cancer cell elimination from targeted drug delivery systems that rely on chemotherapy conjugated with the nanoparticles can be slightly lower than from free chemotherapy (Long et al., 2016). However, there are some problems with modeling chemotherapy nanoconjugates that do not exist when modeling traditional chemotherapy treatments, such as the time of administration differing significantly from the time when the therapeutic effect of the nanoconjugate is realized. One solution to this problem is a model that was proposed by Eliaz et al. (2004) that explained the rate of cell elimination when doxorubicin was encapsulated in liposomes, and the model was tested in vitro on mouse melanoma cells that were known to overexpress CD44 receptors. Utilizing this model would give the necessary information to derive the cancer cell elimination rate for at least some nanoconjugates.

Though little research has been done which would allow for a direct computation of the rate parameters regarding entry into targeted tissues and non-targeted tissues, it may be possible to estimate it from fluorescence imaging done in vivo or in vitro. For example, El-Gogary et al. (2014) prepared a targeted drug delivery system with nanoconjugates allowing for fluorescence which targeted folate-expressing cancer cells. The results, if fluorescence is taken to represent the amount of nanoconjugates that are inside of the cells of certain body areas, suggest that targeting the tumor is rather difficult, with much of the drug entering the lungs and the tumor and to a lesser extent the spleen, kidneys, liver, and heart. There also have been other formulations of nanoconjugates which enter the targeted tissues far more often, but still less often than other organs. For example, Comenge et al. (2012) conjugated cisplatin with gold nanoparticles.

Accumulation was found in the liver, spleen, tumor, and to a much lesser extent, other organs. This would suggest that in some cases nanoparticles can target other tissues more often than cancer cells, thus resulting in $d_3 > d_1$. Nanoconjugate formulations that result in a higher uptake by cancer cells than by healthy cells exist as well. For example, Saltan et al. (2011) created a nanoparticle conjugated with methacrylamido-folic acid that was absorbed by cancer cells at a higher rate than by healthy cells according to flow cytometic analysis and observations with transmission electron microscopy. These taken together suggest that there are situations in which cancer cells uptake nanoparticles at higher rates and at lower rates than healthy cells and that there are methods to determine the rate of nanoconjugate uptake by healthy and cancerous tissues.

What follows from the previous paragraph would be an examination of the rate at which nanoconjugates are removed from cancerous tissue. There is research that suggests that some nanoconjugates may not clear from the body very well in vivo (Longmire, Choyke, & Kobayashi, 2008). Furthermore, the enhanced permeability and retention effect, which can be taken advantage of with nanomedicine, would increase the amount of time nanoconjugates remain in targeted tissue (Lyer, Khaled, Fang, & Maeda, 2006). Since the enhanced retention and permeability effect does not present itself in healthy tissue very often, it would follow that nanoconjugates would filter from healthy tissue faster than cancerous tissue. An example of a study examining the pharmacokinetic properties of cisplatin and a pectin-cisplatin nanoconjugate system showed a striking difference between cisplatin and the conjugated cisplatin (Verma & Sachin, 2008). Specifically, the pectin-cisplatin conjugate had an elimination rate constant of 0.0042 and a half-life of 165 minutes, as opposed to cisplatin alone, which had an elimination rate constant of 0.0172 with a half-life of 40 minutes, though these numbers were obtained

without a tumor in the in vivo test. Taken together, this information suggests that the expected parameter for elimination from cancer cells will be very small, as will the parameter for the elimination rate from healthy tissue, though the enhanced permeability and retention effect existing in cancerous tissue more often than healthy tissue would imply that the rate of elimination from cancerous tissue would be lower than that of healthy tissue.

Another parameter included in the model, the average number of nanoconjugates required to kill a cancerous cell, is not known for sure, but it is known to scale with the size of the cancer cell in some cases (Hashemian, & Mansoori, 2013; Keyhanian, Mansoori, & Rahimpour, 2010; Mansoori, Brandenburg, & Shakeri-Zadeh, 2010; Zharov, Lutfullin, & Galitovskaya, 2005). This would suggest, though not definitively, that the nanoparticle amount required to kill a cancer cell should be a function of the cancer cell size.

Overall, it appears to be the case that there is research to draw from which would determine possible values for the model parameters. Without explicit research to estimate these parameters, however, it will be difficult to obtain precise values for the parameters.

2.3 – Examples of the Pharmacodynamic Model

The first example here will be a simple example focusing on a very small number of cancerous cells. Assume that some treatment has already been conducted with traditional chemotherapy and radiation for some unspecified type of cancer. Assume that there are 210 million cancer cells remaining in the tissue. For numerical stability, the number of cancer cells and nanoconjugate parameters will be in millions of cells and millions of nanoconjugates. Further suppose that the targeted drug delivery system has been perfected to such an extent that nanoconjugates enter the cancerous cells in the targeted tissue with a rate parameter of 1.2 million nanoconjugates per unit time, nanoconjugates enter healthy cells in the targeted tissue

with a rate parameter of .1 million nanoconjugates per unit time, and nanoconjugates enter non-targeted tissue with a rate parameter of 0.1 million nanoconjugates per unit time. Furthermore, the amount of nanoconjugates that are eliminated from cancerous cells per unit time is equal to the amount that enter, and the amount of nanoconjugates that are removed from healthy cells per unit time equals the number of nanoconjugates that enter healthy cells per unit time. Assume that the average number of cancerous cells that are killed per unit time is equal to the rate at which nanoconjugates enter cancer cells per unit time minus .39. Also, assume a cancer growth rate parameter of .2 and diffusion coefficient parameters of $\sigma_1 = .5, \sigma_3 = .61, \sigma_5 = .44$ with intensity parameters $a_1 = 0.142, a_3 = 0.0144, a_5 = 0.177.$ Assume that the cancer cells are fairly large and only require 15 million nanoconjugates to kill. Using this algorithm it was found that 5223 million nanoconjugates would be required to eliminate the rest of the cancer cells within a single time unit. The expected nanoconjugates would be required to eliminate the rest of the cancer cells within half of a time unit would be 7408 million nanoconjugates. Table (2.2) shows the amount of nanoconjugates required to kill all cancer cells by an expected time *t* at *t*=1 and *t*=0.5 at different values of *k* obtained by using the algorithm.

Table 2.2 Expected number of nanoconjugates in millions for example 1 at expected time until all cancer cells are killed for E(*t*) = 1 and E(*t*) = .5 according to the algorithm.

Value of K	Expected number of nanoconjugates for E(*t*)=1	Expected number of nanoconjugates for E(*t*)=.5
5	1743	2468
10	3482	4941
15	5223	7408
20	6965	9878
25	8707	12348
30	10451	14821

Again, it is very unlikely that the parameters in this example are accurate or realistic in vivo. Generally, if cancer is detectable, there would be at least one cubic centimeter of cancerous tissue, and with the assumption that the cancers are 10 μm in diameter there would be about one billion cancer cells in the cancerous tissue. Since in this example there were only 210 million cancer cells, unless the cells were larger, this cancer would likely be undetectable. Furthermore, the diffusion and rate parameters in this example are not necessarily accurate; in practice it is assumed that these parameters are simply 1 and 0.1 respectively. These problems could be responded to by suggesting that this treatment is given after traditional chemotherapy and radiation.

The next example will be far more likely than the previous, at least given the current information regarding nanomedicine. Assume that a tumor was discovered early and contains one billion cancerous cells. This would suggest the minimum size is one cubic centimeter. Furthermore, assume that $a_1 = .12, a_3 = .11, a_5 = .2, \sigma_1 = .7, \sigma_3 = .7, \sigma_5 = .8, \gamma_1 = 1.2, \gamma_2 = 1.1,$ $d_1 = 0.9, d_2 = .1, d_3 = 0.9.$ This would suggest that targeted tissues are entered as often as non-targeted tissue. Assume $\lambda = .81, \lambda_1 = .2.$ This example is a pessimistic scenario since the rates of targeted and untargeted tissue being entered by nanoconjugates are the same. There are still problems, however, as the rate at which nanoconjugates are removed from the tissues is very high. Table (2.3) provides information regarding how many nanoconjugates would be expected to kill cancerous cells by t=1 and t=0.5, which were obtained using the algorithm.

Table 2.3. Expected number of nanoconjugates in millions for example 2 at expected time until all cancer cells are killed for E(*t*) = 1 and E(*t*) = 0.5 according to the algorithm.

Value of K	Expected number of nanoconjugates for E(*t*)=1	Expected number of nanoconjugates for E(*t*)=.5
5	11747	16321
10	23493	32644
15	35235	48966
20	46981	65289
25	58726	81612
30	70472	97934

The final example will use the same parameters as the previous example; however, the rate at which nanoconjugates are removed from cancerous and healthy tissues will be much lower. Assume that $\gamma_1 = 0.1, \gamma_2 = 0.2$. This appears to be a more realistic representation of nanoconjugates behavior in vivo since nanoparticles clear from the body slowly and the EPR effect should reduce the rate at which nanoparticles are removed from cancerous tissue (Lyer et al., 2006). If this model was realistic, there would be an improvement from a decrease in estimated time until all cancer cells are destroyed because of the increase in treatment efficacy due to the EPR effect. This behavior is seen with these examples. Table (2.4) shows this to be the case. The drop in the rate parameters of nanoconjugate removal from tissues appears to have a very large effect on the rate of cancer treatment.

Table 2.4. Expected number of nanoconjugates in millions for example 3 at expected time until all cancer cells are killed for E(*t*) = 1 and E(*t*) = 0.5 according to the algorithm.

Value of K	Expected number of nanoconjugates for E(*t*)=1	Expected number of nanoconjugates for E(*t*)=0.5
5	7193	13057
10	14387	26115
15	21581	39173
20	28775	52232
25	35969	65290
30	43163	78348

CHAPTER 3
EXTENSION OF THE PHARMACODYNAMIC MODEL

3.1 – Introduction and Explanation of the Extension Model

In the previous chapter the model parameters were assumed to be fixed. There may be benefits to treating the parameters as random since many of them are not known specifically and current research into nanotechnology is often insufficient to estimate these parameters precisely. Thus, an estimate based on averaging the parameters over the distribution will be used. If the parameters of the model are treated as random, they will each follow a distribution which must be provided in order to model P(t). Under the assumption of independence of every parameter, if every parameter is treated as random, the joint distribution becomes the following:

$$\int_0^t P(T \le t) f_{\Theta}(\Theta) db =$$

$$\int_0^t \int_{\Theta} P(Inf\{t : I_1(t) - kN(t) \ge 0\} \le t \mid d_1, d_2, d_3, \sigma_1, \sigma_3, \sigma_5, a_1, a_3, a_5, d_1^*, n, k) * \quad (2.27)$$
$$f_{d_1}(d_1) f_{d_2}(d_2) f_{d_3}(d_3) f_{\sigma_1}(\sigma_1) f_{\sigma_3}(\sigma_3) f_{\sigma_5}(\sigma_5) f_{a_1}(a_1) f_{a_3}(a_3) *$$
$$f_{a_5}(a_5) f_{d_1^*}(d_1^*) f_n(n) f_k(k) d\Theta dt,$$

where $\int_{\Theta} ... d\Theta$ represents integrating all parameters of the integral iteratively as well as the summation of any variables assumed to be discrete. In practice, it may be the case that some of the parameters are known, in which case there is no need to assign them a distribution, which would be obtained from experts in the field or derived from studies into the model parameters. For the following examples, it is assumed that all of the parameters are unknown and that all of the continuous parameters $(d_1, d_2, d_3, \sigma_1, \sigma_3, \sigma_5, a_1, a_3, a_5)$ follow an exponential distribution with

parameters $(v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9)$ respectively. All of the discrete parameters (k, n) follow a Poisson distribution with parameters (v_{11}, v_{12}) respectively. Since d_1^* is the difference between two exponential random variables, it will follow a Laplace distribution truncated at 0 with parameters $(v_{10,1}, v_{10,2})$. This satisfies the assumption that the rate of cancer cells being killed is greater than or equal to the rate that cancer cells are proliferating. The following distribution is then given when these assumptions are combined with Eq. (3.1):

$$P(t) = \int_0^t \int_\Theta \sum_{k=0}^{\infty} \sum_{n=0}^{\infty} g(x)(2\pi B(x,x))^{-1/2} \exp\left(\frac{-A^2(x,x)}{2B(x,x)}\right) * \\ v_1 e^{-v_1 d_1} v_2 e^{-v_2 d_2} v_3 e^{-v_3 d_3} v_4 e^{-v_4 \sigma_1} v_5 e^{-v_5 \sigma_3} v_6 e^{-v_6 \sigma_5} v_7 e^{-v_7 a_1} * \\ v_8 e^{-v_8 a_3} v_9 e^{-v_9 a_5} v_{10,1} v_{10,2} e^{(v_{10,1}\lambda - v_{10,2}\lambda_1)} \frac{{v_{11}}^k e^{-v_{11}}}{k!} \frac{{v_{12}}^n e^{-v_{12}}}{n!} d\Theta dx. \tag{2.28}$$

This function is 12 dimensions prior to integrating over *t*, and thus it is extremely computationally intensive to estimate. In order to compute the integrals, as estimations of the integral are extremely difficult, Monte Carlo simulation was employed. Fifty thousand randomly selected values from the prior of each variable were obtained and integration over *t* was attempted with them. If integration failed, resampling was done. The variables were resampled until the assumptions of the model were met, that is, $d_1^* > 0, \gamma_1 \le d^*, \gamma_2 \le d^*$. This was done because the integral would not converge if these assumptions were not met.

3.2 – Examples of the Extension Model

The first model that was tested was example 1 as seen in Chapter 2. It was assumed that the parameters followed exponential distributions, the Laplace distribution, and Poisson distributions with parameters equal to the fixed parameters in example l in Chapter 2. The specific values for the parameters above are $v_1 = 1.2, v_2 = 0.1, v_3 = 0.1, v_4 = 0.5, v_5 = 0.61,$

$v_6 = 0.44, v_7 = 0.142, v_8 = 0.0144, v_9 = 0.177, v_{10,1} = 0.81, v_{10,2} = 0.2, v_{11} = 5,$ and $v_{12} = 210$. The number of nanoconjugates for E(T) = 1 in example 1 with $k = 5$ was 1743 million nanoconjugates. With the new added uncertainty E(T) = 1.1013. This suggests that there will be an increase in the estimated amount of nanoconjugates required in order to kill all cancer cells within a single time unit. Table (3.1) shows the expected amount of nanoconjugates in increments of 50 starting near 1750. Figure (3.1) shows a graph of the expected time to kill all cancer cells for values of C between 1743 and 2050. What can be seen is that there is an increase in the expected time until all cancer cells are eliminated at C=1743; as in example 1, Chapter 2, C=1743 resulted in an expected time of elimination of all cancer cells of E(T)=1. In the extended model, C=1743 results in an expected time to kill all cancer cells of E(T)=1.1012596 time units. At C=2050, the expected time to kill all cancer cells was E(T)=0.7639282 time units.

Table 3.1. Expected time until all cancer cells are killed based on example 1 using the averaging method with v_{12}=5.

Number of nanoconjugates in millions (Value of C)	E(T)
1743	1.1012596
1800	1.0285896
1850	0.9602398
1900	0.8915131
1950	0.8395372
2000	0.8028654
2050	0.7639282

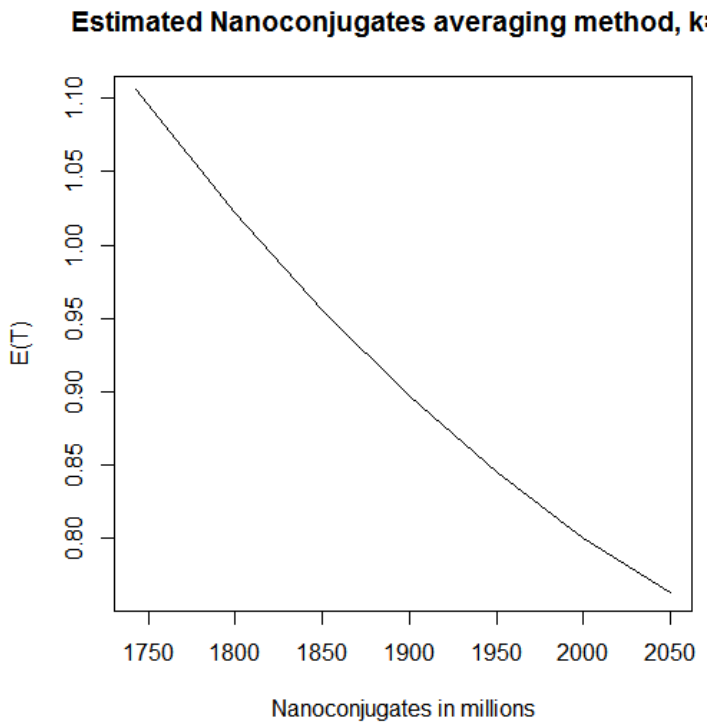

Figure 3.1. Graph of the estimated time until all cancer cells are killed at differing values of C based on parameter values from Chapter 2, example 1.

A second example of the extended model will be shown applying the parameters used previously in Chapter 2, example 3 with $k = 5$. A similar pattern to example 1 was seen in the expected time for nanoconjugates to kill all cancer cells. The expected number of nanoconjugates required for $E(T) = 1$ in Chapter 2, example 3 was C=7193. The expected time at C=7193 nanoconjugates in the extended model, however, was E(T)=1.698633 time units. This increase suggests that larger uncertainty results in a larger expected minimum number of nanoconjugates to kill all of the cancer cells by some time *t*. Table (3.2) shows the expected time until all cancer cells are killed computed using Monte Carlo integration of the unknown

parameters. Figure (3.2) shows a graph of the estimated time to kill all of the cancer cells given a certain value for C.

Table 3.2. Expected time until all cancer cells are killed based on Chapter 2, example 3, using the averaging method with v_{12}=5.

Number of nanoconjugates in millions (Value of C)	E(T)
7000	1.698633
8000	1.578399
9000	1.461240
10000	1.372515
11000	1.275146
12000	1.204018
13000	1.169193
14000	1.117799

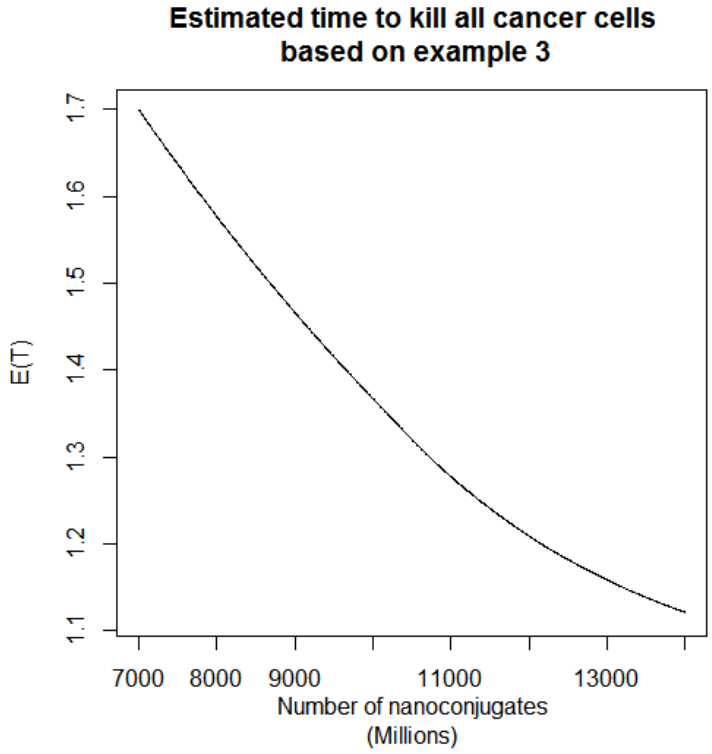

Figure 3.2. Graph of the estimated time required to kill all cancer cells for differing values of C based on parameter values from Chapter 2, example 3.

CHAPTER 4
CONCLUSION AND FUTURE RESEARCH

The model does make some predictions that are in line with the empirical data available. For instance, the decrease in E(T) when γ_1 that occurs would be expected due to the EPR effect. Furthermore, it is easily shown that E(T) decreases in C, which should be the case as an increased dose of anti-cancer drugs results in higher elimination of cancer cells. Also, there is research that can be used in order to determine ranges for the parameter values of the model.

Though there are ways to estimate the parameters, currently conclusions are difficult to draw. Since there is no empirical data to draw upon, it cannot be determined whether or not these models are useful, or how the models even behave in practice. Further research is required in order to determine whether or not these models make useful predictions. If predictions are able to be made, these models will suffice until better models can be derived that provide better predictions. If the models fail to provide successful predictions, they ought to be discarded and new models must be formulated in order to make predictions that better match reality. Also, the model itself is based on several assumptions that may not hold and there may be ways to improve the model.

First, it is unlikely that the growth rate for cancer is constant. Some research shows that cancer growth rates follow a Gompertz model in practice (Norton, 1988; Ulmschneider & Searson, 2015). This problem could be alleviated with the current model by taking the fastest possible growth rate established by the Gompertz model, but it would result in overestimation of the rate of cancer growth overall and may cause a useful drug to be deemed not useful under

certain circumstances, such as a drug that could be used to induce remission prior to a cancer reaching a certain growth rate. Thus another extension of this model could be done which takes a non-constant growth rate into consideration.

Second, the model assumes that the nanodrug will kill all of the cancer cells. If all of the cancer cells are not killed, E(T) will be infinity. This may not be practical because it could be the case that, at least at the current moment in time, nanodrugs will provide life extension by slowing the growth rate of the cancer, as is the case with current chemotherapy treatments in some cases. Thus it appears that this model will only suffice for cases where the treatment of the cancer may result in remission. It may, however, be the case that nanoparticles can increase the number of cancers that are able to be put into remission, and as such this model may be useful when testing nanoconjugates in particular. More research on the applications of nanotechnology in cancer therapies must be carried out in order to know which future medications this model will be applicable to.

Third, the model assumes that a single-compartment pharmacokinetic model is sufficient. Depending on how nanoconjugates behave, this may not be the case. There appears to be a saturation level (i.e. a level where areas of the body will not uptake any more nanoparticles) and this saturation level may result in a multi-compartmental model being necessary because it would likely cause more nanoparticles to remain in the bloodstream. Evidence of this can be seen in Thurn et al. (2011), where the rate of uptake of the nanoconjugate used followed a logarithmic pattern.

The current proposed model also uses an approximation for the density (Durbin, 1985). This approximation may need to be changed because of what appears to be tail behavior exhibited in this analysis of the model proposed by Ebrahimi et al. (2016). In order to

circumvent this, the distribution was often truncated and full integration of the distribution may have resulted in a probability of less than or greater than 1. Generally, unless very high values of C were used, P(T) was very close to 1. In order to correct for P(T) being unequal to 1, the expected time until all nanoconjugates were killed was divided by the integral of the density over the truncated interval. It is unknown whether or not this method would be acceptable when using this model in practice. Further study of the model is warranted as it appears to be the case that the approximation from Durbin (1985) may fail in some specific cases.

Last, the stochastic differential equations are assumed to be taken in respect to a standard Brownian motion. Relaxing this assumption may offer an extension of this model that could allow for better prediction and estimation. A jump process might be the next step for extension of this model in terms of stochastic calculus.

Overall, the only way to validate this model is to test it empirically: Treat the model as a collection of hypotheses, and if the model makes accurate predictions, some of the hypotheses are correct. Even if the predictions are accurate, there may exist better methods for deriving models for nanotechnology reliability that have not yet been explored. As such, this model needs to be first tested against empirical data. If the model is shown to be useful to make predictions about the reliability of targeted drug delivery, then some of the model's assumptions should be relaxed in order to see if better predictions are made, or a better model needs to be derived using the newly available data. It appears that what must be done regardless of the performance of the model is further research into reliability of targeted drug delivery utilizing nanotechnology.

REFERENCES

Aditya, N. P., Shim, M., Lee, I., Lee, Y., Im, M. H., & Ko, S. (2013). Curcumin and genistein coladed nanostructured libid carriers: In vitro digestion and antiprostate cancer activity. *Journal of Agricultural and Food Chemistry*. 61, 1878-1883. http://dx.doi.org/10.1021/jf305143k.

Biju, V., Mundayoor, S., Omkumar, R. V., Anas, A., & Ishikawa, M. (2010). Bioconjugated quantum dots for cancer research: Present status, prospects and remaining issues. *Biotechnology Advances*. 28, 199-213. http://dx.doi.org/10.1016/j.biotechadv.2009.11.007.

Cai, Y., Zhang, W., Chen, Z., Shi, Z., He, C., Chen, M. (2016). Recent insights into the biological activities and drug delivery systems of tanshinones. *International Journal of Nanomedicine*. 11, 121-130. http://dx.doi.org/10.2147/IJN.S84035.

Calixto, G., Bernegossi, J., Fonseca-Santos, B., Chorilli, M. (2014). Nanotechnology-based drug delivery systems for treatment of oral cancer: a review. *International Journal of Nanomedicine*. 9, 3719-3735.

Celli, J. P., Spring, B. Q., Rizvi, I., Evans, C. L., Samkoe, K. S., Verma, S., . . . Hasan, T. (2010). Imaging and photodynamic therapy: Mechanisms, monitoring, and optimization. *Chemical Reviews*. 110, 2795-2838. http://dx.doi.org/10.1021/cr900300p.

Comenge, J., Sotelo, C., Romero, F., Gallego, O., Barnadas, A., Parada, T. G., . . . Puntes, V. F. (2012). Detoxifying antitumoral drugs via nanoconjugation: The case of gold nanoparticles and cisplatin. *PLOS One*. 7(10), e47562. http://dx.doi.org/10.1371/journal.pone.0047562.

De, M., Ghosh, P. S., & Rotello, V. M. (2008). Applications of nanoparticles in biology. *Advanced Materials* 20, 4225-4241. http://dx.doi.org/10.1002/adma.200703183.

Del Monte, U. (2009). Does the cell number 10(9) still really fit one gram of tumor tissue?. *Cell Cycle*. 8(3), 505-506. http://dx.doi.org/10.4161/cc.8.3.7608

DeVita, V. T., Young, R. C., & Canellos, G. P. (1975). Combination versus single agent chemotherapy: A review of the basis for selection of drug treatment in cancer. *Cancer*. 35, 98-110.

Doane, T. L., & Burda, C. (2012). The unique role of nanoparticles in nanomedicine: Imaging, drug delivery and therapy. *Chemical Society Reviews*. 41, 2885-2911.

Durbin, J. (1985). The first-passage density of a continuous Gaussian process to a general boundary. *Journal of Applied Probability*. 22(1), 99-122.

Ebrahimi, N., & Mansoori, G. A. (2014). Reliability for drug targeting in cancer treatment through nanotechnology (A psychometric approach). *International Journal of Medical Nano Research*. 1, 5pp.

Ebrahimi, N., Mansoori, G. A., & Skradski, N. (2016). Reliability for drug targeting in cancer treatment through nanotechnology (A Stochastic differential equation-based flexible model). *Frontiers in Nanoscience and Nanotechnology*. 2(4), 144-148. http://dx.doi.org/10.15761/FNN.1000125.

Eliaz, R. E., Nir, S., Marty, C., Szoka, F. C. (2004). Determination and modeling of kinetics of cancer cell killing by doxorubicin and doxorubicin encapsulated in targeted liposomes. *Cancer Research*. 64, 711-718.

El-Gogary, R. I., Rubio, N., Wang, J. T., Al-Jamal, W. T., Bourgognon, M., Kafa, H., . . . Al-Jamal, K. T. (2014). Polyethylene glycol conjugated polymeric nanocapsules for targeted delivery of quercetin to folate-expressing cancer cells in vitro and in vivo. *ACSNano*. 8(2) 1384-1401.

Hanahan, D., & Weinberg, R. A. (2000). The hallmarks of cancer. *Cell*. 100, 57-70. http://dx.doi.org/10.1016/S0092-8674(00)81683-9.

Hannhan, D., & Weinberg, R. A. (2011). The hallmarks of cancer: The next generation. *Cell*. 144, 646-674. http://dx.doi.org/10.1016/j.cell.2011.02.013.

Hashemian, A. R., & Mansoori, G. A. (2013). Cancer nanodiagnostics and nanotherapeutics through the folate conjugated nanoparticles. *Journal of Bioanalysis and Biomedicine*. 5, 61-64. http://dx.doi.org/10.4172/1948-593X.1000080.

Jain, P. K., Huang, X., El-Sayed, I. H., & El-Sayed M. A. (2008). Noble metals on the nanoscale: Optical and photothermal properties and some applications in imaging, sensing, biology, and medicine. *Accounts of Chemical Research*. 41(12), 1578-1586. http://dx.doi.org/10.1021/ar7002804.

Jennings, S. G., Winer-Muram, H. T., Tann, M., Ying, J., & Dowdeswell, I. (2006). Distribution of stage I lung cancer growth rates determined with serial volumetric CT measurements. *Radiology*. 241(2), 554-563. http://dx.doi.org/10.1148/radiol.2412051185.

Kalomiraki, M., Thermos, K., & Chaniotakis, N. A. (2016). Dendrimers as tunable vectors of drug delivery systems and biomedical and ocular applications. *International Journal of Nanomedicine*. 11, 1-12. http://dx.doi.org/10.2147/IJN.S93069.

Keyhanian, K., Mansoori, G. A., & Rahimpour, M. (2010). Prospects for cancer nanotechnology treatment by azurin. *Dynamic Biochemistry, Process Biotechnology, and Molecular Biology*. 4(1), 48-66.

Lee, K. B., Solanki, A., Kim, D., & Jung, J. (2009). Nanomedicine: Dynamic integration of nanotechnology with biomedical science. In M. Zhang & N. Xi (Eds.), *Nanomedicine: A Systems Engineering Approach* (pp. 1-38). Singapore: Pan Stanford Publishing.

Lim, E. K., Kim, T., Paik, S., Haam, S., H, Y. M., & Lee, K. (2014). Nanomaterials for Theranostics: Recent advances and future challenges. *Chemical Reviews*. 115, 327-394. http://dx.doi.org/10.1021/cr300213b.

Long, Y. B., Gu, W. X., Pang, C., Ma, J., Gau, H. (2016). Construction of coumarin-based cross-linked micelles with pH responsive hydrazone bond and tumor targeting moiety. *Journal of Materials Chemistry B*. 4, 1480-1488. http://dx.doi.org/10.1039/c5tb02729b.

Longmire, M., Choyke, P. L., Kobayashi, H. (2008). Clearance properties of nano-sized particles and molecules as imaging agents: Considerations and caveats. *Nanomedicine*. 3(5), 703-717. http://dx.doi.org/10.2217/17435889.3.5.703.

Lu, F., Doane, T. L., Zhu, J. J., & Burda, C. (2012). Gold nanoparticles for diagnostic sensing and therapy. *Inorganica Chemica Acta.* 393, 142-153. http://dx.doi.org/10.1016/j.ica.2012.05.038.

Lyer, A. K., Khaled, G., Fang, J., Maeda, H. (2006). Exploiting the enhanced permeability and retention effect for tumor targeting. *Drug Discovery Today*. 11, 812-818.

Majidi, S., Sehrig, F. Z., Samiei, M., Milani, M. Abbasi, E., Dadashzadeh, K., & Akbarzadeh, A. (2016). Magnetic nanoparticles: Applications in gene delivery and gene therapy. *Artificial Cells, Nanomedicine, and Biotechnology*. 44(4), 1186-1193. http://dx.doi.org/10.3109/21691401.2015.1014093.

Mansoori, G. A., Brandenburg, K. S., & Shakeri-Zadeh, A. (2010). A comparative study of two folate-conjugated gold nanoparticles for cancer nanotechnology applications. *Cancers*. 2, 1911-1928. http://dx.doi.org/10.3390/cancers2041911.

Miranda, O. R., Creran, B., & Rotello, V. M. (2010). Array-based sensing with nanoparticles: "Chemical noses" for sensing biomolecules and cell surfaces. *Current Opinion in Chemical Biology*. 14(6), 728-736. http://dx.doi.org/10.1016/j.cbpa.2010.07.021.

Murgan, R., & Ramakrishna, S. (2007). Nanoengineered biomimetic bone-building blocks. In G. A. Mansoori, T. F. George, L. Assoufid, & G. Zhang (Eds.), *Molecular Building Blocks for Nanotechnology* (pp. 301-352). New York: Springer Science+Business Media.

Narod, S. A., (2012). Breast cancer in young women. *Nature Reviews Clinical Oncology*. 9, 460-470. http://dx.doi.org/10.1038/nrclinonc.2012.102.

Norton, L. (1988). A gompertzian model of human breast cancer growth. *Cancer Research*. 48(24), 7067-7071.

Patra, F. H., Bhattacharya, R., Wang, E., Katarya, A., Lau, J. S., Dutta, J. S., . . ., Mukhopadhyay, D. (2008). Targeted delivery of Gemcitabine to pancreatic adenocarcinoma using cetuximab as a targeting agent. *Cancer Research*. 68(6), 1970-1978.

Saha, K., Agasti, S. S., Kim, C., Li, X., & Rotello, V. M. (2012). Gold nanoparticles in chemical and biological sensing. *Chemical Reviews*. 112, 2739-2779. http://dx.doi.org/10.1021/cr2001178.

Saltan, N., Kutlu, H. M., Hur, D., Iscan, A., & Say, R. (2011). Interaction of cancer cells with magnetic nanoparticles modified by methacrylamido-folic acid. *International Journal of Nanomedicine*. 6, 477-484. http://dx.doi.org/10.2147/IJN.S16803.

Stein, W. D., Gulley, J. L., Schlom, J., Madan, R. A., Dahut, . . . W., Figg, W. D. (2010). Tumor regression and growth rates determined in five intramural NCI prostate cancer trials: The growth rate constant as an indicator of therapeutic efficacy. *Clinical Cancer Research*. 17(4), 907-917. http://dx.doi.org/10.1158/1078-0432.CCR-10-1762.

Tang, L., Gabrielson, N. P., Uckun, F. M., Fan, T. M., & Cheng, J. (2013). Size-dependent tumor penetration and in vivo efficacy of monodisperse drug-silica nanoconjugates. *Molercular Pharmaceutics*. 10(3), 883-892. http://dx.doi.org/10.1021/mp300684a.

Thurn, K. T., Arora, H., Paunesku, T., Wu, A., Brown, E. M. B., Doty, C., … Woloschak, G. (2011). Endocytosis of titanium dioxide nanoparticles in prostate cancer PC-3M cells. *Nanomedicine.* 7(2), 123-130. http://dx.doi.org/ 10.1016/j.nano.2010.09.004.

Tseng, J. F., Warshaw, A. L., Sahani, D. V., Lauwers, G. Y., Rattner, D. W., & Castillo, C. F. (2005). Serous cystadenoma of the pancreas: Tumor growth rates and recommendations for treatment. *Annals of Surgery*. 242, 413-421. http://dx.doi.org/10.1097/01.sla.0000179651.21193.2c.

Ulmschneider, M. B., & Searson, P. C. (2015). Mathematical models of the steps involved in the systemic delivery of a chemotherapeutic to a solid tumor: From circulation to survival. *Journal of Controlled Release*. 212, 78-84. http://dx.doi.org/10.1016/j.jconrel.2015.06.026.

Verma, A. K., & Sachin, K. (2008). Novel hydrophilic drug polymer nano-conjugates of cisplatin showing long blood retention profile: Its release kinetics, cellular uptake and bio-distribution. *Current Drug Delivery*. 5, 120-126.

Yang, B., Zuo, Y., Zou, Q., Li, L., Li, J., Man, Y., & Li, Y. (2016). Effect of ultrafine poly(ε-caprolactone) fibers on calcium phosphate cement: In vitro degradation and in vivo regeneration. *International Journal of Nanomedicine*. 11, 163-177. http://dx.doi.org/10.2147/IJN.S91596.

Zharov, V. P., Lutfullin, R. R., & Galitovskaya, E. N. (2005). Microbubbles-overlapping mode for laser killing of cancer cells with absorbing nanoparticle clusters. *Journal of Applied Physics*. 38, 2571-2581.

APPENDIX

R CODE USED FOR EXAMPLES IN CHAPTER 2 AND CHAPTER 3

```
#Chapter 2
#Covariance function for N(t) Cov(N(t),N(t+s))
CovN = function(t, s=0,sigma,a,d1star)
{
  sigma[5]^2*((exp(-2*d1star*t)-exp(-2*a[5]*t))*(2*a[5]-2*d1star)^-1)*exp(-d1star*s)
}

#Covariance function for I(t) function Cov(I(t),I(t+s))
CovI = function(t, s=0, sigma, gamma, a, d, dstar, i=1)
{
  sigma[i+2]^2*(2*(a[i+2]-gamma[i]))^-1*exp(-gamma[i]*s)*(exp(-2*gamma[i]*t)-exp(-
2*a[i+2]*t))+
    (d[i]*sigma[1]*(dstar-gamma[i])^-1)^2*((exp(-2*dstar*t)-exp(-2*a[1]*t))*(2*(a[1]-dstar))^-
1*exp(-dstar*s)+ (exp(-2*gamma[i]*t)-exp(-2*a[1]*t))*(2*(a[1]-gamma[i]))^-1*exp(-
gamma[i]*s)- (exp(-dstar*s)+exp(-gamma[i]*s))*(exp(-(dstar+gamma[i])*t)-exp(-
2*a[1]*t))*(2*a[1]-(dstar+gamma[1]))^-1)
}
#The mean function for I(t)-kN(t) , A(t)
GaussianMean = function(t, C, n,k,d,gamma,lambda,dstar,d1star)
{
  ret = (C*d[1]*(dstar-gamma[1])^-1)*(exp(-gamma[1]*t)-exp(-dstar*t))-k*n*exp(-d1star*t)
}


#The covariance function for I(t)-kN(t) , B(t,t)
GaussianVar = function(t, s=0, k, sigma, a, gamma, d, dstar, d1star)
{
CovI(t=t, s=s, sigma=sigma, gamma=gamma, a=a, d=d, dstar=dstar) +
k^2*CovN(t=t,s=s,sigma=sigma,a=a,d1star=d1star)
}
#the following 6 lines are a line by line writeup of the g(t) function and are not important on their
own
#this was done for easier debugging
nl1 = function(t,k,a,sigma,d1star)
{
  k^2*sigma[5]^2*(2*(a[5]-d1star))^-1*((2*a[5]-d1star)*exp(-2*a[5]*t)-d1star*exp(-
2*d1star*t))
}
nl2 = function(t,a,sigma,gamma)
{
  sigma[3]^2*(2*(a[3]-gamma[1]))^-1*((2*a[3]-gamma[1])*exp(-2*a[3]*t)-gamma[1]*exp(-
2*gamma[1]*t))
}
nl3 = function(t,d,a,sigma,gamma,dstar)
{
```

```
  (d[1]*sigma[1]*(dstar-gamma[1])^-1)^2*(2*(a[1]-dstar))^-1*((2*a[1]-dstar)*exp(-2*a[1]*t)-
dstar*exp(-2*dstar*t))
}

nl4 = function(t,d,a,sigma,gamma,dstar)
{
  (d[1]*sigma[1]*(dstar-gamma[1])^-1)^2*(2*(a[1]-gamma[1]))^-1*((2*a[1]-gamma[1])*exp(-
2*a[1]*t)-gamma[1]*exp(-2*gamma[1]*t))
}

nl5 = function(t,d,a,sigma,gamma,dstar)
{
  (d[1]*sigma[1]*(dstar-gamma[1])^-1)^2*(2*a[1]-(dstar+gamma[1]))^-1*((2*a[1]-dstar)*exp(-
2*a[1]*t)-gamma[1]*exp(-(dstar+gamma[1])*t))
}

nl6 = function(t,d,a,sigma,gamma,dstar)
{
  (d[1]*sigma[1]*(dstar-gamma[1])^-1)^2*(2*a[1]-(dstar+gamma[1]))^-1*((2*a[1]-
gamma[1])*exp(-2*a[1]*t)-dstar*exp(-(dstar+gamma[1])*t))
}
#Derivative of A
ADeriv = function(t,C,n,k,d,gamma,lambda,dstar,d1star)
{
  C*d[1]*(dstar-gamma[1])^-1*(-gamma[1]*exp(-gamma[1]*t) + dstar*exp(-dstar*t)) +
k*n*d1star*exp(-d1star*t)
}

#the g function used in the pdf , g(t)
gfunction = function(t,C,n,k,d,a,sigma,lambda,gamma,dstar,d1star)
{
    ret = -GaussianMean(t=t,C=C,n=n,k=k,d = d,gamma = gamma,lambda = lambda,dstar =
dstar,d1star = d1star)*
    GaussianVar(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,dstar = dstar,d1star
= d1star)^-1*
    (nl1(t=t,k = k,a=a,sigma=sigma,d1star=d1star)+
       nl2(t=t,a=a,sigma=sigma,gamma=gamma)+
       nl3(t=t,d=d,a=a,sigma=sigma,gamma=gamma,dstar=dstar)+
       nl4(t=t,d=d,a=a,sigma=sigma,gamma=gamma,dstar=dstar)-
       nl5(t=t,d=d,a=a,sigma=sigma,gamma=gamma,dstar=dstar)-
       nl6(t=t,d=d,a=a,sigma=sigma,gamma=gamma,dstar=dstar))+
    ADeriv(t=t,C = C,n = n,k = k,d = d,gamma = gamma,lambda = lambda,dstar = dstar,d1star =
d1star)
ifelse(ret >= 0, ret,0) # fix for tail behavior
}
```

```
# The pdf of the model.
GaussianProbability = function(t,C,n,k,d,a,sigma,lambda,gamma,dstar,d1star)
{
  ret=gfunction(t=t,C=C,n = n,k = k,d = d,a = a,sigma = sigma,lambda = lambda,gamma =
gamma,dstar = dstar,d1star = d1star)*
    (2*pi*GaussianVar(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,dstar =
dstar,d1star = d1star))^(-.5)*
    exp(-.5*GaussianMean(t = t,C = C,n = n,k = k,d = d,gamma = gamma,lambda = lambda,dstar
= dstar,d1star = d1star)^2*
          GaussianVar(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,dstar =
dstar,d1star = d1star)^-1)
}
#vector version of the integral of the pdf of the model
pfunc = function(t,lower=1e-100,C,n,k,a,sigma,d,gamma,lambda,dstar,d1star)
{
integrate(GaussianProbability,lower=lower,upper=t,C=C,n=n,k=k,a=a,sigma=sigma,d=d,gamma
=gamma,lambda=lambda,dstar=dstar,d1star=d1star)$value
}
pfunc.v = Vectorize(pfunc,"t")
# The expected value of t given the parameters under the assumptions of the model
GaussianExpectedValue = function(t,C,n,k,d,aval,sigma,lambda,gamma,dstar,d1star)
{
  ret = t*gfunction(t=t,C=C,n = n,k = k,d = d,a = aval,sigma = sigma,lambda = lambda,gamma =
gamma,dstar = dstar,d1star = d1star)*
  (2*pi*GaussianVar(t = t,s = 0,k = k,sigma = sigma,a = aval,gamma = gamma,d = d,dstar =
dstar,d1star = d1star))^(-.5)*
  exp(-.5*GaussianMean(t = t,C = C,n = n,k = k,d = d,gamma = gamma,lambda = lambda,dstar =
dstar,d1star = d1star)^2*
        GaussianVar(t = t,s = 0,k = k,sigma = sigma,a = aval,gamma = gamma,d = d,dstar =
dstar,d1star = d1star)^-1)
}
#Badfunction basically finds the end points of the distribution and truncates them because
strange things happen at the tails
#It is called bad function because it is slow and bad.
Badfunction = function(C,n,k,d,a,sigma,lambda,gamma,dstar,d1star,tolerance=1e-
3,byincrease=.01,start.at=0,max_iterations=10000)
{
  #initialize variables.  I don't know if we have to do this in R but in C++ we do so there.  Also
nick, remember not to put ;
  CurNumber = 0
  firstFound = FALSE
  Iternumber = start.at #start here.  Hopefully we'd start somewhere around the expectation we
want - some epsilon.
```

```
  #starting at 0 in some cases can make this take a very long time.
  lower = 0
  upper = 0
  iteration = 0
  #while we haven't hit the max number of iterations we want and we have a finite CurNumber
  while((is.nan(CurNumber) | CurNumber<Inf) & iteration < max_iterations)
  {
    CurNumber =
GaussianExpectedValue(t=Iternumber,C=C,n=n,k=k,d=d,a=a,sigma=sigma,lambda=lambda,gam
ma=gamma,dstar=dstar,d1star=d1star)
    try(if(is.infinite(CurNumber)) stop("Badfunction - CurNumber is infinite.  This is bad."))
    if(!is.nan(CurNumber) & (CurNumber > tolerance & firstFound==FALSE))
    {
      lower=Iternumber
      firstFound=TRUE
    }
    else if(CurNumber <= tolerance & firstFound==TRUE)
    {
      upper=Iternumber
      break; #bad programming practice =)
    }
    Iternumber=Iternumber+byincrease #number here to find out where we should go.
    iteration=iteration+1
  }
  try(if(iteration >= max_iterations) stop("Max iterations reached in BadFunction"))
  list("lower"=lower,"upper"=upper) #return our lower and upper integration sections.  If
tolerance is chosen well it shouldn't cause a large difference
}
#vectorizing the funcions, might be useful for speed increase of algorithm
#will test this out later
hfunc =
function(lint=0,uint=Inf,C,n,k,a,sigma,d,gamma,lambda,dstar,d1star)#,TruncateWindow=c(0,5),t
olerance = 1e-3)
{
integrate(GaussianExpectedValue,lower=lint,upper=uint,C=C,n=n,k=k,aval=a,sigma=sigma,d=d
,gamma=gamma,lambda=lambda,dstar=dstar,d1star=d1star)$value
}
zfunc = Vectorize(hfunc,"C")
#this is the test function to integrate within the bounds we set up.
testfunc = function(C,n,k,a,sigma,d,gamma,lambda,dstar,d1star,tolerance=1e-
3,start.at=0,max_iterations=10000,bounds.increaseby=.01)
{
  bounds = Badfunction(C = C,n = n,k = k,d = d,a = a,sigma = sigma,lambda = lambda,gamma =
gamma,dstar = dstar,d1star = d1star,tolerance=tolerance,start.at=start.at,max_iterations =
max_iterations,byincrease = bounds.increaseby)
```

```
integrate(GaussianExpectedValue,lower=bounds$lower,upper=bounds$upper,C=C,n=n,k=k,aval
=a,sigma=sigma,d=d,gamma=gamma,lambda=lambda,dstar=dstar,d1star=d1star)$value
}
testfunc.v = Vectorize(testfunc,"C")


#the algorithm proposed by ebrihimi et al based on a frequentist framework with a slight
modification for increments.
AlgFrequentist =
function(alpha,Cstart,n,k,a,sigma,d,gamma,lambda,start.at=0,max_iterations=1e99,max_bounds
_iterations=1e4,tolerance=1e-3,bounds.increaseby=.01,increment = 1)
{
  FD=FALSE #first iteration done
  LT=FALSE #less than or greater than handler.
  C = Cstart #what we start at
  currentT=Inf #no matter what this is going to start somewhere, just use infinity.
  iteration = 0
  #Integrated=0
  dstar=d[1]+d[2]+d[3]
  d1star=lambda[1]-lambda[2]

  #find where to truncate the pdf because of tail behavior.  We know that p(t) will be increasing
with C so we only need to find this once

  while(iteration <= max_iterations)
  {
    #Previous inetgral
    PrevIntegrate = currentT

    #Expected Value
    currentT = testfunc.v(C = C,n = n,k = k,a = a,sigma = sigma,d = d,gamma = gamma,lambda =
lambda,dstar = dstar,d1star = d1star,tolerance = tolerance,start.at =start.at,max_iterations =
max_bounds_iterations,bounds.increaseby=bounds.increaseby)
    #this is the value of the integral
    #if less than, we're going to be subtracting nanoconjugates
    if(currentT < alpha)
    {
      if(FD==TRUE && LT==FALSE) #we were above alpha, now we're below.
      {
        C=C+increment
        #return our C, our current time, and the integrand stuff
        break
      }
      FD=TRUE
```

```
    LT=TRUE
    C=C-increment
   }
   #greater than means adding nanoconjugates
   else if(currentT > alpha)
   {
    if(FD==TRUE && LT==TRUE) # we were below alpha now we're above
    {
     C=C-increment
     #return our C, our current time, and the integrand stuff
     break
    }
    FD=TRUE
    LT=FALSE
    C=C+increment
   }
   #if it's equal, we did it perfectly, this is very unlikely (p(this) = 0) but if it happens we're done
   else if(currentT==alpha)
   {
    FD=TRUE
    #return our C, our current time, and the integrand stuff
    break
   }
   try(if(C < n*k) stop("Something broke, C < n*k, this is impossible based on this models
assumptions.  It is likely that alpha is set too high"))
   iteration=iteration+1
  }
  if(increment > 1)
  {
   increment = increment/10
   AlgFrequentist(alpha = alpha,Cstart = C,n = n,k = k,a = a,sigma = sigma,d = d,gamma =
gamma,lambda = lambda,start.at = start.at,max_iterations =
max_iterations,max_bounds_iterations = max_bounds_iterations,tolerance =
tolerance,bounds.increaseby = bounds.increaseby,increment = increment)
  }
  else
   list("Expected Nanoconjugates" = C,"Expected Time Units" = PrevIntegrate,"Current
Integral" = currentT, "Previous Integral"= PrevIntegrate, iteration=iteration)
}
ntest = 210
atest=c(.142,0,.0144,0,.177)
sigmatest=c(.5,0,.61,0,.44)
gammatest=c(1.2,.006)
dtest=c(1.2,.1,.1)
lambdatest=c(.81,.2)
```

```
tolerancetest=1e-15
start.at.test=0
increase.by.test=.001


ktest=5
ex1.5.1tu = AlgFrequentist(alpha = 1,Cstart = 1750,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e3,max_bounds_iterations = 1e5,bounds.increaseby =
increase.by.test,increment = 10)
ex1.5.halftu = AlgFrequentist(alpha = .5,Cstart = 2050,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 10)

ktest=10
ex1.10.1tu = AlgFrequentist(alpha = 1,Cstart = 4050,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 100)
ex1.10.halftu = AlgFrequentist(alpha = .5,Cstart = 4050,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 100)

ktest=15
ex1.15.1tu = AlgFrequentist(alpha = 1,Cstart = 8000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 100)
ex1.15.halftu = AlgFrequentist(alpha = .5,Cstart = 8000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 100)

ktest=20
ex1.20.1tu = AlgFrequentist(alpha = 1,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)
ex1.20.halftu = AlgFrequentist(alpha = .5,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)
```

```
ktest=25
ex1.25.1tu = AlgFrequentist(alpha = 1,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)
ex1.25.halftu = AlgFrequentist(alpha = .5,Cstart = 14000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)

ktest=30
ex1.30.1tu = AlgFrequentist(alpha = 1,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)
ex1.30.halftu = AlgFrequentist(alpha = .5,Cstart = 14000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)

ktest=5
ex2.5.1tu = AlgFrequentist(alpha = 1,Cstart = 11500,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e50,bounds.increaseby =
increase.by.test,increment = 100)
ex2.5.halftu = AlgFrequentist(alpha = .5,Cstart = 16500,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 100)

ktest=10
ex2.10.1tu = AlgFrequentist(alpha = 1,Cstart = 22500,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=100)
ex2.10.halftu = AlgFrequentist(alpha = .5,Cstart = 32000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 1000)


ktest=15
ex2.15.1tu = AlgFrequentist(alpha = 1,Cstart = 33750,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
```

```
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex2.15.halftu = AlgFrequentist(alpha = .5,Cstart = 48000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 1000)

ktest=20
ex2.20.1tu = AlgFrequentist(alpha = 1,Cstart = 45000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex2.20.halftu = AlgFrequentist(alpha = 0.5,Cstart = 64000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=25
ex2.25.1tu = AlgFrequentist(alpha = 1,Cstart = 56000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex2.25.halftu = AlgFrequentist(alpha = 0.5,Cstart = 80000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=30
ex2.30.5tu = AlgFrequentist(alpha = 1,Cstart = 66000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex2.30.1tu = AlgFrequentist(alpha = 0.5,Cstart = 96000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=5
ex3.5.1tu = AlgFrequentist(alpha = 1,Cstart = 6000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e50,bounds.increaseby =
increase.by.test,increment = 1000)
ex3.5.halftu = AlgFrequentist(alpha = .5,Cstart = 9000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
```

```
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment = 1000)

ktest=10
ex3.10.1tu = AlgFrequentist(alpha = 1,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex3.10.halftu = AlgFrequentist(alpha = .5,Cstart = 10000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=15
ex3.15.1tu = AlgFrequentist(alpha = 1,Cstart = 15000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex3.15.halftu = AlgFrequentist(alpha = .5,Cstart = 15000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=20
ex3.20.1tu = AlgFrequentist(alpha = 1,Cstart = 20000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex3.20.halftu = AlgFrequentist(alpha = .5,Cstart = 20000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)

ktest=25
ex3.25.1tu = AlgFrequentist(alpha = 1,Cstart = 25000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex3.25.halftu = AlgFrequentist(alpha = .5,Cstart = 25000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)


ktest=30
```

```
ex3.30.1tu = AlgFrequentist(alpha = 1,Cstart = 30000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)
ex3.30.halftu = AlgFrequentist(alpha = .5,Cstart = 30000,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test,increment=1000)



#Chapter 3
Nick_Average =
function(C,n,k,a,sigma,d,gamma,lambda,start.at=0,max_iterations=1e99,max_bounds_iterations
=1e4,tolerance=1e-3,bounds.increaseby=.01,loops=50000)
{
 fails=0
 n.avg = rpois(loops,n)
 k.avg = rpois(loops,k)
 d1star = lambda[1]-lambda[2]
 a1 = rexp(loops,1/a[1])
 a2 = 0
 a3 = rexp(loops,1/a[3])
 a4 = 0
 a5 = rexp(loops,1/a[5])

 #a.avg = data.frame(a1,a2,a3,a4,a5)

 sigma1 = rexp(loops,1/sigma[1])
 sigma2 = 0
 sigma3 = rexp(loops,1/sigma[3])
 sigma4 = 0
 sigma5 = rexp(loops,1/sigma[5])

 #sigma.avg = data.frame(sigma1,sigma2,sigma3,sigma4,sigma5)

 d1 = rexp(loops,1/d[1])
 d2 = rexp(loops,1/d[2])
 d3 = rexp(loops,1/d[3])

 #d.avg = data.frame(d1,d2,d3)

 gamma1 = rexp(loops,1/gamma[1])
 gamma2 = rexp(loops,1/gamma[2])
```

```
  #gamma.avg = data.frame(gamma1,gamma2)

  lambda1 = rexp(loops,1/lambda[1])
  lambda2 = rexp(loops,1/lambda[2])

  #lambda.avg = data.frame(lambda1,lambda2)

  require(smoothmest)
  #dstar.avg = rgamma(loops,shape=3,rate=d.avg[1]+d.avg[2]+d.avg[3])
  d1star.avg = rdoublex(loops,mu=mean(lambda1),lambda=mean(lambda2))

  integrate.vec = numeric(loops) #need this for large factorials
  i=1
  while(i <= loops)
  {
   while(d1star.avg[i] < 0)
   {
    d1star.avg[i] = rdoublex(1,mu=mean(lambda1),lambda=mean(lambda2))
   }
   while(gamma1[i] > (d1[i]+d2[i]+d3[i]) | gamma2[i] > (d1[i]+d2[i]+d3[i]))
   {
    gamma1[i] = rexp(1,1/gamma[1])
    gamma2[i] = rexp(1,1/gamma[2])

    d1[i] = rexp(1,1/d[1])
    d2[i] = rexp(1,1/d[2])
    d3[i] = rexp(1,1/d[3])

   }

   integrate.vec[i] = testfuncb(C = C,n = n,k = k,a = c(a1[i],0,a3[i],0,a5[i]),
          sigma = c(sigma1[i],0,sigma3[i],0,sigma5[i]),
          d = c(d1[i],d2[i],d3[i]),
          gamma = c(gamma1[i],gamma2[i]),
          d1star = d1star.avg[i],tolerance = tolerance,start.at = start.at,max_iterations =
max_bounds_iterations,bounds.increaseby = bounds.increaseby,

v=c(d[1],d[2],d[3],a[1],a[3],a[5],sigma[1],sigma[3],sigma[5],gamma[1],gamma[2],d1star.avg[i],
n.avg[i],k.avg[i]))
   if(integrate.vec[i] == 0) #it failed, resample.
   {
    fails = fails+1
    print(i)
    print(a1[i])
    print(a3[i])
```

```
    print(a5[i])

    print(d1[i])
    print(d2[i])
    print(d3[i])

    print(sigma1[i])
    print(sigma3[i])
    print(sigma5[i])

    print(gamma1[i])
    print(gamma2[i])
    print(d1star.avg[i])
    print(fails)
    print(fails/i)
    try(if(integrate.vec[i]==0) stop("Error converging"))
    a1[i] = rexp(1,1/a[1])
    a3[i] = rexp(1,1/a[3])
    a5[i] = rexp(1,1/a[5])

    sigma1[i] = rexp(1,1/sigma[1])
    sigma3[i] = rexp(1,1/sigma[3])
    sigma5[i] = rexp(1,1/sigma[5])

    d1[i] = rexp(1,1/d[1])
    d2[i] = rexp(1,1/d[2])
    d3[i] = rexp(1,1/d[3])

    gamma1[i] = rexp(1,1/gamma[1])
    gamma2[i] = rexp(1,1/gamma[2])

    d1star.avg[i] = rdoublex(1,mu=mean(lambda1),lambda=mean(lambda2))

    n.avg[i] = rpois(1,n)
    k.avg[i] = rpois(1,k)

    i=i-1
   }
   i=i+1
  }
  ret = mean(integrate.vec)

  ret
}
```

```
nl1b = function(t,k,a,sigma,d1star)
{
  k^2*sigma[5]^2*(2*(a[5]-d1star))^-1*((2*a[5]-d1star)*exp(-2*a[5]*t)-d1star*exp(-
2*d1star*t))
}

nl2b = function(t,a,sigma,gamma)
{
  sigma[3]^2*(2*(a[3]-gamma[1]))^-1*((2*a[3]-gamma[1])*exp(-2*a[3]*t)-gamma[1]*exp(-
2*gamma[1]*t))
}

nl3b = function(t,d,a,sigma,gamma)
{
  (d[1]*sigma[1]*((d[1]+d[2]+d[3])-gamma[1])^-1)^2*(2*(a[1]-(d[1]+d[2]+d[3])))^-1*((2*a[1]-
(d[1]+d[2]+d[3]))*exp(-2*a[1]*t)-(d[1]+d[2]+d[3])*exp(-2*(d[1]+d[2]+d[3])*t))
}

nl4b = function(t,d,a,sigma,gamma)
{
  (d[1]*sigma[1]*((d[1]+d[2]+d[3])-gamma[1])^-1)^2*(2*(a[1]-gamma[1]))^-1*((2*a[1]-
gamma[1])*exp(-2*a[1]*t)-gamma[1]*exp(-2*gamma[1]*t))
}

nl5b = function(t,d,a,sigma,gamma)
{
  (d[1]*sigma[1]*((d[1]+d[2]+d[3])-gamma[1])^-1)^2*(2*a[1]-((d[1]+d[2]+d[3])+gamma[1]))^-
1*((2*a[1]-(d[1]+d[2]+d[3]))*exp(-2*a[1]*t)-gamma[1]*exp(-((d[1]+d[2]+d[3])+gamma[1])*t))
}

nl6b = function(t,d,a,sigma,gamma)
{
  (d[1]*sigma[1]*((d[1]+d[2]+d[3])-gamma[1])^-1)^2*(2*a[1]-((d[1]+d[2]+d[3])+gamma[1]))^-
1*((2*a[1]-gamma[1])*exp(-2*a[1]*t)-(d[1]+d[2]+d[3])*exp(-((d[1]+d[2]+d[3])+gamma[1])*t))
}

gfunctionb = function(t,C,n,k,d,a,sigma,gamma,d1star)
{
  ret = -GaussianMeanb(t=t,C=C,n=n,k=k,d = d,gamma = gamma,d1star = d1star)*
    GaussianVarb(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,d1star = d1star)^-
1*
    (nl1b(t=t,k = k,a=a,sigma=sigma,d1star=d1star)+
       nl2b(t=t,a=a,sigma=sigma,gamma=gamma)+
       nl3b(t=t,d=d,a=a,sigma=sigma,gamma=gamma)+
       nl4b(t=t,d=d,a=a,sigma=sigma,gamma=gamma)-
```

```
      nl5b(t=t,d=d,a=a,sigma=sigma,gamma=gamma)-
      nl6b(t=t,d=d,a=a,sigma=sigma,gamma=gamma))+
    ADerivb(t=t,C = C,n = n,k = k,d = d,gamma = gamma,d1star = d1star)

  ifelse(ret >= 0, ret,0) # fix for tail behavior
}

CovIb = function(t, s=0, sigma, gamma, a, d,i=1)
{
  sigma[i+2]^2*(2*(a[i+2]-gamma[i]))^-1*exp(-gamma[i]*s)*(exp(-2*gamma[i]*t)-exp(-
2*a[i+2]*t))+
    (d[i]*sigma[1]*((d[1]+d[2]+d[3])-gamma[i])^-1)^2*((exp(-2*(d[1]+d[2]+d[3])*t)-exp(-
2*a[1]*t))*(2*(a[1]-(d[1]+d[2]+d[3])))^-1*exp(-(d[1]+d[2]+d[3])*s)+
                                    (exp(-2*gamma[i]*t)-exp(-2*a[1]*t))*(2*(a[1]-gamma[i]))^-1*exp(-
gamma[i]*s)-
                                    (exp(-(d[1]+d[2]+d[3])*s)+exp(-gamma[i]*s))*(exp(-
((d[1]+d[2]+d[3])+gamma[i])*t)-exp(-2*a[1]*t))*(2*a[1]-((d[1]+d[2]+d[3])+gamma[i]))^-1)
}

GaussianMeanb = function(t, C, n,k,d,gamma,d1star)
{
  ret = (C*d[1]*((d[1]+d[2]+d[3])-gamma[1])^-1)*(exp(-gamma[1]*t)-exp(-(d[1]+d[2]+d[3])*t))-
k*n*exp(-d1star*t)

}

GaussianVarb = function(t, s=0, k, sigma, a, gamma, d, d1star)
{
  CovIb(t=t, s=s, sigma=sigma, gamma=gamma, a=a, d=d) +
k^2*CovN(t=t,s=s,sigma=sigma,a=a,d1star=d1star)
}

ADerivb = function(t,C,n,k,d,gamma,d1star)
{
  C*d[1]*((d[1]+d[2]+d[3])-gamma[1])^-1*(-gamma[1]*exp(-gamma[1]*t) +
(d[1]+d[2]+d[3])*exp(-(d[1]+d[2]+d[3])*t)) + k*n*d1star*exp(-d1star*t)
}

#nntest = AlgFrequentist(alpha = .5,Cstart = 6150,n = ntest,k = ktest,a = atest,sigma =
sigmatest,d = dtest,gamma = gammatest,lambda = lambdatest,tolerance=tolerancetest,start.at =
start.at.test,max_iterations = 1e6,max_bounds_iterations = 1e4,bounds.increaseby =
increase.by.test)
GaussianProbabilityb = function(t,C,n,k,d,a,sigma,gamma,d1star,v)
{
  gfunctionb(t=t,C=C,n = n,k = k,d = d,a = a,sigma = sigma,gamma = gamma,d1star = d1star)*
```

```
   (2*pi*GaussianVarb(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,d1star =
d1star))^(-.5)*
   exp(-.5*GaussianMeanb(t = t,C = C,n = n,k = k,d = d,gamma = gamma,d1star = d1star)^2*
        GaussianVarb(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,d1star =
d1star)^-1)
}

GaussianExpectedValueb =
function(t,C,n,k,d,a,sigma,gamma,d1star,v=c(d[1],d[2],d[3],a[1],a[3],a[5],sigma[1],sigma[3],sig
ma[5],gamma[1],gamma[2],d1star))
{
 t*gfunctionb(t=t,C=C,n = n,k = k,d = d,a = a,sigma = sigma,gamma = gamma,d1star = d1star)*
   (2*pi*GaussianVarb(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,d1star =
d1star))^(-.5)*
   exp(-.5*GaussianMeanb(t = t,C = C,n = n,k = k,d = d,gamma = gamma,d1star = d1star)^2*
        GaussianVarb(t = t,s = 0,k = k,sigma = sigma,a = a,gamma = gamma,d = d,d1star =
d1star)^-1)
}

testfuncb = function(C,n,k,a,sigma,d,gamma,d1star,v,tolerance=1e-
3,start.at=0,max_iterations=10000,bounds.increaseby=.01)
{
 ret = 0
 ret1 = 0
 ret2 = 0
 bounds = Badfunctionb(C = C,n = n,k = k,d = d,a = a,sigma = sigma,gamma = gamma,d1star =
d1star,v=v,tolerance=tolerance,start.at=start.at,max_iterations = max_iterations,byincrease =
bounds.increaseby)
 if(bounds$converged == TRUE)
   ret1 =
integrate(GaussianExpectedValueb,lower=bounds$lower,upper=bounds$upper,C=C,n=n,k=k,a=
a,sigma=sigma,d=d,gamma=gamma,d1star=d1star,v=v)$value

 if(ret1 != 0)
 {
   ret2 =
integrate(GaussianProbabilityb,lower=bounds$lower,upper=bounds$upper,C=C,n=n,k=k,a=a,sig
ma=sigma,d=d,gamma=gamma,d1star=d1star,v=v)$value
   ret = ret1/ret2
 }
 ret
}
testfuncb.v = Vectorize(testfuncb,"C")
```

```
Badfunctionb = function(C,n,k,d,a,sigma,gamma,d1star,v,tolerance=1e-
3,byincrease=.01,start.at=0,max_iterations=10000)
{
 #initialize variables.  I don't know if we have to do this in R but in C++ we do so there.  Also
nick, remember not to put ;
 CurNumber = 0
 firstFound = FALSE
 converged = FALSE
 Iternumber = start.at #start here.  Hopefully we'd start somewhere around the expectation
 #starting at 0 in some cases can make this take a very long time.
 lower = 0
 upper = 0
 iteration = 0
 #while we haven't hit the max number of iterations we want and we have a finite CurNumber
 while((is.nan(CurNumber)| is.na(CurNumber) | CurNumber<Inf) & iteration < max_iterations)
 {
  CurNumber =
GaussianExpectedValueb(t=Iternumber,C=C,n=n,k=k,d=d,a=a,sigma=sigma,gamma=gamma,d1
star=d1star,v=v)
  try(if(is.infinite(CurNumber)) stop("Badfunction - CurNumber is infinite.  This is bad."))
  if(!is.nan(CurNumber) & !is.na(CurNumber) & (CurNumber > tolerance &
firstFound==FALSE))
  {
   lower=Iternumber
   firstFound=TRUE
  }
  else if(CurNumber <= tolerance & firstFound==TRUE)
  {
   upper=Iternumber
   converged = TRUE
   break; #bad programming practice =)
  }
  Iternumber=Iternumber+byincrease #number here to find out where we should go.
  iteration=iteration+1
 }
 try(if(iteration >= max_iterations) stop("Max iterations reached in BadFunction"))
 list("lower"=lower,"upper"=upper,"converged"=converged) #return our lower and upper
integration sections.  If tolerance is chosen well it shouldn't cause a large difference
}

ex1.50000i.c1743 = Nick_Average(C = 1743,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
```

```
ex1.50000i.c1800 = Nick_Average(C = 1800,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
ex1.50000i.c1850 = Nick_Average(C = 1850,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
ex1.50000i.c1900 = Nick_Average(C = 1900,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
ex1.50000i.c1950 = Nick_Average(C = 1950,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
ex1.50000i.c2000 = Nick_Average(C = 2000,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
ex1.50000i.c2050 = Nick_Average(C = 2050,n = 5,k = ktest,a = atest,sigma = sigmatest,d =
dtest,gamma = gammatest,lambda = lambdatest,start.at = start.at.test,bounds.increaseby =
.001,loops = 50000,max_bounds_iterations = 1e5,tolerance = 1e-15)
```