2011-05-18

Adaptive Threat Detector Testing Using Bayesian Gaussian Process Models

Bradley Thomas Ferguson
Brigham Young University - Provo

Adaptive Threat Detector Testing Using

Bayesian Gaussian Process Models

Bradley Ferguson

A project submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Dr. Shane Reese, Chair
Dr. Natalie Blades
Dr. Gilbert Fellingham

Department of Statistics

Brigham Young University

August 2011

ABSTRACT

Adaptive Threat Detector Testing Using
Bayesian Gaussian Process Models

Bradley Ferguson
Department of Statistics, BYU
Master of Science

Detection of biological and chemical threats is an important consideration in the modern national defense policy. Much of the testing and evaluation of threat detection technologies is performed without appropriate uncertainty quantification. This paper proposes an approach to analyzing the effect of threat concentration on the probability of detecting chemical and biological threats. The approach uses a probit semi-parametric formulation between threat concentration level and the probability of instrument detection. It also utilizes a bayesian adaptive design to determine at which threat concentrations the tests should be performed. The approach offers unique advantages, namely, the flexibility to model non-monotone curves and the ability to test in a more informative way. We compare the performance of this approach to current threat detection models and designs via a simulation study.

Keywords: Gaussian process, bayesian, adaptive design

ACKNOWLEDGMENTS

I would like to thank and acknowledge all those that have helped me through this master's project. First, I must thank Dr. Reese. He was willing to take me on as a research assistant at the beginning of my junior year and has helped me ever since. He has consistently found time to meet with me and help me progress in my research. Second, I would like thank my parents and family. They were always asking me how my research was going and would get excited with my accomplishments. Lastly, I must thank my wife Emily. She is my best friend and has been incredibly patient with all the time that this project has taken. Whenever things get hard, she is there for me.

CONTENTS

Contents iv

1 Introduction 1

1.1 Biological Threats 1

1.2 Bayesian Gaussian Process Models 2

1.3 Adaptive Design 3

2 Literature Review 4

2.1 Bayesian Analysis 4

2.2 Markov Chain Monte Carlo 5

2.3 Gaussian Process Models 7

2.4 Adaptive Design 9

3 Methods 11

3.1 Logistic Model 11

3.2 Gaussian Process Model 15

3.3 Adaptive Trial Design 19

3.4 Simulation Study 21

4 Results 23

4.1 Posterior Distributions 23

4.2 Mean Squared Error of $\widehat{C50}$ 29

4.3 Posterior Variances 31

5 Conclusions 34
5.1 Future Work 36

Bibliography 37

Appendices 39

Appendix A: Complete Conditional Distributions 40
A.1 Gaussian Process Model 40
A.2 Logistic Model 42

Appendix B: Code 43
B.1 Simulation 43
B.2 Results 67

CHAPTER 1

INTRODUCTION

Statistical models are developed in order to summarize and understand phenomena in nature. By design, they are used to capture some sort of relationship. While no model can completely capture the true relationship, some are more effective than others. In the case of biological threat detector testing, there are two values of interest: first, the concentration of the biological agent that is being tested, and second, whether or not the detector's alarm was triggered. The relationship of interest between these two variables is a probability curve that explains the probability of a successful detection given a specific threat concentration.

1.1 Biological Threats

In the past decade, bioterrorism has become a central focus of the United States military. Just one week after the events of September 11, 2001, multiple letters were sent out that contained anthrax spores which resulted in 5 deaths and 17 seriously infected. Two years later, a similar incident occurred in which two letters containing ricin were distributed, one found at the White House, another in South Carolina. Since then, bioterrorism related funding has dramatically increased. In 2003, the National Institute of Allergy and Infectious Diseases received a 1.5 billion dollar increase in funding. A year later, the Project Bioshield Act was passed which provided 5.6 billion dollars to bioterrorism prevention.

The United States Center of Disease Control and Prevention established the following formal definition for a bioterrorism attack:

> "A bioterrorism attack is the deliberate release of viruses, bacteria, or other germs (agents) used to cause illness or death in people, animals, or plants. These agents are typically found in nature, but it is possible that they could be changed to

> increase their ability to cause disease, make them resistant to current medicines, or to increase their ability to be spread into the environment. Biological agents can be spread through the air, through water, or in food. Terrorists may use biological agents because they can be extremely difficult to detect and do not cause illness for several hours to several days" (Borelli 2007).

Because of the danger brought about by bioterrorism, many detection instruments have been developed to detect threat agents. These instruments must be tested in order to determine their effectiveness. An appropriate statistical model and experimental design should be utilized when testing these instruments in order to best determine their ability to detect biological threats at varying concentrations.

1.2 Bayesian Gaussian Process Models

As the threat detection result is a binary response (detection or no detection), the most common statistical model to use is a parametric generalized linear model that utilizes the logit link. While this method is intuitive and simple, it lacks the ability to capture non-monotone probability curves. A Bayesian approach can be applied to this problem by introducing a prior distribution to specify a prior belief about certain parameters in the model. While this can be of use when the prior is informative, it still does a poor job at modeling non-monotone curves.

Instead of assuming a parametric model to model the relationship between threat concentration and response, a nonparametric model can be used. By doing so, no assumptions about the relationship are made. In this case, because there are no parameters in the model, a different kind of prior must be used—a stochastic process. A stochastic process prior is a prior distribution over *functions*.

When choosing a stochastic process prior distribution, the goal is to choose a process that would behave in a similar fashion to the function of interest. In the case of biological threat detection, a Gaussian process is the distribution of choice. It is robust, computation-

ally convenient, and can be made as flexible or rigid as needed. This is done by specifying an appropriate mean and covariance kernel that emulates attributes of the function. Then, with an appropriate assumed likelihood, the posterior distribution of the underlying function can be learned by incorporating data into the model and using Markov Chain Monte Carlo computational methods.

1.3 Adaptive Design

Because threat detection testing is an expensive and time consuming endeavor, an efficient experimental design piece should be implemented. There has been much progress made in the area of clinical trials using adaptive design techniques that use early results to help decide new dose levels and patient allocation. The same sort of benefits can be gained in the area of threat detector testing. By using an adaptive experimental design, the results from the Gaussian process model can be used to determine which concentrations should be tested to provide the most information.

CHAPTER 2

LITERATURE REVIEW

2.1 Bayesian Analysis

Introduction to Bayesian Methods

For most of the twentieth century, frequentist analysis dominated the field of statistics. Many statisticians such as L. J. Savage, Bruno de Finetti, and Dennis Lindley advocated for Bayesian methods (Carlin and Louis 2009) but the methods required solutions to complex integrals which made implementation prohibitive.

Bayesian analysis is based on Bayes Theorem which was the first use of the idea of inverse probability. Bayes Theorem states that for two events A and B,

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}.$$

A common approach to statistical analysis is to assume a parametric sampling distribution $f(\mathbf{y}|\boldsymbol{\theta})$ that governs a data set $\mathbf{y}$ and then use likelihood-based methods to make inference about the unknown parameters $\boldsymbol{\theta}$. From a Bayesian perspective, inference about the parameters uses Bayes Theorem in the form

$$P(\boldsymbol{\theta}|\mathbf{y}) = \frac{f(\mathbf{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{\int f(\mathbf{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}},$$

where $\pi(\boldsymbol{\theta})$ is an assumed prior distribution for $\boldsymbol{\theta}$, specified before any data is collected.

Often the posterior distribution, $f(\mathbf{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})$, is an unknown distribution and the marginalizing density, $\int f(\mathbf{y}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})d\boldsymbol{\theta}$, is impossible to integrate analytically. With advances in computer technology, these quantities may now be approximated using Markov Chain Monte Carlo (MCMC) computer algorithms. The posterior distribution of the parameters represents the uncertainty associated with each parameter after data has been collected.

Having a distributions of parameters conveniently allows for direct probability statements to be made about where the parameters lie.

In this paper, we approach the problem of biological threat detector testing from a Bayesian perspective. A Bayesian model is well suited to an adaptive experimental design as both involve updating assumptions in the light of new data. At each step in the experimental design, we use the posterior distribution to make decisions about where to perform the next tests. All of our computations are done in R using MCMC algorithms.

2.2 Markov Chain Monte Carlo

Much of the computation in this paper is done using MCMC methods to sample from the joint posterior distribution. Two of the most common methods are Gibbs Sampling and Metroplis-Hastings. We will discuss both in detail.

Gibbs Sampling

The Gibbs Sampling algorithm, named after J. W. Gibbs, is a way to sample from a joint distribution of two or more random variables. When doing Bayesian inference, these random variables are typically parameters of interest. The first step is to initialize a vector of random variables

$$\boldsymbol{\theta^0} = (\theta_1^0, \theta_2^0, ..., \theta_k^0).$$

Let $\pi(\boldsymbol{\theta})$ be the joint density that we want to sample from and let $\pi(\theta_i|\theta_{-i})$ be the complete conditional distribution of θ_i given all the other parameters. We then proceed by drawing

from the complete conditional distributions in the following way (Hamada et al. 2008):

$$\text{Draw } \theta_1^1 \text{ from } \pi(\theta_1|\theta_{-1}^0)$$

$$\text{Draw } \theta_2^1 \text{ from } \pi(\theta_2|\theta_1^1, \theta_3^0, \theta_4^0, ..., \theta_k^0)$$

$$\text{Draw } \theta_3^1 \text{ from } \pi(\theta_3|\theta_1^1, \theta_2^1, \theta_4^0, \theta_5^0, ..., \theta_k^0)$$

$$\vdots$$

$$\text{Draw } \theta_k^1 \text{ from } \pi(\theta_k|\theta_{-k}^1).$$

The value of k is then incremented and the process is repeated. The resulting sequence $\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, ..., \boldsymbol{\theta_k}$ are draws from the posterior distribution of $\boldsymbol{\theta}$. Smith and Roberts (2003) discuss that if the components in a Gibbs sampling algorithm are highly correlated then the convergence will take much longer. They propose blocking correlated scalars together to form sub-vectors and drawing simultaneously from a multivariate conditional distribution. In this paper, we use their approach as the parameters that make up our unknown probability function are sampled jointly from a Gaussian process. This Gaussian process is simply an extension of a multivariate normal conditional distribution.

Metropolis-Hastings

For some of the parameters in this paper, their respective complete conditional distributions are not known distributions. In these settings, a Gibbs sampler does not work and a Metropolis-Hastings MCMC procedure must be used instead. Metropolis-Hastings procedures have grown in popularity and although there are many ways to implement them, they typically exhibit the following general framework as found in Carlin and Louis (2009). Let $g(\boldsymbol{\theta})$ be the complete conditional distribution for $\boldsymbol{\theta}$ and let $q(\cdot|\boldsymbol{\theta})$ be a symmetric distribution known as the candidate distribution. Choose a starting value $\boldsymbol{\theta^0}$ and iterate the following

from $i = 2, ..., N$:

1. Draw $\boldsymbol{\theta^*}$ from $q(\cdot|\boldsymbol{\theta^{i-1}})$
2. Compute $p = \dfrac{g(\boldsymbol{\theta^*})}{g(\boldsymbol{\theta^{i-1}})}$
3. If $p < 1$, set $\boldsymbol{\theta^i} = \boldsymbol{\theta^*}$ with probability p. Otherwise set $\boldsymbol{\theta_i} = \boldsymbol{\theta^*}$.

A more thorough exposition on this topic can be found in Chib and Greenberg (1995) where they discuss the effects of selecting different candidate distributions. In this paper, we will use the normal distribution as our candidate distribution because it is symmetric and we can conveniently center it at a previous value in our Markov chain.

2.3 Gaussian Process Models

A Gaussian process is a generalization of a Gaussian distribution. Instead of describing random variables, Gaussian processes describe random functions. Gaussian process models are highly flexible statistical models in which a Gaussian process prior is placed on an unknown function. These models are very popular in machine learning because they allow prior knowledge (often in the form of prior data) to help learn an unknown function.

A Gaussian process is uniquely defined by its mean function $\mu(\mathbf{x})$ and covariance kernel $k(\mathbf{x}, \mathbf{x}')$. The mean function controls the location of the function while the covariance kernel controls the local smoothness. The covariance kernel function must be symmetric and produce only positive output. The most common form of covariance kernel is the univariate Euclidean distance function,

$$k(\mathbf{x}, \mathbf{x}') = \frac{e^{-\beta(x-x')}}{\tau},$$

which gives high correlation to values that are close together and low correlation to values that are far apart. Rasmussen and Williams (2006) explore different covariance kernels such as a piecewise polynomial kernel that gives zero correlation to values beyond a certain threshold and a γ-exponential kernel that is similar to the Euclidean distance function but instead measures $|\mathbf{x} - \mathbf{x}'|^\gamma$, making it non-differentiable when $\gamma = 1$. Abrahamsen (1997) examines

more covariance kernels including some that work well with high-dimensional data. We implement the Euclidean distance covariance function in this paper because of its flexibility and simplicity.

Often, the mean function and/or the covariance function can be parameterized by hyperparameters. This lends itself to hierarchical modeling in which prior distributions are specified for each of the hyperparameters. Kennedy et al. (2002) use hierarchical Gaussian process models to incorporate complex computer-simulated data with actual observed data in an effort to better understand hyperparameters in the model. After arriving at a posterior distribution of underlying functions, they present posterior distributions on each of the hyperparameters and then make probability statements about each one. In this paper, we also incorporate hyperparameters. We use them in the covariance function and place priors on each of them so that the data can help learn their correct values.

Gaussian Process Models for Binary Response Data

The most convenient form of data to model with Gaussian process models is normally distributed data. The Gaussian process prior has a conjugate Gaussian process posterior when the likelihood is normal (Nickisch and Rasmussen 2008), which allows for more efficient computational algorithms. In the case of biological threat detector testing, the data is binary classification data, so the reasonable assumed sampling distribution is binomial.

Nickisch and Rasmussen (2008) explore whether the logit link function or the probit link function should be used. They conclude that both behave very similarly and that the main difference is that the logit link has a weaker penalty for misclassification compared to that of the probit. In this paper, a probit link is used for computational simplicity which is described later.

Albert and Chib (1993) propose a latent-variable data augmentation technique that makes for more efficient MCMC convergence. This technique is implemented in this paper. Choudhuri et al. (2007) expand the latent variable data augmentation technique and explore

representing the Gaussian process mean function as an additive model. They perform a simulation study where they show that their model outperforms a local likelihood estimator. We do not treat the mean function as an additive model as we are only working with one covariate.

2.4 Adaptive Design

The main benefit of utilizing an adaptive experimental design is the continual updating of the design based on data and prior experience. The Bayesian paradigm fits this design well because the Bayesian philosophy is rooted in using prior knowledge to make inference. We follow the same approach as Loredo (2003), which is (1) make observations and/or gather data, (2) make inferences from the data using an assumed likelihood and prior, and (3) update the design. These steps are then repeated until the experiment is complete.

The use of adaptive experimental designs has become especially popular in the area of clinical drug trials. When testing drugs, some doses may be more effective than others, and it wastes time and money when patients are given doses that are ineffective. An adaptive design scheme allows for ineffective doses to be detected early so that resources can be spent on the dose levels that produce desired results. Carlin et al. (1997) discuss different decisions that can be made at the interim step. They describe the decision theory behind continuing or stopping an experiment. In this paper we do not stop the experiment until a predetermined number of tests is reached. Stacey and Reese (2007) assign new doses to patients based on a combination of ED95 probabilities and the variance associated in the parameters corresponding to each dose level. The ED95 measures how effective each drug dose is. In our interim step we compare each concentration with the estimated $\widehat{C50}$ and/or $\widehat{C90}$ values. Concentrations that are closer to these estimated values get assigned a higher probability of being tested. We also assign a higher probability of being tested to concentrations that have greater uncertainty in their estimated probability of detection.

Cheng and Shen (2005) propose designs where the number of tests performed is determined from prior information and the data. This allows for clinical drug trials to end early if there is enough evidence of drug effectiveness. In this paper the ultimate number of tests is fixed. We perform a simulation study to determine the desired total number of tests, along with the desired number of tests to introduce at each interim step. We base our conclusions on when the variance in our estimators fail to get significantly smaller.

CHAPTER 3

METHODS

In this chapter we explain in detail the experiment we will be performing. We describe two different models we will compare, a Bayesian logistic regression model and a Bayesian Gaussian process model. We discuss model assumptions and derive the posterior distributions of each model. We find that the Gaussian process model is more computationally intensive yet much more flexible. Lastly, we explain how to implement the adaptive design, and we outline our simulation experiment to compare the different approaches.

3.1 Logistic Model

A common approach to modeling binary data from a Bayesian perspective is using a logistic model. We will compare this model with the Gaussian process model discussed later. Let $\mathbf{y} = (y_1, ..., y_n)$ be the response variable from the detection instruments, where each y_i takes on values of 0 or 1 (success or failure). We will let $\mathbf{x} = (x_1, ..., x_n)$ be the concentration of the biological agent. Our goal is understand the underlying relationship between these two variables, namely $P(Y = 1|X = x)$. Because we are working with binary data, we assume

$$y_i \sim \text{Bernoulli}(p(x_i)),$$

where $p(x_i) \in [0, 1]$ is $P(Y = 1|X = x)$, the unknown parameter of interest. The logistic cdf is used as a link function to relate $p(x_i)$ to a linear function of the threat concentration in the form

$$\log\left(\frac{p(x_i)}{1 - p(x_i)}\right) = \alpha + \beta x_i, \tag{3.1}$$

where $\alpha, \beta \in (-\infty, \infty)$ (Hastie et al. 2001). Thus the probability of detection can be expressed as

$$p(x_i) = \frac{\exp(\alpha + \beta x_i)}{1 + \exp(\alpha + \beta x_i)}, \tag{3.2}$$

which is a function of the threat concentration.

The logistic model has two main benefits. First, it is simple to use and is not computationally expensive. Second, the parameters have interpretable quantities. The parameter α is the probability of detection when concentration is zero or no biological agent is present—namely, the probability of a false alarm. The β parameter is the logged-value for which a change in x accounts for a change in the log-odds of y. The main drawback of this model is illustrated in Figure 3.1. The figure shows different probability curves that are generated with various values of α and β. Note the model's inability to model non-monotone curves. This could be a potential problem in biological weapons testing because sometimes detection instruments get overloaded in the presence of too much threat concentration, resulting in a failed detection. We would want a model that could take into account this possibility. The Gaussian process model described later can handle these situations much more effectively.

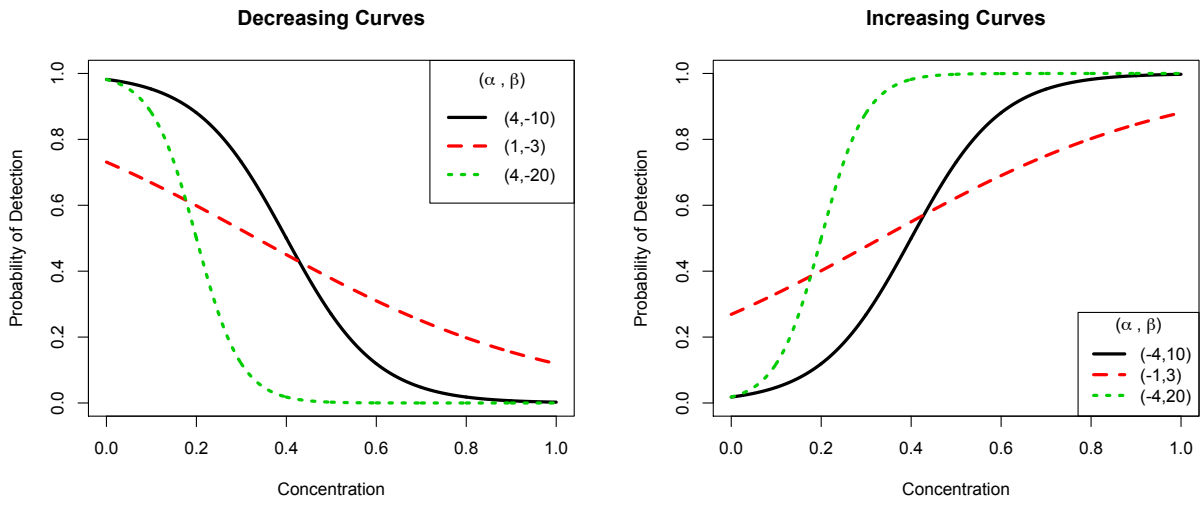

Figure 3.1: Plots of sample logistic model curves for various α and β values. Note the smooth, monotonic nature of the curves.

Likelihood

With the logit link, the likelihood can be represented as

$$f(\mathbf{y}|\alpha,\beta,\mathbf{x}) = \prod_{i=1}^{n}\left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{y_i}\left(\frac{1}{1+\exp(\alpha+\beta x_i)}\right)^{1-y_i}. \tag{3.3}$$

Prior Specification

We choose prior distributions for α and β that have the same support as that of the parameters. Because α and β are not restricted to a specific interval of the real line, it is reasonable to assume they follow normal distributions. The prior distributions for this model are then

$$\alpha \sim \text{Normal}(m_\alpha, s_\alpha^2) \quad \text{and} \quad \beta \sim \text{Normal}(m_\beta, s_\beta^2).$$

We do not know how α and β will interact so we will assume *a priori* independence. We will also use diffuse Normal(0,1000) priors for both α and β to allow for many possible relationships.

Joint Posterior Distribution

Using the likelihood and priors specified above, the joint posterior distribution is

$$\begin{aligned} f(\alpha,\beta|\mathbf{y},\mathbf{x}) &\propto \prod_{i=1}^{n}\left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{y_i}\left(\frac{1}{1+\exp(\alpha+\beta x_i)}\right)^{1-y_i} \\ &\times (2\pi s_\alpha^2)^{-\frac{1}{2}}\exp\left(\frac{-(\alpha-m_\alpha)^2}{2s_\alpha^2}\right) \\ &\times (2\pi s_\alpha^2)^{-\frac{1}{2}}\exp\left(\frac{-(\alpha-m_\alpha)^2}{2s_\alpha^2}\right). \end{aligned} \tag{3.4}$$

Because this unnormalized posterior distribution is an unknown distribution we will need to use a Gibbs-Metropolis algorithm to draw from it. As described previously, a Gibbs sampling algorithm requires that we simultaneously sample from the complete conditional distributions $f(\alpha|\beta,\mathbf{y},\mathbf{x})$ and $f(\beta|\alpha,\mathbf{y},\mathbf{x})$. The complete conditional distributions for α and

β are

$$[\alpha] \propto \prod_{i=1}^{n} \left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)} \right)^{y_i} \left(\frac{1}{1+\exp(\alpha+\beta x_i)} \right)^{1-y_i} \left(\frac{1}{\sqrt{2\pi s_\alpha^2}} \right) \exp\left(\frac{-(\alpha-m_\alpha)^2}{2s_\alpha^2} \right),$$

and

$$[\beta] \propto \prod_{i=1}^{n} \left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)} \right)^{y_i} \left(\frac{1}{1+\exp(\alpha+\beta x_i)} \right)^{1-y_i} \left(\frac{1}{\sqrt{2\pi s_\beta^2}} \right) \exp\left(\frac{-(\beta-m_\beta)^2}{2s_\beta^2} \right).$$

Because both of these distributions are unknown, we must implement the Metropolis-Hastings algorithm within the Gibbs sampler. Once the Gibbs sampler has completed, we are left with draws from the joint posterior distribution of α and β. We discard an appropriate number of draws near the beginning because the algorithm takes some time to correctly draw from the right distributions. Once this burn-in period has been discarded, we can move on to posterior inference.

Posterior Distribution

With each combination of (α, β) from our joint posterior distribution, we can compute a corresponding probability curve. With this distribution of curves we can determine the expected probability of detection for any concentration along with a corresponding 95% credible interval. One of the many benefits of the Bayesian paradigm is the interpretability of credible intervals. In the case of threat detection, a 95% credible interval would allow us to make a direct probability statement about the probability of detection at a certain threat concentration.

In biological threat detection testing, it is often of interest to determine the $C50$ and $C90$ concentrations. These are the threat concentrations that have a probability of 50% and 90% detection respectively. In the logistic model, these quantities can be easily obtained by solving for x_i in our probability function by the equation,

$$x_i = \frac{\log(\frac{p_i}{1-p_i}) - \alpha}{\beta}, \tag{3.5}$$

where p_i is the probability of detection at concentration x_i. Figure 3.2 illustrates how the $C50$ and $C90$ concentrations are calculated for a sample probability curve.

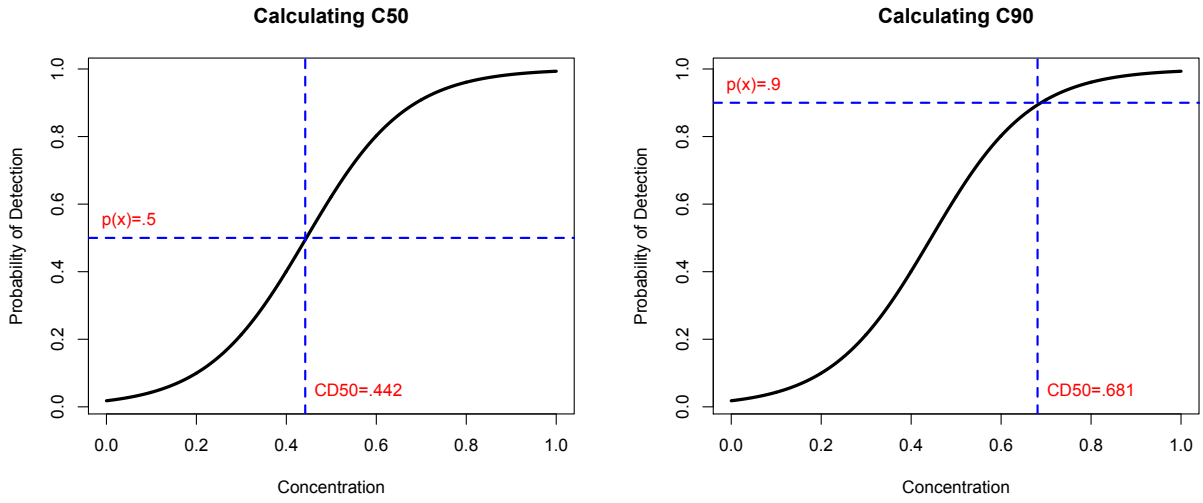

Figure 3.2: Calculating the $C50$ and $C90$ concentrations for a sample probability curve.

3.2 Gaussian Process Model

Instead of using the function $\text{logit}(p(x_i)) = \alpha + \beta x_i$ to relate x_i with $p(x_i)$, we will use some unknown function $\eta(x_i)$. By using $\eta(x_i)$, we are making no assumptions about the form of $\eta(x_i)$; it could be one of infinitely many functions of threat concentration. We will again use a link function to relate $\eta(x_i)$ to $p(x_i)$, but this time we will use the standard normal cdf (probit link). We will illustrate later how the probit link allows us to be more computationally efficient when we are drawing from the posterior distribution of interest.

Gaussian Process Prior

Because the parameter of interest, $\eta(\mathbf{x})$, is a function, a prior distribution must be used that (1) is a distribution over functions and (2) has infinite support. A Gaussian process prior of the form

$$\eta(\mathbf{x}) \sim GP(\mu(\mathbf{x}), \sigma(\mathbf{x}, \mathbf{x}'))$$

fulfills both of these requirements. A Gaussian process is a stochastic process such that any finite number of draws, $(\eta(x_1), \eta(x_2), ..., \eta(x_n))'$, have a multivariate normal distribution

with $E(\eta(x_i)) = \mu(x_i)$ and $cov(\eta(x_i), \eta(x_j)) = \sigma(x_i, x_j)$, for $i, j = 1, ..., n$. The two functional hyperparameters $\mu(\mathbf{x})$ and $\sigma(\mathbf{x}, \mathbf{x}')$ in the Gaussian process prior can be selected to reflect any prior belief about $\eta(\mathbf{x})$. The mean function $\mu(\mathbf{x})$ reflects our prior belief about $\eta(\mathbf{x})$, while the covariance kernel $\sigma(\mathbf{x}, \mathbf{x}')$ reflects our prior belief about the local smoothness of $\eta(\mathbf{x})$.

We will always standardize the $\mathbf{x}$ values so *a priori* we assume $\mu(\mathbf{x}) = \mathbf{0}$. There is a extensive literature on choosing an appropriate covariance kernel, and in this paper we will use the standard Euclidean distance function

$$\sigma(\mathbf{x}, \mathbf{x}') = \frac{\exp(-\gamma(\mathbf{x} - \mathbf{x}')^2)}{\tau}, \tag{3.6}$$

where $\gamma \in (0, \infty)$ and $\tau \in (0, \infty)$. This distance function has a nice interpretation when modeling the results of threat detection tests because it gives higher correlation to values that are closer together and lower correlation to values that are farther apart. This is reasonable because it suggests that if two tests are performed at similar concentrations then their corresponding probabilities of detection should also be similar. The γ and τ values are hyperparameters that govern the magnitude of the correlations obtained from the covariance kernel.

Hyperpriors

We do not know which values γ and τ should take in our covariance kernel so we will place prior distributions on both of them. Because both hyperparameters have strictly positive support, we assume *a priori*

$$\gamma \sim \text{Uniform}(\alpha_\gamma, \beta_\gamma) \text{ and } \tau \sim \text{Gamma}(\alpha_\tau, \beta_\tau),$$

where $\alpha_\gamma, \beta_\gamma, \alpha_\tau, \beta_\tau \in (0, \infty)$. The uniform prior is used for γ because our earlier models suggested that the prior heavily influenced the posterior distribution. We chose the Gamma distribution for τ because it established conjugacy in the model. We do not know how these

hyperparameters interact, so we will assume *a priori* independence. Figure 3.3 displays realizations from the Gaussian process prior using the standard Euclidean distance covariance function. The first plot shows the effect of increasing the value of γ. Larger values of γ bring about smaller correlations between points, which allows for more flexibility. The second panel illustrates how larger values of τ have the opposite effect and bring about higher correlations.

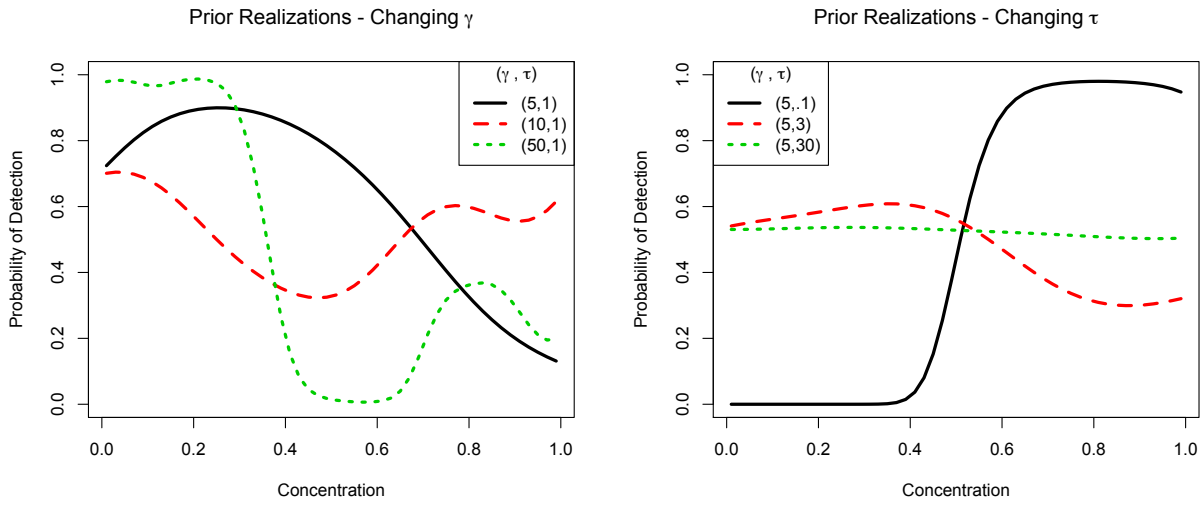

Figure 3.3: Plots of realizations from the Gaussian process prior with varying values of γ and τ. Note that for high values of γ the curves become much more flexible. Also note that for large values of τ the curves become more rigid.

From the plots above, we decided on a Uniform$(0, 20)$ distribution for γ and a Gamma$(.5, 2)$ for τ. The parameters lead to probability curves that are flexible but not overly so.

Data Augmentation for Computational Efficiency

The unnormalized posterior distribution for $\eta(\mathbf{x})$ is

$$\pi(\eta(\mathbf{x})|\gamma, \tau, \mathbf{x}, \mathbf{y}) \propto f(\mathbf{y}|\eta(\mathbf{x}), \gamma, \tau, \mathbf{x})\pi(\eta(\mathbf{x}))\pi(\gamma)\pi(\tau).$$

In order to sample from this unnormalized posterior, we need to determine the complete conditional distributions for $\eta(\mathbf{x}), \gamma$, and τ, and implement a Gibbs-Metropolis sampler. Choudhuri et al. (2007) found that doing so led to very slow movements in the Markov chain because the individual components of $\eta(\mathbf{x})$ are highly correlated and partial conjugacy is not obtained for $\eta(\mathbf{x}), \gamma$, or τ. Albert and Chib (1993) propose augmenting the data set with a latent variable Z_i that establishes partial conjugacy in the model.

Let $\mathbf{z} = (z_1, ..., z_n)$ be unobservable latent variables such that, conditional on $\eta(x_i)$, the z_i's are independent normal random variables with mean $\eta(x_i)$ and unit variance; that is

$$z_i \sim Normal(\eta(x_i), 1).$$

Then, define y_i to be a function of z_i in the following way:

$$y_i = \begin{cases} 1 & \text{if } z_i > 0 \\ 0 & \text{if } z_i \leq 0. \end{cases}$$

This suggests that

$$\begin{aligned} P(y_i = 1) &= P(z_i > 0) \\ &= P\left(\frac{z_i - \eta(x_i)}{1} > \frac{0 - \eta(x_i)}{1}\right) \\ &= \Phi(\eta(x_i)), \end{aligned} \tag{3.7}$$

where Φ is the standard normal cdf. Thus, our y_is are Bernoulli random variables with success probability $\Phi(\eta(x_i))$, which leads to the probit link model.

Joint Posterior Distribution

Since $\mathbf{z}$ is unobservable, yet required for the model, we will sample from the joint posterior distribution of $(\mathbf{z}, \eta(\mathbf{x}), \gamma, \tau)$ and discard $\mathbf{z}$ at the end. With the z_i's, and assuming *a priori*

independence in the priors, the joint posterior density is

$$\begin{aligned}
\pi(\eta(\mathbf{x}), \mathbf{z}|\gamma, \tau, \mathbf{y}) &\propto f(\mathbf{y}|\mathbf{z}, \eta(\mathbf{x}), \gamma, \tau)\pi(\mathbf{z}, \eta(\mathbf{x}), \gamma, \tau) \\
&\propto f(\mathbf{y}|\mathbf{z})\pi(\mathbf{z}|\eta(\mathbf{x}))\pi(\eta(\mathbf{x}))\pi(\gamma)\pi(\tau) \\
&\propto \left[\prod_{i=1}^{n}[I(z_i > 0)(y_i = 1) + I(z_i \leq 0)(y_i = 0)] \times \frac{1}{\sqrt{2\pi}} \exp\left(\frac{(z_i - \eta(x_i))^2}{2}\right)\right] \\
&\times \frac{1}{(\sqrt{2\pi})^n |\mathbf{\Sigma}|^{1/2}} \exp\left(\frac{-\eta(\mathbf{x})'\mathbf{\Sigma}^{-1}\eta(\mathbf{x})}{2}\right) \\
&\times \frac{\beta_\gamma^{\alpha_\gamma}}{\Gamma(\alpha_\gamma)}\gamma^{\alpha_\gamma - 1} \exp\left(-\gamma\beta_\gamma\right) \\
&\times \frac{\beta_\tau^{\alpha_\tau}}{\Gamma(\alpha_\tau)}\tau^{\alpha_\tau - 1} \exp\left(-\tau\beta_\tau\right). \qquad (3.8)
\end{aligned}$$

Complete Conditional Distributions

The complete conditional distributions required for the Gibbs-Metropolis sampler are

$$\begin{aligned}
[\eta(\mathbf{x})] &\sim MVN(\mathbf{Z}'(I + \mathbf{\Sigma}^{-1})^{-1}, (I + \mathbf{\Sigma}^{-1})^{-1}), \qquad (3.9) \\
[z_i] &\sim \begin{cases} N(\eta(x_i), 1)^+ & \text{if } y_i = 1 \\ N(\eta(x_i), 1)^- & \text{if } y_i = 0 \end{cases} \\
[\tau] &\sim Gamma\left(\alpha_\tau + \frac{n}{2}, \frac{1}{2}\eta(\mathbf{x})'\mathbf{\Sigma}_0^{-1}\eta(\mathbf{x}) + \beta_\tau\right), \\
[\gamma] &\propto \gamma^{\alpha_\gamma - 1}|\mathbf{\Sigma}|^{-\frac{1}{2}} \exp\left(-\gamma\beta_\gamma - \frac{1}{2}\eta(\mathbf{x})'\mathbf{\Sigma}^{-1}\eta(\mathbf{x})\right),
\end{aligned}$$

where $\mathbf{\Sigma}_{0(i,j)} = \exp\left(-\gamma(x_i - x_j)^2\right)$ and $N(\eta(x_i), 1)^+$ is a normal distribution truncated at the left by 0 and $N(\eta(x_i), 1)^-$ is a normal distribution truncated at the right by 0.

3.3 Adaptive Trial Design

In this section we will discuss the design piece of our experiment. Instead of using a traditional experimental design, we propose an iterative adaptive framework that works efficiently with our Bayesian model. The main difference between this approach and classical designs is that we do not perform the same number of tests at each concentration. Instead, as more

data is gathered and more information becomes available, we would like to adapt our design to perform a higher number of tests at concentrations in which we are most interested.

Adaptive Allocation

In a classical experimental design, we fix the number of tests, n, performed at a specific number of concentrations, t. This means that at each concentration x_i where $i = 1, ..., t$, the same number of tests are performed despite the fact that we may not be equally interested in all t concentrations. As stated earlier, two of the most important quantities to determine from our experiment are the $C50$ and $C90$ concentrations. Because of this, we would like to use an experimental design piece that allows us to perform more tests around the concentrations that we believe to be the $C50$ and $C90$ concentrations.

In addition to wanting more tests performed at concentrations of interest, we would also like to perform more tests where the uncertainty in the probability of detection is greatest. One of the biggest downfalls of a classical approach is that sometimes after a couple of tests, the uncertainty in the probability of detection at a certain concentration may be very low compared to other concentrations, yet the concentrations are tested equally. We would like to have a design piece that performs more tests where we have less information.

When using an adaptive experimental design, we do not perform all $n \times t$ tests at the same time. Instead, we start off by testing the threat detection instrument a small (and equal) number of times at each of the t concentrations. With these initial results, we implement our Gibbs-Metropolis algorithm to get a posterior distribution of our probability function $p(x)$. Using the mean of this posterior distribution we estimate $\widehat{C50}$ and $\widehat{C90}$ along with the absolute distances

$$d_{50_i} = |\widehat{C50} - x_i| \text{ and } d_{90_i} = |\widehat{C90} - x_i|,$$

for each of the t concentrations. The values d_{50_i} and d_{90_i} give us an indication of how close each concentration is to the true values of $C50$ and $C90$. We also use the posterior distribution of $p(x)$ to compute the standard deviation σ_i of $p(x_i)$ for each concentration.

The next step of the adaptive design is to perform n more tests, add the results to our data set, and refit the model with these new results. In order to test more frequently at concentrations that are closer to the $C50$ and $C90$ values and more frequently where there is greater uncertainty, we will assign a probability to each concentration and construct a mechanism that gives higher probability to performing tests at such concentrations.

We will construct a vector $\mathbf{p} = p_1, ..., p_t$, where p_i is the probability of performing one of the next n tests at concentration x_i. If we define p_i such that

$$p_i = \frac{(1 - d_{50_i} + 1 - d_{90_i})\sigma_i}{\sum_{i=1}^{t}(1 - d_{50_i} + 1 - d_{90_i})\sigma_i} \qquad i = 1, ..., t, \tag{3.10}$$

then each p_i is a probability between 0 and 1. This method of assigning probabilities gives higher probabilities to concentrations that are closer in absolute value to $\widehat{C50}$ and $\widehat{C90}$ and to concentrations with higher variances associated with them. If we wanted to use only d_{50} or d_{90} as our criteria in determining where to perform the next tests, then we would simply use

$$p_i = \frac{(1 - d_i)\sigma_i}{\sum_{i=1}^{t}(1 - d_i)\sigma_i} \qquad i = 1, ..., t \tag{3.11}$$

instead, where $d_i = d_{50_i}$ or d_{90_i}. With this $\mathbf{p}$ vector, we can then sample with replacement the concentrations $x_1, ..., x_t$, where each x_i corresponds to a specific p_i. This will give us a selection of concentrations for performing the next n tests. This process is then repeated for each of the next batches of tests that need to be performed.

3.4 Simulation Study

In order to determine which model is the most effective, we will perform a simulation study. We will generate data from four known probability curves and then for each curve we will compare the performance of the following four models: a standard logistic model with a fixed number of tests, a standard logistic model implementing the adaptive design, our Gaussian process model with a fixed number of tests, and our Gaussian process model implementing the adaptive design. This way we can compare both the logistic model versus the Gaussian

process model and the classical design versus the adaptive design. We will also vary the total number of tests, N, that will be performed for each model at $N = 24, 72$, and 120. The six different concentrations at which the tests will be performed are

$$\begin{aligned} \mathbf{x} &= (10, 25, 50, 100, 500, 1000) \\ &= (x_{10}, x_{25}, x_{50}, x_{100}, x_{500}, x_{1000}). \end{aligned}$$

Our MCMC algorithms will be performed with 50,000 iterations, each with a burn-in period of 5,000. The measure by which we will compare the four models is the bias in the $\widehat{C50}$ estimators along with the overall variability of the posterior probability curves.

The four probability curves that will be used in the simulation study are illustrated in Figure 3.4. Each curve represents a possible relationship between threat concentration and detection probability. The black non-monotonic curve shows the possibility of the detection instrument getting overloaded at higher concentration levels. The blue flat curve represents a null case in which there is no relationship between concentration and probability of detection. The green s-shaped curve represents the situation in which the probability of detection increases quickly and levels off at higher concentrations. Lastly, the red slow increasing curve depicts the case in which the detection probability gradually increases with concentration.

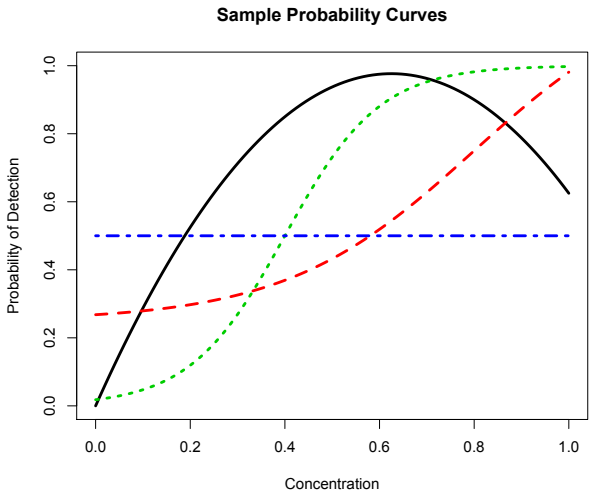

Figure 3.4: Four probability curves that will be used in the simulation study.

CHAPTER 4

RESULTS

4.1 Posterior Distributions

The following three sections provide plots of the posterior distributions with the three sample sizes.

Small Sample Size (24)

Figure 4.1 displays plots of the posterior distributions for the smallest sample size. For the non-monotone curve, the Gaussian process adaptive model clearly did a better job modeling the underlying curve. It also had the lowest MSE when estimating $\widehat{C50}$ (for plots of MSE across sample sizes, refer to section 4.2). With the slowly increasing curve, the Gaussian process non-adaptive model performed best both in terms of estimating $\widehat{C50}$ and getting at the shape of the curve. In the sharply increasing curve, the two logistic models modeled the underlying curve best with the Gaussian process non-adaptive model having the lowest MSE in $\widehat{C50}$. Because there were no data points between .5 and 1, the Gaussian process models had a hard time modeling that region and tried to go straight to the last point. In the case of the null curve, the logistic non-adaptive model did a nearly perfect job at modeling the curve; the others did a fair job.

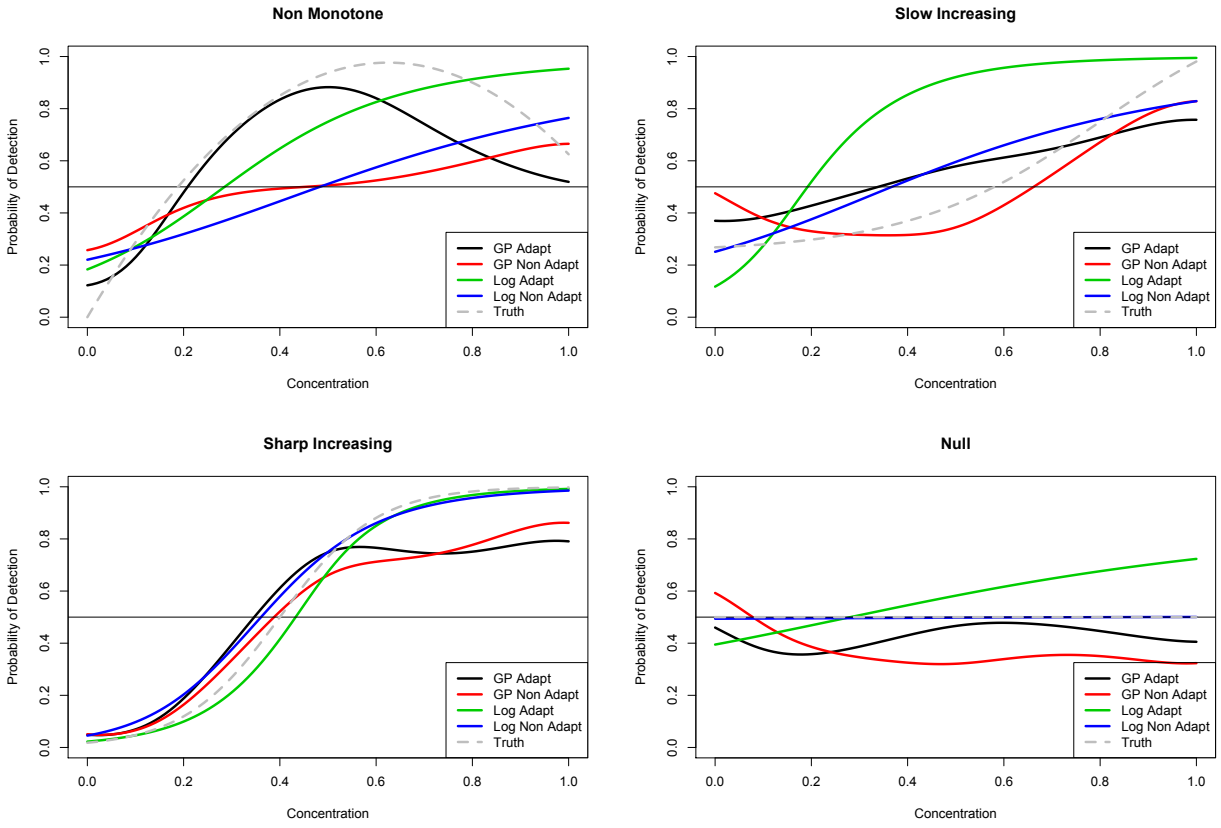

Figure 4.1: Plots of Mean Posterior Curves Across Functions for Small Sample Size

Table 4.1 displays the number of tests that were performed at each concentration. For the non-monotone curve, the true $C50$ value is 180.4. These results show that the Gaussian process model performed the greatest number of tests at the value closest to $C50$, x_{100}, with the second highest number of tests performed at x_{500}. A similar result occurred for the logistic model except that the greatest number of tests were performed at x_{500}. This is likely due to the difficulty the logistic model had in estimating $\widehat{C50}$ in the non-monotone curve.

The true $C50$ value for the slowly increasing curve is 580.3. Note that the Gaussian process model performed the greatest number of tests closest to the value x_{500} (tied with x_{10} and x_{50}). The logistic model did not perform the most tests around $C50$. This again is likely due to the difficulty that the logistic model had in modeling the slowly increasing probability curve.

The sharply increasing curve was the easiest curve to model and estimate $\widehat{C50}$. Consequently, both models performed the highest number of tests at x_{500} which is the closest concentration to the $C50 = 400$ value.

The null curve did not have a true $C50$ value since it had 50% probability of detection across the entire curve. Because of that, we expected the number of tests performed to be roughly uniform across the concentrations. Table 4.1 suggests that this is the case.

Table 4.1: Number of Tests Performed for Small Sample Size. Each cell contains the number of tests that were performed at each concentration.

Model	Concentration					
	10	25	50	100	500	1000
GP Adaptive Non-monotone	4	2	3	7	6	2
GP Adaptive Slow Increasing	5	4	5	3	5	2
GP Adaptive Sharp Increasing	4	3	6	3	6	2
GP Adaptive Null	3	5	4	5	4	3
Logistic Adaptive Non-monotone	5	3	2	4	8	2
Logistic Adaptive Slow Increasing	3	6	3	6	4	2
Logistic Adaptive Sharp Increasing	3	2	4	2	10	3
Logistic Adaptive Null	4	4	3	5	3	5
Non-adaptive	4	4	4	4	4	4

Medium Sample Size (72)

The posterior distributions for the medium sample size (Figure 4.2) illustrated a similar story to that of the small sample size. With more data, the Gaussian process non-adaptive and logistic adaptive models did a much better job at modeling the non-monotone curve. While neither of them performed quite as well as the Gaussian process adaptive model, the Gaussian process non-adaptive performed equally well at estimating $\widehat{C50}$. For the slowly

increasing curve, the Gaussian process models did much better than the logistic models both in terms of the MSE in $\widehat{C50}$ and overall shape. The logistic models had a hard time getting the slowly increasing aspect of the curve. In the sharply increasing curve, again, the logistic models did better at modeling the shape of the curve while the Gaussian process non-adaptive model was near perfect at estimating $\widehat{C50}$. With the null curve, the logistic non-adaptive again performed the best.

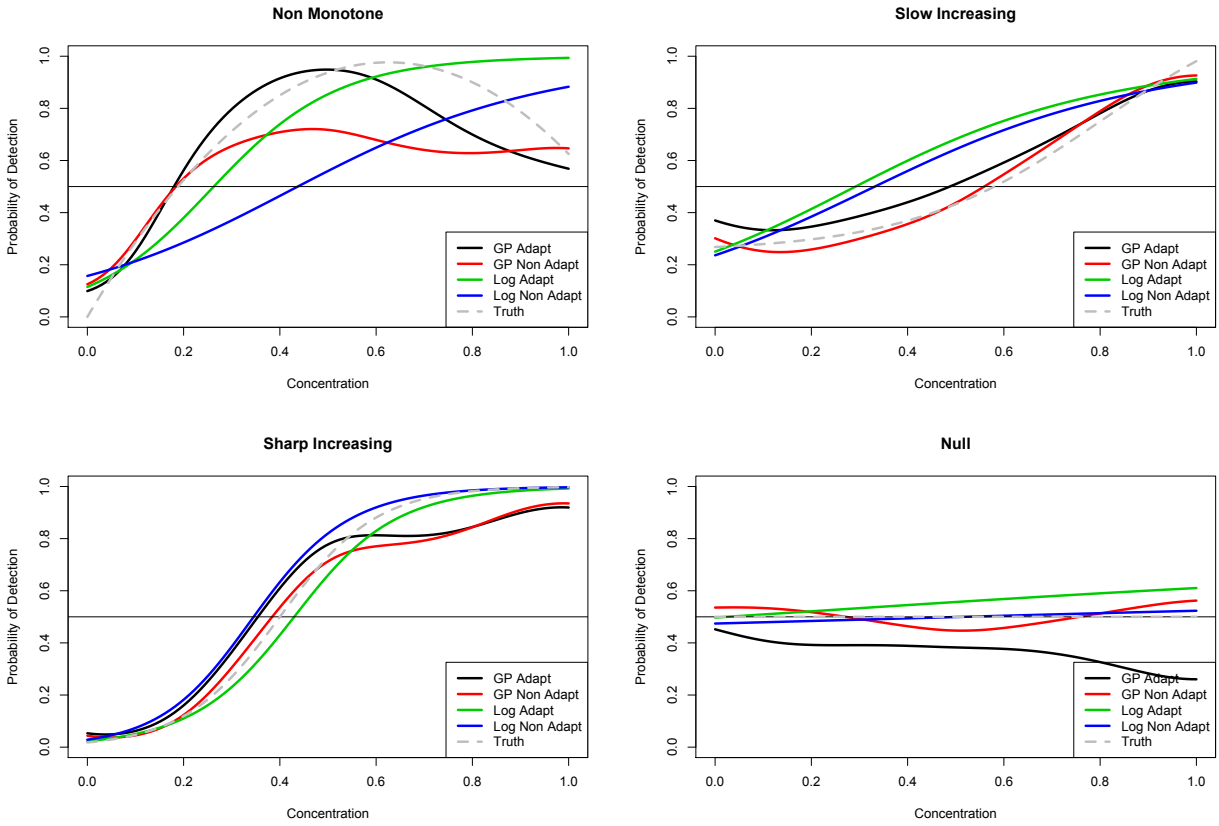

Figure 4.2: Plots of Mean Posterior Curves Across Functions for Medium Sample Size

Table 4.2 displays the number of tests that were performed at each concentration for the medium-sized data set. The Gaussian process model consistently performed more tests around the $C50$ concentration in all of the curves. Note that the logistic model did not perform the highest number of tests around the $C50$ value in the non-monotone curve. This is again likely due to the fact that the logistic model had a hard time modeling the

true underlying curve. For the slowly and sharply increasing curves, the logistic model did perform more tests around the true $C50$ with an extremely high number of tests performed at x_{500}.

Table 4.2: Number of Tests Performed for Medium Sample Size. Each cell contains the number of tests that were performed at each concentration.

Model	Concentration					
	10	25	50	100	500	1000
GP Adaptive Non-monotone	14	5	9	20	17	7
GP Adaptive Slow Increasing	13	11	12	7	20	9
GP Adaptive Sharp Increasing	11	6	11	10	25	9
GP Adaptive Null	18	11	11	13	13	6
Logistic Adaptive Non-monotone	17	8	11	14	20	2
Logistic Adaptive Slow Increasing	12	12	13	15	18	2
Logistic Adaptive Sharp Increasing	8	4	8	15	33	4
Logistic Adaptive Null	14	11	7	12	15	13
Non-adaptive	12	12	12	12	12	12

Large Sample Size (120)

Figure 4.3 displays the results of the simulation study for the largest sample size. With the non-monotone curve, the logistic models again had a difficult time capturing the underlying curve. The Gaussian process models did equally well in estimating $\widehat{C50}$ with the Gaussian process model doing better at modeling the height of the curve. The plot of the slowly increasing curve tells a similar story. Both Gaussian process models again did about the same in estimating $\widehat{C50}$ with the Gaussian process non-adaptive doing slightly better at modeling the entire curve. For the sharply increasing curve, there was no difference between the two Gaussian process models; both doing equally well at estimating $\widehat{C50}$. For the logistic

models, the adaptive model did quite a bit better at estimating $\widehat{C50}$ and modeling the entire curve. In the case of the null curve, there does not appear to be a big difference between the four curves.

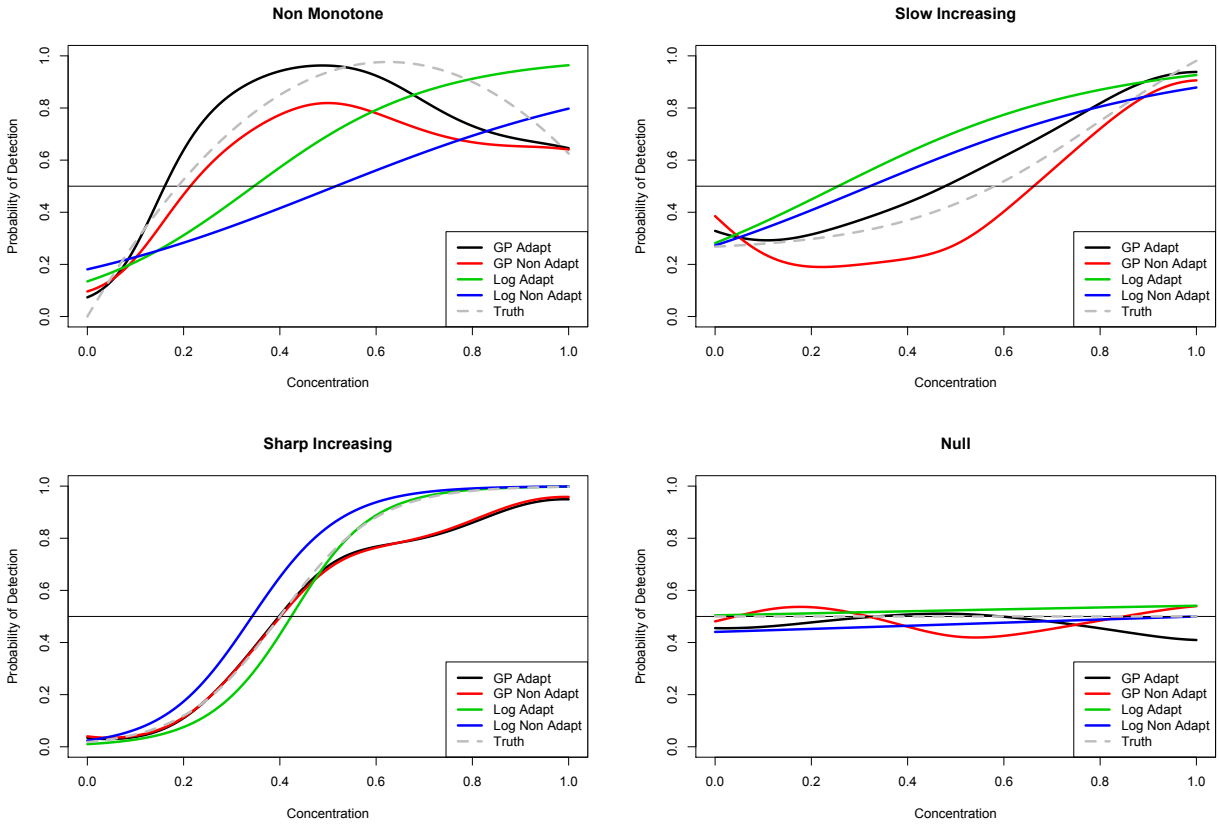

Figure 4.3: Plots of Mean Posterior Curves Across Functions for Large Sample Size

For the large data set, Table 4.3 illustrates that the Gaussian process model again performed the most number of tests at concentrations close to the true $C50$. Note also the low number of tests performed at the x_{25} concentration. This is likely because x_{10} and x_{50} are right below and above x_{25} which decreases the uncertainty in the probability curve at that point. As the adaptive experimental design gives lower probability to tests being performed at concentrations that have lower variance in the probability curve, x_{25} was not tested as frequently.

The logistic model again did not perform too many tests around the true $C50$ value in the non-monotone curve but it did in the slowly and sharply increasing curves. It is interesting to note that the logistic model very rarely performed tests at x_{1000}. This is likely due to the nature of the logistic model. Because it is strictly increasing, once it reaches its peak, it stays there and the uncertainty in the probability curve gets very small. The Gaussian process model, however, does not have that limitation.

Table 4.3: Number of Tests Performed for Large Sample Size. Each cell contains the number of tests that were performed at each concentration.

Model	Concentration					
	10	25	50	100	500	1000
GP Adaptive Non-monotone	23	10	17	38	23	9
GP Adaptive Slow Increasing	23	14	19	12	37	15
GP Adaptive Sharp Increasing	15	12	18	16	43	16
GP Adaptive Null	24	20	21	21	21	13
Logistic Adaptive Non-monotone	24	17	18	23	32	6
Logistic Adaptive Slow Increasing	23	20	17	23	29	8
Logistic Adaptive Sharp Increasing	15	8	14	21	56	6
Logistic Adaptive Null	24	28	14	16	25	13
Non-adaptive	20	20	20	20	20	20

4.2 Mean Squared Error of $\widehat{C50}$

The three plots in Figure 4.4 display the MSE of $\widehat{C50}$ across sample size. For the non-monotone curve, the Gaussian process non-adaptive model did uniformly best across all sample sizes. The Gaussian process adaptive model did equally well for the medium and large samples. For the slowly increasing curve, the Gaussian process models had the lowest MSE with the non-adaptive model doing slightly better in the smallest sample. For the

logistic models, the non-adaptive model also had a smaller MSE. All four models had a hard time estimating $\widehat{C50}$ with this probability curve. In the sharply increasing curve, all four models did a good job estimating $\widehat{C50}$. The Gaussian process non-adaptive model did slightly better than the other three models for the small and medium sample sizes with the Gaussian process adaptive model doing better in the largest sample size.

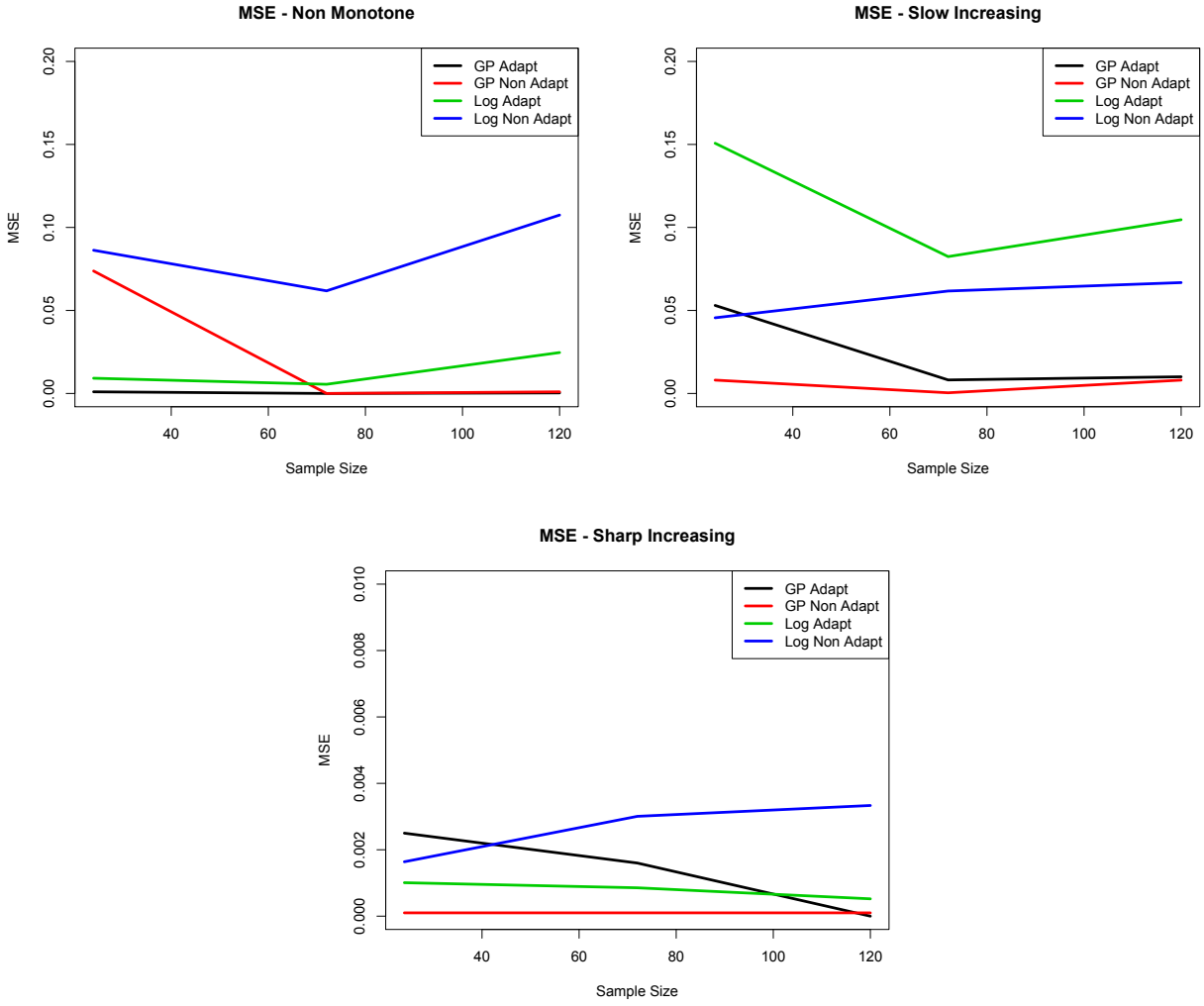

Figure 4.4: MSE of $\widehat{C50}$

4.3 Posterior Variances

At Observed Points

Figure 4.5 displays plots of the mean variances of the posterior distributions at the observed data points. Notice for the non-monotone curve, the Gaussian process non-adaptive model did uniformly better than the other three. In the rest of the plots, the logistic adaptive model had the lowest mean variance. Note that in all four cases, the Gaussian process adaptive model has a mean variance equal to or lower than the Gaussian process non-adaptive model. The same results hold for the logistic models. In all four cases, the mean variance for the logistic adaptive model was always lower than that of the logistic non-adaptive. This is likely due to the fact that the adaptive design was built to perform more tests at observations that have less variance in the probability curve.

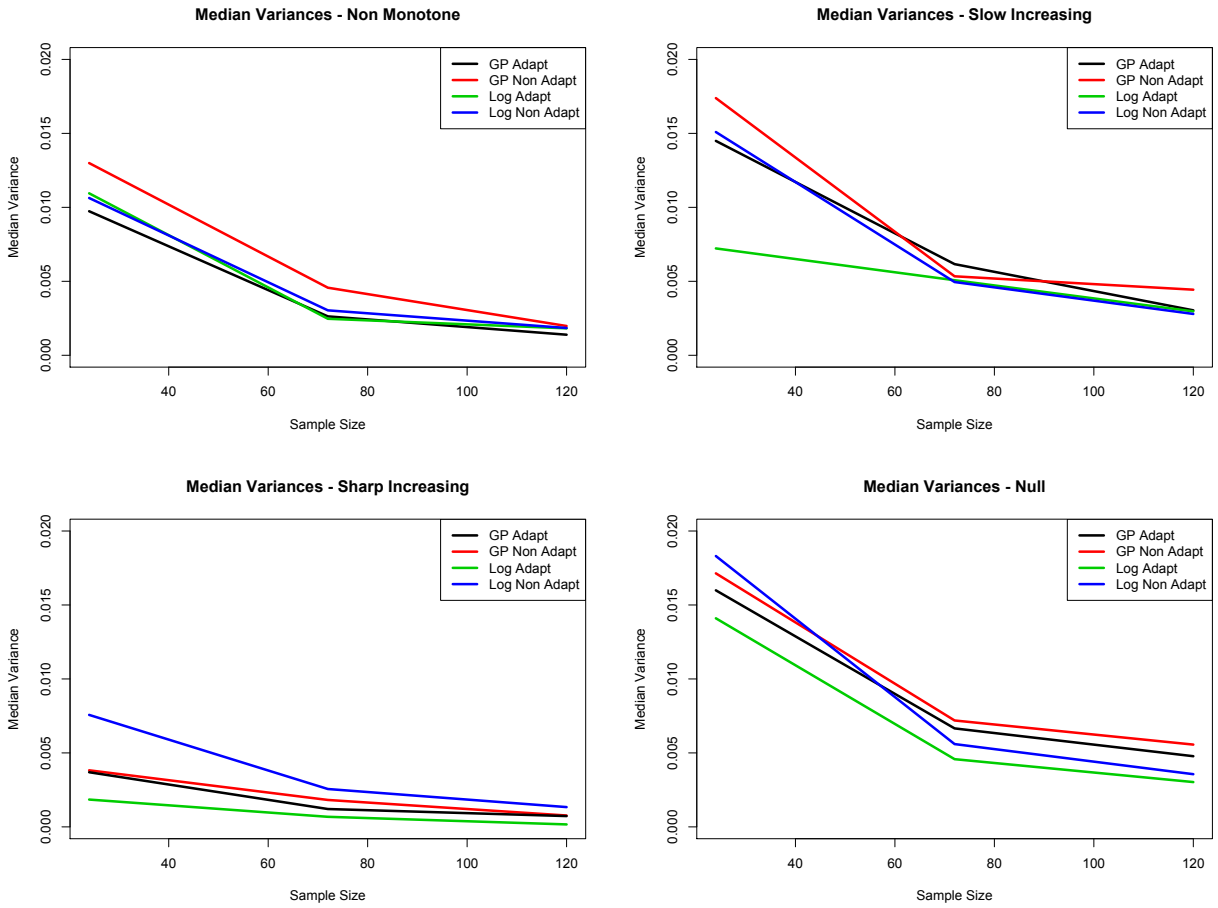

Figure 4.5: Variance of the Posterior Distribution at Observed Points

The mean variance in the posterior probability curves at unobserved points tells a similar story. All four plots in Figure 4.6 illustrate the fact that between observed points, the logistic model has less variance. This is unsurprising as the Gaussian process models are designed to do much better at observed points compared to between points. Note again that in all four plots the adaptive designs always (with the exception of the small sample size for the Gaussian process models for the non-monotone curve) out performs the non-adaptive models. Also, note that for the Gaussian process models, the adaptive design did considerably better with larger sample sizes. The same cannot be said for the logistic models. As the data set grew, the logistic adaptive and non-adaptive models did about the same.

Between Observed Points

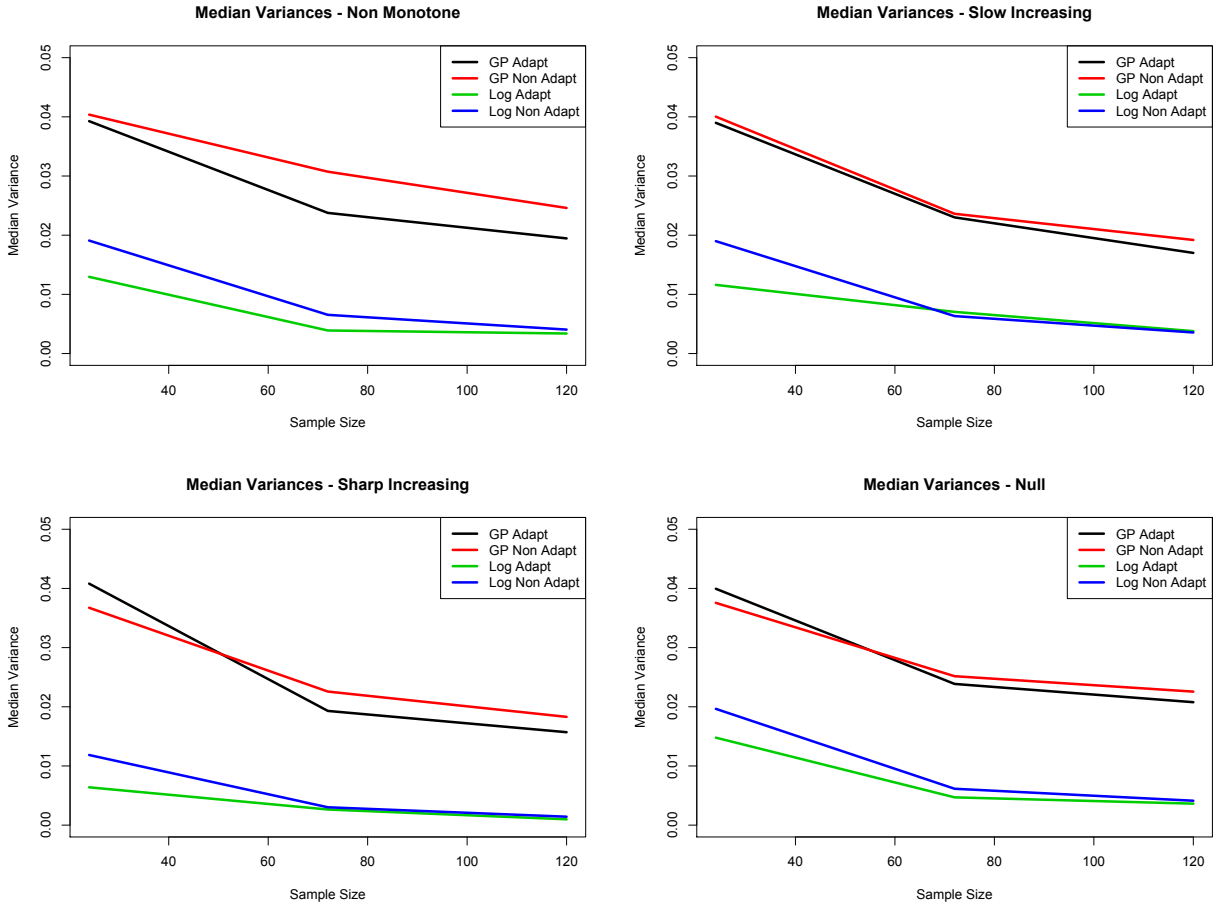

Figure 4.6: Variance of the Posterior Distribution at Unobserved Points

CHAPTER 5

CONCLUSIONS

Threat detector testing is costly and time consuming and should be performed as efficiently as possible. We looked at four different approaches to this problem and compared their performances via a simulation study. In our simulation study, we ran two different models with two different experimental designs to compare model/design performance. We ran each of the four model/design combinations at three different samples sizes across four different probability curves and found that the Gaussian process model combined with the adaptive experimental design was overall the most useful combination.

Our results suggest that if the underlying probability curve is believed to be non-standard (non-monotone or slowly increasing) then the Gaussian process model is a better choice. In the simulation study, both the adaptive and non-adaptive Gaussian process models were better at modeling the probability curve when it was non-monotone or slowly increasing. Conversely, the monotonic nature of the logistic models hindered their ability to model curves that were not s-shaped. While it is true that the overall variance in the posterior probability curve was lower for the logistic models, they did too poor a job at modeling the underlying curve to be considered better than the Gaussian process models.

There wasn't a strong difference between the adaptive and non-adaptive Gaussian process models in terms of estimating $\widehat{C50}$ and modeling the entire curve. In large sample sizes, they either performed the same or the adaptive model did better. In smaller sample sizes, it depended on the probability curve that was being modeled — the adaptive model outperforming in the non-monotone case and the non-adaptive outperforming in the slowly increasing case. Both did equally well for the sharp increasing and null curves. The biggest difference between the two models was the uncertainty in their posterior probability curves.

The Gaussian process adaptive model essentially always had a lower variance both at the observed points and between the observed points. This is due to the fact that the adaptive experimental design is designed not to waste tests at concentrations that are already well understood from the data. Instead, tests were often performed were there was greater uncertainty in the probability curve.

If the underlying probability curve is believed to be a traditional s-shaped curve, then our findings suggest that the logistic model is the better choice, especially at small sample sizes. With only 24 observations, both logistic models had a good grasp on the underlying curve and both got better as the number of observations increased. Also, uncertainty in the curve was quite a bit lower in the logistic models compared to the Gaussian process models.

Much like the Gaussian process models, the adaptive experimental design was the better design scheme for the logistic models. In most of the cases, the adaptive logistic model outperformed the non-adaptive in terms of estimating $\widehat{C50}$ and modeling the entire curve. The the only exception to this was the slowly increasing curve and possibly the null curve at smaller sample sizes. In terms of the variance in the estimated curves, the adaptive logistic model was always the best, both at observed points and at unobserved points. Again, this is due to the fact that the adaptive design scheme is less likely to perform tests at concentrations where there is lower uncertainty in the probability curve.

If nothing is known about the underlying probability curve, the model of choice would be the Gaussian process model with the adaptive experimental design. It performed tests at concentrations we were most interested in and was able to model a wider variety of curves. Also, while the logistic model does outperform the Gaussian process model when the underlying curve is s-shaped, the cost of using a Gaussian process model is not much. It may have more variance but it appears to do equally well at estimating $\widehat{C50}$.

5.1 Future Work

In the future, it would be interesting to compare the four model/design combinations in a space filling design. Our study used only 6 different threat concentrations that were not equally spaced. The results may differ if there were more concentrations being considered and/or if they were equally spaced apart. From what we have seen thus far, the Gaussian process models would likely be even more powerful because the variance between the observed concentrations gets drastically reduced when there are not large gaps between the data points.

Another aspect of this study that would be interesting to examine further would be the choice of criteria that is used in the adaptive design steps. Our criteria involved only the variance in the posterior probability curve and the absolute proximity to $\widehat{C50}$. It would be interesting to look at different sets of criteria when deciding where to assign the next test. Maybe a Gaussian kernel should be used to get a measure of closeness between $\widehat{C50}$ and the probability curve at each data point. Maybe the standard deviation or a 95% credible interval should be used instead of the variance when quantifying the uncertainty in the posterior probability curve.

BIBLIOGRAPHY

Abrahamsen, P. (1997), *A Review of Gaussian Random Fields and Correlation Functions*, Oslo, Norway: Norwegian Computing Center.

Albert, J., and Chib, S. (1993), "Bayesian Analysis of Binary and Polychotomous Response Data," *Journal of the American Statistical Association*, 88, 669–679.

Anscombe, F. J. (1964), "Normal likelihood functions," *Annals of the Institute of Statistical Mathematics*, 26, 1–19.

Borelli, J. V. (2007), *Bioterrorism: Prevention, Preparedness And Protection*, New York: Nova Science Publishers.

Carlin, B. P., Kadane, J. B., and Gelfand, A. E. (1997), "Approaches for Optimal Sequential Decision Analysis in Clinical Trials," *Biometrics*, 54, 964–975.

Carlin, B. P., and Louis, T. A. (2009), *Bayesian Methods for Data Analysis* (3rd ed.), Boca Raton, FL: Chapman & Hall/CRC.

Cheng, Y., and Shen, Y. (2005), "Bayesian Adaptive Designs for Clinical Trials," *Biometrika*, 92, 633–646.

Chib, S., and Greenberg, E. (1995), "Understanding the Metropolis-Hastings Algorithm," *The American Statistician*, 49, 327–335.

Choudhuri, N., Ghosal, S., and Roy, A. (2007), "Nonparametric binary regression using a Gaussian process prior," *Statistical Methodology*, 4, 227–243.

Hamada, M. S., Wilson, A. G., Reese, C. S., and Martz, H. F. (2008), *Bayesian Reliability*, New York: Springer Science+Business Media, LLC.

Hastie, T., Tibshirani, R., and Friedman, J. (2001), *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, New York: Springer Science+Business Media, LLC.

Kennedy, M. C., O'Hagan, A., and Higgins, N. (2002), "Bayesian Analysis of Computer Code Outputs," *Quantitative Methods for Current Environmental Issues*, 1, 227–243.

Loredo, T. J. (2003), "Bayesian Adaptive Exploration in a Nutshell," *Phystat*, 1, 162–165.

Nickisch, H., and Rasmussen, C. E. (2008), "Approximations for Binary Gaussian Process Classification," *Journal of Machine Learning Research*, 9, 2035–2078.

Rasmussen, C. E., and Williams, C. K. I. (2006), "Gaussian Processes for Machine Learning," *The American Statistician*, 49, 327–335.

Smith, A. F. M., and Roberts, G. O. (2003), "Bayesian Computation Via the Gibbs Sampler and Related Markov Chain Monte Carlo Methods," *Journal of the Royal Statistical Society*, 55, 3–23.

Stacey, A., and Reese, C. S. (2007), "An Adaptive Bayesian Approach to Bernoulli-Response Clinical Trials," *Master's Thesis, Brigham Young University*, 1, 1.

APPENDICES

APPENDIX A

COMPLETE CONDITIONAL DISTRIBUTIONS

A.1 Gaussian Process Model

Complete Conditional for $\eta(\boldsymbol{x})$

$$\begin{aligned}
\pi(\eta(\mathbf{x})|\mathbf{z},\mathbf{x}) &\propto \frac{1}{(2\pi)^{\frac{k}{2}}(|\Sigma|)^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\frac{1}{(2\pi)^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\mathbf{z}-\eta(\mathbf{x}))'(\mathbf{z}-\eta(\mathbf{x}))\right)\\
&\propto \exp\left(-\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\exp\left(-\frac{1}{2}(\mathbf{z}-\eta(\mathbf{x}))'(\mathbf{z}-\eta(\mathbf{x}))\right)\\
&= \exp\left(-\frac{1}{2}[\mathbf{z}'\mathbf{z}-2\mathbf{z}'\eta(\mathbf{x})+\eta(\mathbf{x})'\eta(\mathbf{x})+\eta(\mathbf{x})'\Sigma^{-1}\eta(\mathbf{x})]\right)\\
&\propto \exp\left(-\frac{1}{2}[-2\mathbf{z}'\eta(\mathbf{x})+\eta(\mathbf{x})'\eta(\mathbf{x})+\eta(\mathbf{x})'\Sigma^{-1}\eta(\mathbf{x})]\right)\\
&= \exp\left(-\frac{1}{2}[\eta(\mathbf{x})'(I+\Sigma^{-1})\eta(\mathbf{x})-2\mathbf{z}'\eta(\mathbf{x})]\right)\\
&= \exp\left(-\frac{1}{2}[\eta(\mathbf{x})'(I+\Sigma^{-1})\eta(\mathbf{x})-2\mathbf{z}'(I+\Sigma^{-1})^{-1}(I+\Sigma^{-1})\eta(\mathbf{x})]\right)
\end{aligned}$$

Define

$$\begin{aligned}
\mu^* &= \mathbf{z}'(I+\Sigma^{-1})^{-1}\\
\Sigma^* &= (I+\Sigma^{-1})^{-1}
\end{aligned}$$

thus

$$\begin{aligned}
\pi(\eta(\mathbf{x})|\mathbf{z},\mathbf{x}) &\propto \exp\left(-\frac{1}{2}[\eta(\mathbf{x})'\Sigma^{*-1}\eta(\mathbf{x})-2\mu^*\Sigma^{*-1}\eta(\mathbf{x})]\right)\\
&\propto \exp\left(-\frac{1}{2}[\eta(\mathbf{x})'\Sigma^{*-1}\eta(\mathbf{x})-2\mu^*\Sigma^{*-1}\eta(\mathbf{x})+\mu^*\Sigma^{*-1}\mu^*]\right)\\
&= \exp\left(-\frac{1}{2}[(\eta(\mathbf{x})-\mu^*)'\Sigma^{*-1}(\eta(\mathbf{x})-\mu^*)]\right)\\
&\sim MVN(\mu^*,\Sigma^*) \qquad \text{(A. 1)}
\end{aligned}$$

Complete Conditional for τ

$$\begin{aligned}\pi(\tau|\eta(\mathbf{x}),\mathbf{x}) &\propto \frac{1}{\Gamma(\alpha_\tau)\beta_\tau^{\alpha_\tau}}\tau^{\alpha_\tau-1}\exp(\tau\beta_\tau)\frac{1}{(2\pi)^{\frac{k}{2}}(|\Sigma|)^{\frac{1}{2}}}\exp\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\\ &\propto \tau^{\alpha_\tau-1}\exp(\tau\beta_\tau)(|\Sigma|)^{-\frac{1}{2}}\exp\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\end{aligned}$$

Define

$$\Sigma_0 = \exp\left(-\gamma(\mathbf{x}-\mathbf{x}')^2\right)$$

$$\begin{aligned}&\propto \tau^{\alpha_\tau+\frac{k}{2}-1}\exp -\tau\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma_0^{-1}(\eta(\mathbf{x}))+\beta_\tau\right)\\ &\sim \text{Gamma}\left(\alpha_\tau+\frac{k}{2},\frac{1}{2}(\eta(\mathbf{x}))'\Sigma_0^{-1}(\eta(\mathbf{x}))+\beta_\tau\right)\end{aligned} \tag{A. 2}$$

Complete Conditional for γ

$$\begin{aligned}\pi(\gamma|\eta(\mathbf{x}),\mathbf{x}) &\propto \frac{1}{b-a}\frac{1}{(2\pi)^{\frac{k}{2}}(|\Sigma|)^{\frac{1}{2}}}\exp\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\\ &\propto (|\Sigma|)^{-\frac{1}{2}}\exp\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\\ &\propto (|\Sigma_0|)^{-\frac{1}{2}}\exp\left(\frac{1}{2}(\eta(\mathbf{x}))'\Sigma^{-1}(\eta(\mathbf{x}))\right)\end{aligned} \tag{A. 3}$$

Complete Conditional for z_i

$$\begin{aligned}\pi(z_i|\eta(x_i),x_i) &\propto \left[\prod_{i=1}^{n} I(z_i>0)(y_i=1)+I(z_i\leq 0)(y_i=0)\times\frac{1}{\sqrt{2\pi}}\exp\left(\frac{(z_i-\eta(x_i))^2}{2}\right)\right]\\ &\sim \begin{cases} N(\eta(x_i),1)^+ & \text{if } y_i=1\\ N(\eta(x_i),1)^- & \text{if } y_i=0\end{cases}\end{aligned} \tag{A. 4}$$

A.2 Logistic Model

Complete Conditional for α

$$\begin{aligned}
\pi(\alpha|\mathbf{x},\mathbf{y}) &\propto \left[\prod_{i=1}^{m}\left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{y_i}\left(1-\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{1-y_i}\right] \\
&\times \frac{1}{(2\pi s_\alpha^2)}\exp\left(\frac{-(\alpha-m_\alpha)^2}{2s_\alpha^2}\right) \\
&\propto \left[\prod_{i=1}^{m}(\exp(\alpha+\beta x_i))^{y_i}\left(\frac{1}{1+\exp(\alpha+\beta x_i)}\right)\right] \\
&\times \frac{1}{(2\pi s_\alpha^2)}\exp\left(\frac{-(\alpha-m_\alpha)^2}{2s_\alpha^2}\right) \qquad \text{(A. 5)}
\end{aligned}$$

Complete Conditional for β

$$\begin{aligned}
\pi(\alpha|\mathbf{x},\mathbf{y}) &\propto \left[\prod_{i=1}^{m}\left(\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{y_i}\left(1-\frac{\exp(\alpha+\beta x_i)}{1+\exp(\alpha+\beta x_i)}\right)^{1-y_i}\right] \\
&\times \frac{1}{(2\pi s_\beta^2)}\exp\left(\frac{-(\beta-m_\beta)^2}{2s_\beta^2}\right) \\
&\propto \left[\prod_{i=1}^{m}(\exp(\alpha+\beta x_i))^{y_i}\left(\frac{1}{1+\exp(\alpha+\beta x_i)}\right)\right] \\
&\times \frac{1}{(2\pi s_\beta^2)}\exp\left(\frac{-(\beta-m_\beta)^2}{2s_\beta^2}\right) \qquad \text{(A. 6)}
\end{aligned}$$

APPENDIX B

CODE

B.1 Simulation

```
#####################################################################
#######################  FINAL EXPERIMENT ###########################
# Bradley Ferguson
# Masters Project

#####################################################################
#####################################################################
# Comparing Gaussian Process Adaptive vs Non-adaptive and Logistic model
   Adaptive vs. Non-Adaptive
# Doing three sample sizes, 24, 72, and 120 (small, med, large)
# Generating data from 4 probability curves, 1-Non-monotone, 2-Slow
   increasing, 3-Shark increasing, 4-Null
# Running the model at 50000 MCMC iterations with a burnin period of 5000
#####################################################################
#####################################################################

##################
### Libraries ###
##################
library(MASS)
library(msm)

#############################
### Probability Functions ###
#############################

fx1 = function(x){
   out = 2.5*x*(1.25-x)
   return(out)
}
fx2 = function(x){exp(-4+5*x)/(1 + exp(-4+5*x))+.25} # slow increase
fx3 = function(x){exp(-4+10*x)/(1 + exp(-4+10*x))} # sharp increase
fx4 = function(x){.5+0*x} # null

############################
### Covariance Functions ###
############################

corMatrixFast = function(x,gamma,tau){
dd = dist(x,upper=T,diag=T)^2
corMat = exp(-gamma*as.matrix(dd)) / tau
return(corMat)
```

```
}

corMatrix = function(x1,x2,gamma,tau){ # function that creates correlation
    matrix
l1 = length(x1)
l2 = length(x2)
corMat = matrix(nrow=l1,ncol=l2)
for(i in 1:l2){
   for(j in 1:l1){
         corMat[j,i] = exp((-gamma*(x1[j]-x2[i])^2))/tau
      }
}
   return(corMat)
}

################################
### Error Checking Function ###
################################

errorCheck = function(Sigma){

   count = 0
   psi = 10^-4
   n = dim(Sigma)[1]
   inverse = try(solve(Sigma),TRUE) # checking for singularity
   while(inherits(inverse,"try-error") || det(Sigma)==0){
      Sigma = Sigma + diag(n)*psi
      inverse = try(solve(Sigma),TRUE)
      count = count + 1
   }
   sigList = list(Sigma=Sigma,psi=count*psi)
   return(sigList)
}

########################
### Initial Settings ###
########################

MCMCiter = 50000
xvals = c(10,25,50,100,500,1000)
numRounds = 19
numTests = rep(2,length(xvals))
burnin=.1*MCMCiter
newTestNum = 6
xvalsStand = xvals/max(xvals)
xNew = seq(min(xvalsStand),max(xvalsStand),length=100)
varVals1 = matrix(nrow=numRounds,ncol=length(xvals))
varVals2 = matrix(nrow=numRounds,ncol=length(xNew))
# Specifiying if we want ED50, ED90, or both, as our criteria
EDTag = c(1,2,3) # 1 = both ED50 and ED90, 2 = ED50 only, 3 = ED90 only
EDTag = 2

########################
### GAUSSIAN PROCESS ###
```

```
####### Adaptive #######
########################

for(overallCount in 1:4){

xvals = c(10,25,50,100,500,1000)
numRounds = 19
numTests = rep(2,length(xvals))
burnin=.1*MCMCiter
newTestNum = 6
xvalsStand = xvals/max(xvals)
xNew = seq(min(xvalsStand),max(xvalsStand),length=100)
varVals1 = matrix(nrow=numRounds,ncol=length(xvals))
varVals2 = matrix(nrow=numRounds,ncol=length(xNew))

for(totalCount in 1:numRounds){

probFunction = overallCount

###############
### Priors ###
###############

# gamma follows a unif(gamAlpha,gamBeta)
gamAlpha = .01
gamBeta = 20
gammaMat = rep(0,MCMCiter)
gammaMat[1] = 3

# tau follows a gamma(gamAlpha,gamBeta)
tauAlpha = 1/2
tauBeta = 2
tauMat = rep(0,MCMCiter)
tauMat[1] = .1

if(probFunction == 1){fx = fx1}
if(probFunction == 2){fx = fx2}
if(probFunction == 3){fx = fx3}
if(probFunction == 4){fx = fx4}


if(totalCount==1){
xx = seq(0,1,length=100)
prob = fx(xx)
#plot(xx,prob,col='grey',type='l',ylim=c(0,1))
#abline(h=c(.5))
}

# Concentrations
if(totalCount==1){
xUnstand = c(rep(xvals[1],numTests[1]),rep(xvals[2],numTests[2]),rep(xvals
   [3],numTests[3]),rep(xvals[4],numTests[4]),rep(xvals[5],numTests[5]),
   rep(xvals[6],numTests[6]))
x = xUnstand/max(xUnstand)
```

```
}else{
xUnstandAdd = c(rep(xvals[1],newGroups[1]),rep(xvals[2], newGroups[2]),rep
   (xvals[3], newGroups[3]),rep(xvals[4], newGroups[4]),rep(xvals[5],
   newGroups[5]),rep(xvals[6], newGroups[6]))
xAdd = xUnstandAdd/max(xUnstand)
x = c(x,xAdd)
}

# Creating Tags
tags = rep(0,sum(numTests))
uVals = unique(x)
groups = seq(1,length(numTests),by=1)
for(i in 1:sum(numTests)){
   for(j in 1:length(xvals)){
      if(x[i]==uVals[j]){tags[i] = groups[j]}
   }
}

########################
### Data Generation ###
########################

if(totalCount==1){
prob = fx(x)
m = sum(numTests)
y = rbinom(m,1,prob)
data = cbind(y,1,x)
}else{
prob = fx(xAdd)
mAdd = length(xAdd)
yAdd = rbinom(mAdd,1,prob)
m = sum(numTests)
y = c(y,yAdd)
data = cbind(y,1,x)
}

############################
### Initializing Values ###
############################

n = length(y)
Sigma = diag(n)
mu = rep(0,n)
etaMat = matrix(nrow = n,ncol=MCMCiter)
probMat = matrix(nrow=n,ncol=MCMCiter)
predEtaMat = matrix(nrow=length(xNew),ncol=MCMCiter)
probPredMat = matrix(nrow=length(xNew),ncol=MCMCiter)
#yVec = (-2*(y==0))+rep(1,n) #-1's and 1's
zMat = matrix(nrow = n,ncol=MCMCiter)
posIndicator = which(y==1)
negIndicator = which(y==0)
numSuccess = sum(y)
numFailure = n-sum(y)
for (i in 1:numSuccess){
```

```
   zMat[posIndicator[i],1] = rtnorm(1,0,1,lower=0,upper=Inf)
}
for(i in 1:numFailure){
   zMat[negIndicator[i],1] = rtnorm(1,0,1,lower=-Inf,upper=0)
}
#zMat[,1] = zMat[,1]*yVec
psiMat = matrix(0,nrow=9,ncol=MCMCiter)

# Tuning information for Gamma (Metropolis-Hastings)
csig = 15
ar = 1

# eta follows a guassian process prior
mu = rep(0,n)
Sigma = corMatrixFast(x,gammaMat[1],tauMat[1])
Sigma = errorCheck(Sigma)

################################
### Gibbs/Metroplis Hastings ###
################################
counter = 0
for(i in 2:MCMCiter){

# updating Sigma
Sigma = corMatrixFast(x,gammaMat[i-1],tauMat[i-1])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
psiMat[1,i] = checkError$psi

# updating eta
tempSig = diag(n)+solve(Sigma)
checkError = errorCheck(tempSig)
tempSig = checkError$Sigma
psiMat[2,i] = checkError$psi
muStar = t(zMat[,(i-1)])%*%solve(tempSig)
sigmaStar = solve(tempSig)
etaMat[,i] = mvrnorm(1,muStar,sigmaStar)
probMat[,i] = pnorm(etaMat[,i])

# updating z
for (k in 1:numSuccess){
   zMat[posIndicator[k],i] = rtnorm(1,etaMat[posIndicator[k],i],1,lower=0,
      upper=Inf)
}
for(k in 1:numFailure){
   zMat[negIndicator[k],i] = rtnorm(1,etaMat[negIndicator[k],i],1,lower=-
      Inf,upper=0)
}
#zMat[,i] = abs(rnorm(n,etaMat[,i],1))
#zMat[,i] = zMat[,i]*yVec

# getting values in between points
sigma11 = corMatrixFast(xNew,gammaMat[i-1],tauMat[i-1])
sigma12 = corMatrix(xNew,x,gammaMat[i-1],tauMat[i-1])
```

```
sigma21 = t(sigma12)
sigma22 = Sigma
checkError = errorCheck(sigma11)
sigma11 = checkError$Sigma
psiMat[3,i] = checkError$psi
checkError = errorCheck(sigma22)
sigma22 = checkError$Sigma
psiMat[4,i] = checkError$psi
muStar = rep(0,length(xNew)) + sigma12%*%solve(sigma22)%*%t(t(etaMat[,i]))
sigmaStar = sigma11 - sigma12%*%solve(sigma22)%*%sigma21
sigmaStar = (sigmaStar + t(sigmaStar))/2
newEta = mvrnorm(1,muStar,sigmaStar) #chol only one that worked
predEtaMat[,i] = newEta
probPredMat[,i] = pnorm(predEtaMat[,i])

# updating tau
sigmaNot = corMatrixFast(x,gammaMat[i-1],1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[5,i] = checkError$psi

tauAlphaStar = tauAlpha + n/2
tauBetaStar = .5*(t(etaMat[,i])%*%solve(sigmaNot)%*%etaMat[,i]) + tauBeta
tauMat[i] = rgamma(1,tauAlphaStar,tauBetaStar) # using rate parameterizion
    as that is how I found nice closed form solution

# updating gamma
gammaMat[i] = gammaMat[i-1]
cand = rnorm(1,gammaMat[i-1],csig)
if(cand > gamAlpha && cand < gamBeta){
counter = counter + 1
Sigma = corMatrixFast(x,gammaMat[i],tauMat[i])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
psiMat[6,i] = checkError$psi
sigmaNot = corMatrixFast(x,gammaMat[i],1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[7,i] = checkError$psi

llo = log(det(sigmaNot)^(-1/2))-.5*t(etaMat[,i])%*%solve(Sigma)%*%etaMat[,
   i]

Sigma = corMatrixFast(x,cand,tauMat[i])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
psiMat[8,i] = checkError$psi
sigmaNot = corMatrixFast(x,cand,1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[9,i] = checkError$psi
lln = log(det(sigmaNot)^(-1/2))-.5*t(etaMat[,i])%*%solve(Sigma)%*%etaMat[,
   i]
```

```
u = runif(1)
if(log(u) < (lln-llo)){
   gammaMat[i] = cand
   ar = ar + 1
   }
}

}

aRate = ar/MCMCiter
print(paste('Round ',totalCount,'prob function',probFunction,' finished at
   ',date()))

################################
### Summerizing Information ###
################################
qFun1 = function(vec){quantile(vec,.025)}
qFun2 = function(vec){quantile(vec,.975)}

finalP = apply(probMat[,burnin:MCMCiter],1,mean)
finalPLower = apply(probMat[,burnin:MCMCiter],1,qFun1)
finalPUpper = apply(probMat[,burnin:MCMCiter],1,qFun2)
finalPred = apply(probPredMat[,burnin:MCMCiter],1,mean)
finalPredLower = apply(probPredMat[,burnin:MCMCiter],1,qFun1)
finalPredUpper = apply(probPredMat[,burnin:MCMCiter],1,qFun2)
finalGamma = gammaMat[burnin:MCMCiter]
finalTau = tauMat[burnin:MCMCiter]
finalPsiMat = psiMat[,burnin:MCMCiter]

#################
### Plotting ###
#################
#plot(x,prob,col='grey',type='l',ylim=c(0,1))
#plot(seq(0,1,length=100),fx(seq(0,1,length=100)),ylim=c(0,1),type='lines
   ')
#lines(xNew,finalPred,col=totalCount)
#lines(xNew,finalPred,col=blue,lty=2)
#lines(x,finalPUpper,col='blue',lty=2)
#lines(x,finalPLower,col='blue',lty=2)
#points(xNew,finalPred,col='red')
#points(x,finalP,col='blue')
#abline(h=c(.5,.9))

#################
### Updating ###
#################

# Finding variances per concentration
uTags = unique(tags)
for(i in 1:ncol(varVals1)){
   index = matrix(1*(tags==uTags[i]))
         interval = which(index!=0)
         tagMatrix = probMat[interval,(burnin:MCMCiter)]
         varVals1[totalCount,i] = var(as.vector(tagMatrix))
```

```
}

for(i in 1:ncol(varVals2)){
   varVals2[totalCount,i] = var(probPredMat[i,(burnin:MCMCiter)])
}

# Calculating ED50
allX = c(x,xNew)
allP = c(finalP,finalPred)
probXMat = cbind(allX,allP)
probXMat = probXMat[order(probXMat[,1]),] # sorted by concentration
ED = c(.5,.9)

maxP = max(allP)
minP = min(allP)
maxX = allX[which(allP==maxP)]
minX = allX[which(allP==minP)]

# checking to see if C50 and C90 can be calculated
ED50 = TRUE # TRUE means it can be calculated
ED90 = TRUE # TRUE means it can be calculated
if((maxP<ED[1] && minP<ED[1]) || (maxP>ED[1] && minP>ED[1])){
   ED50 = FALSE
}
if((maxP<ED[2] && minP<ED[2]) || (maxP>ED[2] && minP>ED[2])){
   ED90 = FALSE
}

if(ED50==TRUE){
# checking for parabolic nature
parabolic = FALSE
index1 = which(probXMat[,1]<maxX)
if(length(index1)>0){
maxPIndex1 = max(probXMat[index1,2])
minPIndex1 = min(probXMat[index1,2])
   if((maxPIndex1 > ED[1] && minPIndex1<ED[1])){
      ED1Index = index1
   }
}


index2 = which(probXMat[,1]<minX)
if(length(index2)>0){
maxPIndex2 = max(probXMat[index2,2])
minPIndex2 = min(probXMat[index2,2])
   if((maxPIndex2 > ED[1] && minPIndex2<ED[1])){
      ED1Index = index2
   }
}

newProbXMat = probXMat[ED1Index,]
diff1 = abs(newProbXMat[,2] - ED[1])
diff2 = abs(newProbXMat[,2] - ED[2])
EDindex1 = which(diff1==min(diff1))
```

```
EDindex2 = which(diff2==min(diff2))
EDval1 = newProbXMat[EDindex1,1]
EDval2 = newProbXMat[EDindex2,1]
EDval = EDval1
}

# Assigning new probabilities
newProbs = rep(0,length(xvals))
if(ED50==TRUE){
for(i in 1:length(xvals)){
   if(EDTag == 1){
      d1 = abs(xvalsStand[i]-EDval1)
      d2 = abs(xvalsStand[i]-EDval2)
      d = (1-d1) + (1-d2) # the smaller the d values, the greater the
         weight is given to them
   }else if(EDTag == 2){
      d1 = (abs(xvalsStand[i]-EDval1))
      d = 1 - d1
   }else if(EDTag == 3){
      d1 = (abs(xvalsStand[i]-EDval2))
      d = 1 - d1
   }
   newProbs[i] = (d) * sqrt(varVals1[totalCount,i])
}
newProbs = newProbs/(sum(newProbs))
}else{
newProbs = rep(1/6,6)
EDval = NA
}

# Saving information to a list
fileName=paste('GP_Adapt_Trial',totalCount,'Function',probFunction,'ED',
   EDTag)
l = list(x=x,finalP=finalP,finalPred=finalPred,finalPredLower=
   finalPredLower,finalPredUpper=finalPredUpper,finalPUpper=finalPUpper,
   finalPLower=finalPLower,varVals1=varVals1,varVals2=varVals2,numTests =
   numTests,xNew=xNew,aRate=aRate,EDval=EDval,finalGamma=finalGamma,
   finalPsiMat=finalPsiMat,finalTau=finalTau)
save(l,file=fileName)

# Creating new data
newAllocation = sample(groups,size=newTestNum,replace=TRUE,prob=newProbs)
newGroups = rep(0,length(groups))
newNumTests = numTests
for(i in 1:length(groups)){
   newGroups[i] = sum(newAllocation==groups[i])
}
numTests = numTests + newGroups


}
numTests = numTests - newGroups

}
```

```
######################
## GAUSSIAN PROCESS ##
### Non - Adaptive ###
######################

for(overallCount in 1:4){

xvals = c(10,25,50,100,500,1000)
numRounds = 19
numTests = rep(2,length(xvals))
burnin=.1*MCMCiter
newTestNum = 6
xvalsStand = xvals/max(xvals)
xNew = seq(min(xvalsStand),max(xvalsStand),length=100)
varVals1 = matrix(nrow=numRounds,ncol=length(xvals))
varVals2 = matrix(nrow=numRounds,ncol=length(xNew))

for(totalCount in 1:numRounds){

probFunction = overallCount

##############
### Priors ###
##############

# gamma follows a unif(gamAlpha,gamBeta)
gamAlpha = .01
gamBeta = 20
gammaMat = rep(0,MCMCiter)
gammaMat[1] = 3

# tau follows a gamma(gamAlpha,gamBeta)
tauAlpha = 1/2
tauBeta = 2
tauMat = rep(0,MCMCiter)
tauMat[1] = .1

if(probFunction == 1){fx = fx1}
if(probFunction == 2){fx = fx2}
if(probFunction == 3){fx = fx3}
if(probFunction == 4){fx = fx4}


if(totalCount==1){
xx = seq(0,1,length=100)
prob = fx(xx)
#plot(xx,prob,col='grey',type='l',ylim=c(0,1))
#abline(h=c(.5))
}


# Concentrations
```

```
if(totalCount==1){
xUnstand = c(rep(xvals[1],numTests[1]),rep(xvals[2],numTests[2]),rep(xvals
   [3],numTests[3]),rep(xvals[4],numTests[4]),rep(xvals[5],numTests[5]),
   rep(xvals[6],numTests[6]))
x = xUnstand/max(xUnstand)
}else{
xUnstandAdd = c(rep(xvals[1],newGroups[1]),rep(xvals[2], newGroups[2]),rep
   (xvals[3], newGroups[3]),rep(xvals[4], newGroups[4]),rep(xvals[5],
   newGroups[5]),rep(xvals[6], newGroups[6]))
xAdd = xUnstandAdd/max(xUnstand)
x = c(x,xAdd)
}

# Creating Tags
tags = rep(0,sum(numTests))
uVals = unique(x)
groups = seq(1,length(numTests),by=1)
for(i in 1:sum(numTests)){
   for(j in 1:length(xvals)){
      if(x[i]==uVals[j]){tags[i] = groups[j]}
   }
}

########################
### Data Generation ###
########################

if(totalCount==1){
prob = fx(x)
m = sum(numTests)
y = rbinom(m,1,prob)
data = cbind(y,1,x)
}else{
prob = fx(xAdd)
mAdd = length(xAdd)
yAdd = rbinom(mAdd,1,prob)
m = sum(numTests)
y = c(y,yAdd)
data = cbind(y,1,x)
}
############################
### Initializing Values ###
############################

n = length(y)
Sigma = diag(n)
mu = rep(0,n)
etaMat = matrix(nrow = n,ncol=MCMCiter)
probMat = matrix(nrow=n,ncol=MCMCiter)
predEtaMat = matrix(nrow=length(xNew),ncol=MCMCiter)
probPredMat = matrix(nrow=length(xNew),ncol=MCMCiter)
#yVec = (-2*(y==0))+rep(1,n) #-1's and 1's
zMat = matrix(nrow = n,ncol=MCMCiter)
posIndicator = which(y==1)
```

```
negIndicator = which(y==0)
numSuccess = sum(y)
numFailure = n-sum(y)
for (i in 1:numSuccess){
   zMat[posIndicator[i],1] = rtnorm(1,0,1,lower=0,upper=Inf)
}
for(i in 1:numFailure){
   zMat[negIndicator[i],1] = rtnorm(1,0,1,lower=-Inf,upper=0)
}
#zMat[,1] = zMat[,1]*yVec
psiMat = matrix(0,nrow=9,ncol=MCMCiter)

# Tuning information for Gamma (Metropolis-Hastings)
csig = 15
ar = 1

# eta follows a guassian process prior
mu = rep(0,n)
Sigma = corMatrixFast(x,gammaMat[1],tauMat[1])
Sigma = errorCheck(Sigma)

################################
### Gibbs/Metroplis Hastings ###
################################
if(sum(totalCount == c(1,3,11,19))>0){ # only look at desired sample sizes

counter = 0
for(i in 2:MCMCiter){

# updating Sigma
Sigma = corMatrixFast(x,gammaMat[i-1],tauMat[i-1])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
psiMat[1,i] = checkError$psi

# updating eta
tempSig = diag(n)+solve(Sigma)
checkError = errorCheck(tempSig)
tempSig = checkError$Sigma
psiMat[2,i] = checkError$psi
muStar = t(zMat[,(i-1)])%*%solve(tempSig)
sigmaStar = solve(tempSig)
etaMat[,i] = mvrnorm(1,muStar,sigmaStar)
probMat[,i] = pnorm(etaMat[,i])

# updating z
for (k in 1:numSuccess){
   zMat[posIndicator[k],i] = rtnorm(1,etaMat[posIndicator[k],i],1,lower=0,
      upper=Inf)
}
for(k in 1:numFailure){
   zMat[negIndicator[k],i] = rtnorm(1,etaMat[negIndicator[k],i],1,lower=-
      Inf,upper=0)
}
```

```
#zMat[,i] = abs(rnorm(n,etaMat[,i],1))
#zMat[,i] = zMat[,i]*yVec

# getting values in between points
sigma11 = corMatrixFast(xNew,gammaMat[i-1],tauMat[i-1])
sigma12 = corMatrix(xNew,x,gammaMat[i-1],tauMat[i-1])
sigma21 = t(sigma12)
sigma22 = Sigma
checkError = errorCheck(sigma11)
sigma11 = checkError$Sigma
psiMat[3,i] = checkError$psi
checkError = errorCheck(sigma22)
sigma22 = checkError$Sigma
psiMat[4,i] = checkError$psi
muStar = rep(0,length(xNew)) + sigma12%*%solve(sigma22)%*%t(t(etaMat[,i]))
sigmaStar = sigma11 - sigma12%*%solve(sigma22)%*%sigma21
sigmaStar = (sigmaStar + t(sigmaStar))/2
newEta = mvrnorm(1,muStar,sigmaStar) #chol only one that worked
predEtaMat[,i] = newEta
probPredMat[,i] = pnorm(predEtaMat[,i])


# updating tau
sigmaNot = corMatrixFast(x,gammaMat[i-1],1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[5,i] = checkError$psi


tauAlphaStar = tauAlpha + n/2
tauBetaStar = .5*(t(etaMat[,i])%*%solve(sigmaNot)%*%etaMat[,i]) + tauBeta
tauMat[i] = rgamma(1,tauAlphaStar,tauBetaStar) # using rate parameterizion
    as that is how I found nice closed form solution

# updating gamma
gammaMat[i] = gammaMat[i-1]
cand = rnorm(1,gammaMat[i-1],csig)
if(cand > gamAlpha && cand < gamBeta){
counter = counter + 1
Sigma = corMatrixFast(x,gammaMat[i],tauMat[i])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
psiMat[6,i] = checkError$psi
sigmaNot = corMatrixFast(x,gammaMat[i],1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[7,i] = checkError$psi

llo = log(det(sigmaNot)^(-1/2))-.5*t(etaMat[,i])%*%solve(Sigma)%*%etaMat[,
   i]

Sigma = corMatrixFast(x,cand,tauMat[i])
checkError = errorCheck(Sigma)
Sigma = checkError$Sigma
```

```
psiMat[8,i] = checkError$psi
sigmaNot = corMatrixFast(x,cand,1)
checkError = errorCheck(sigmaNot)
sigmaNot = checkError$Sigma
psiMat[9,i] = checkError$psi

lln = log(det(sigmaNot)^(-1/2))-.5*t(etaMat[,i])%*%solve(Sigma)%*%etaMat[,
   i]

u = runif(1)
if(log(u) < (lln-llo)){
   gammaMat[i] = cand
   ar = ar + 1
   }
}
#print(i)
}

aRate = ar/MCMCiter
print(paste('Round ',totalCount,'prob function',probFunction,' finished at
   ',date()))

################################
### Summerizing Information ###
################################
qFun1 = function(vec){quantile(vec,.025)}
qFun2 = function(vec){quantile(vec,.975)}

finalP = apply(probMat[,burnin:MCMCiter],1,mean)
finalPLower = apply(probMat[,burnin:MCMCiter],1,qFun1)
finalPUpper = apply(probMat[,burnin:MCMCiter],1,qFun2)
finalPred = apply(probPredMat[,burnin:MCMCiter],1,mean)
finalPredLower = apply(probPredMat[,burnin:MCMCiter],1,qFun1)
finalPredUpper = apply(probPredMat[,burnin:MCMCiter],1,qFun2)
finalGamma = gammaMat[burnin:MCMCiter]
finalTau = tauMat[burnin:MCMCiter]
finalPsiMat = psiMat[,burnin:MCMCiter]


#################
### Updating ###
#################

# Finding variances per concentration
uTags = unique(tags)
for(i in 1:ncol(varVals1)){
   index = matrix(1*(tags==uTags[i]))
         interval = which(index!=0)
         tagMatrix = probMat[interval,(burnin:MCMCiter)]
         varVals1[totalCount,i] = var(as.vector(tagMatrix))
}

for(i in 1:ncol(varVals2)){
   varVals2[totalCount,i] = var(probPredMat[i,(burnin:MCMCiter)])
```

```
}

# Calculating ED50
allX = c(x,xNew)
allP = c(finalP,finalPred)
probXMat = cbind(allX,allP)
probXMat = probXMat[order(probXMat[,1]),] # sorted by concentration
ED = c(.5,.9)

maxP = max(allP)
minP = min(allP)
maxX = allX[which(allP==maxP)]
minX = allX[which(allP==minP)]

# checking to see if C50 and C90 can be calculated
ED50 = TRUE # TRUE means it can be calculated
ED90 = TRUE # TRUE means it can be calculated
if((maxP<ED[1] && minP<ED[1]) || (maxP>ED[1] && minP>ED[1])){
   ED50 = FALSE
}
if((maxP<ED[2] && minP<ED[2]) || (maxP>ED[2] && minP>ED[2])){
   ED90 = FALSE
}

# checking for parabolic nature
parabolic = FALSE
index1 = which(probXMat[,1]<maxX)
if(length(index1)>0){
maxPIndex1 = max(probXMat[index1,2])
minPIndex1 = min(probXMat[index1,2])
   if((maxPIndex1 > ED[1] && minPIndex1<ED[1])){
      ED1Index = index1
   }
}


index2 = which(probXMat[,1]<minX)
if(length(index2)>0){
maxPIndex2 = max(probXMat[index2,2])
minPIndex2 = min(probXMat[index2,2])
   if((maxPIndex2 > ED[1] && minPIndex2<ED[1])){
      ED1Index = index2
   }
}

newProbXMat = probXMat[ED1Index,]
diff1 = abs(newProbXMat[,2] - ED[1])
diff2 = abs(newProbXMat[,2] - ED[2])
EDindex1 = which(diff1==min(diff1))
EDindex2 = which(diff2==min(diff2))
EDval1 = newProbXMat[EDindex1,1]
EDval2 = newProbXMat[EDindex2,1]
EDval = EDval1
if(ED50==FALSE){EDval=NA}
```

```
# Saving information to a list
fileName=paste('GP_Non_Adapt_Trial',totalCount,'Function',probFunction,'ED
   ',EDTag)
l = list(x=x,finalP=finalP,finalPred=finalPred,finalPredLower=
   finalPredLower,finalPredUpper=finalPredUpper,finalPUpper=finalPUpper,
   finalPLower=finalPLower,varVals1=varVals1,varVals2=varVals2,numTests =
   numTests,xNew=xNew,aRate=aRate,EDval=EDval,finalGamma=finalGamma,
   finalPsiMat=finalPsiMat,finalTau=finalTau)
save(l,file=fileName)


} # ending the if statement


# Creating new data
newGroups = rep(1,length(groups))
numTests = numTests + newGroups


}
numTests = numTests - newGroups

}


#################
### LOGISTIC ###
### Adaptive ###
#################

for(overallCount in 1:4){

xvals = c(10,25,50,100,500,1000)
numRounds = 19
numTests = rep(2,length(xvals))
burnin=.1*MCMCiter
newTestNum = 6
xvalsStand = xvals/max(xvals)
xNew = seq(min(xvalsStand),max(xvalsStand),length=100)
varVals1 = matrix(nrow=numRounds,ncol=length(xvals))
varVals2 = matrix(nrow=numRounds,ncol=length(xNew))

for(totalCount in 1:numRounds){

probFunction = overallCount

###############
### Priors ###
###############

# alpha follows a normal(meanAlpha,sdAlpha)
meanAlpha = 0
sdAlpha = 100
```

```
alphaVec = rep(0,MCMCiter)
alphaVec[1] = 1

# beta follows a normal(meanBeta,sdBeta)
meanBeta = 0
sdBeta = 100
betaVec = rep(0,MCMCiter)
betaVec[1] = 1

#####################################
### Complete Conditional Function ###
#####################################

aFunction = function(data,alpha,beta){
m = length(data[,1])
n = data[,2]
y = data[,1]
x = data[,3]
prob=1/(1+exp(-(alpha+beta*x)))
result = sum(dbinom(y,n,prob,log=T)) + dnorm(alpha,meanAlpha,sdAlpha,log=T
   )
result
}
bFunction = function(data,alpha,beta){
m = length(data[,1])
n = data[,2]
y = data[,1]
x = data[,3]
prob=1/(1+exp(-(alpha+beta*x)))
result = sum(dbinom(y,n,prob,log=T)) + dnorm(beta,meanBeta,sdBeta,log=T)
result
}


if(probFunction == 1){fx = fx1}
if(probFunction == 2){fx = fx2}
if(probFunction == 3){fx = fx3}
if(probFunction == 4){fx = fx4}

if(totalCount==1){
xx = seq(0,1,length=100)
prob = fx(xx)
#plot(xx,prob,col='grey',type='l',ylim=c(0,1))
#abline(h=c(.5))
}


# Concentrations
if(totalCount==1){
xUnstand = c(rep(xvals[1],numTests[1]),rep(xvals[2],numTests[2]),rep(xvals
   [3],numTests[3]),rep(xvals[4],numTests[4]),rep(xvals[5],numTests[5]),
   rep(xvals[6],numTests[6]))
x = xUnstand/max(xUnstand)
}else{
```

```
xUnstandAdd = c(rep(xvals[1],newGroups[1]),rep(xvals[2], newGroups[2]),rep
   (xvals[3], newGroups[3]),rep(xvals[4], newGroups[4]),rep(xvals[5],
   newGroups[5]),rep(xvals[6], newGroups[6]))
xAdd = xUnstandAdd/max(xUnstand)
x = c(x,xAdd)
}

# Creating Tags
tags = rep(0,sum(numTests))
uVals = unique(x)
groups = seq(1,length(numTests),by=1)
for(i in 1:sum(numTests)){
   for(j in 1:length(xvals)){
      if(x[i]==uVals[j]){tags[i] = groups[j]}
   }
}

########################
### Data Generation ###
########################

if(totalCount==1){
prob = fx(x)
m = sum(numTests)
y = rbinom(m,1,prob)
data = cbind(y,1,x)
}else{
prob = fx(xAdd)
mAdd = length(xAdd)
yAdd = rbinom(mAdd,1,prob)
m = sum(numTests)
y = c(y,yAdd)
data = cbind(y,1,x)
}

############################
### Initializing Values ###
############################

n = length(y)
csAlpha = c(1.5,1.5,2,1.5)
csBeta = c(1.5,1.5,2,1.5)
countAlpha=0
countBeta=0
probMat = matrix(nrow=n,ncol=MCMCiter)
probPredMat = matrix(nrow=length(xNew),ncol=MCMCiter)

################################
### Gibbs/Metroplis Hastings ###
################################
for(i in 2:MCMCiter)
{
   candAlpha = rnorm(1,alphaVec[i-1],csAlpha[overallCount])
      alphaOld = aFunction(data,alphaVec[i-1],betaVec[i-1])
```

```
      alphaNew = aFunction(data,candAlpha,betaVec[i-1])
      if ((alphaNew - alphaOld) > log(runif(1,0,1)))
      {
         alphaVec[i] = candAlpha
         countAlpha= countAlpha + 1
      }else
      {
         alphaVec[i] = alphaVec[i-1]
      }
   betaVec[i] = betaVec[i-1]
   candBeta = rnorm(1, betaVec[i-1],csBeta[overallCount])
      betaOld = bFunction(data,alphaVec[i],betaVec[i-1])
      betaNew = bFunction(data,alphaVec[i],candBeta)
      if ((betaNew - betaOld) > log(runif(1,0,1)))
      {
         betaVec[i] = candBeta
         countBeta = countBeta + 1
      }else
      {
         betaVec[i] = betaVec[i-1]
      }

   probMat[,i] = exp(alphaVec[i]+betaVec[i]*x)/(1+exp(alphaVec[i]+betaVec[
      i]*x))
   probPredMat[,i] = exp(alphaVec[i]+betaVec[i]*xNew)/(1+exp(alphaVec[i]+
      betaVec[i]*xNew))
}

aRate = countAlpha/MCMCiter
bRate = countBeta/MCMCiter
print(paste('Round ',totalCount,'prob function',probFunction,' finished at
   ',date()))

################################
### Summerizing Information ###
################################

finalAlpha = alphaVec[burnin:MCMCiter]
finalBeta = betaVec[burnin:MCMCiter]
pFunction = function(alpha,beta,x){
result = exp(alpha+beta*x)/(1+exp(alpha+beta*x))
result
}
xx = seq(0,1,length=100)
qFun1 = function(vec){quantile(vec,.025)}
qFun2 = function(vec){quantile(vec,.975)}
finalP = rep(0,length(xx))
finalPUpper = rep(0,length(xx))
finalPLower = rep(0,length(xx))

for(k in 1:length(xx)){
   finalP[k] = mean(pFunction(finalAlpha,finalBeta,xx[k]))
   finalPLower[k] = qFun1(pFunction(finalAlpha,finalBeta,xx[k]))
   finalPUpper[k] = qFun2(pFunction(finalAlpha,finalBeta,xx[k]))
```

```
}

################
### Updating ###
################

# Finding variances per concentration
uTags = unique(tags)
for(i in 1:ncol(varVals1)){
   index = matrix(1*(tags==uTags[i]))
         interval = which(index!=0)
         tagMatrix = probMat[interval,(burnin:MCMCiter)]
         varVals1[totalCount,i] = var(as.vector(tagMatrix))
}

for(i in 1:ncol(varVals2)){
   varVals2[totalCount,i] = var(probPredMat[i,(burnin:MCMCiter)])
}

# Calculating ED50
ED50 = (log((.5)/(1-.5))-mean(alphaVec)) / mean(betaVec)
ED90 =  (log((.9)/(1-.9))-mean(alphaVec)) / mean(betaVec)
EDval1 = ED50
EDval2 = ED90
EDval = EDval1

# Assigning new probabilities
newProbs = rep(0,length(xvals))
if(EDval>0 && EDval< 1){
for(i in 1:length(xvals)){
   if(EDTag == 1){
      d1 = abs(xvalsStand[i]-EDval1)
      d2 = abs(xvalsStand[i]-EDval2)
      d = (1-d1) + (1-d2) # the smaller the d values, the greater the
         weight is given to them
   }else if(EDTag == 2){
      d1 = (abs(xvalsStand[i]-EDval1))
      d = 1 - d1
   }else if(EDTag == 3){
      d1 = (abs(xvalsStand[i]-EDval2))
      d = 1 - d1
   }
   newProbs[i] = (d) * sqrt(varVals1[totalCount,i])
}
newProbs = newProbs/(sum(newProbs))
}else{
newProbs = rep(1/6,6)
EDval = NA
print(c(totalCount,EDval))
}


# Saving information to a list
```

```
fileName=paste('Log_Adapt_Trial',totalCount,'Function',probFunction,'ED',
   EDTag)
l = list(x=x,finalP=finalP,finalPUpper=finalPUpper,finalPLower=finalPLower
   ,varVals1=varVals1,varVals2=varVals2,numTests = numTests,aRate=aRate,
   bRate=bRate,EDval=EDval)
save(l,file=fileName)

# Creating new data
newAllocation = sample(groups,size=newTestNum,replace=TRUE,prob=newProbs)
newGroups = rep(0,length(groups))
newNumTests = numTests
for(i in 1:length(groups)){
   newGroups[i] = sum(newAllocation==groups[i])
}
numTests = numTests + newGroups

}
numTests = numTests - newGroups

}



####################
##### LOGISTIC #####
### Non-Adaptive ###
####################

for(overallCount in 1:4){

xvals = c(10,25,50,100,500,1000)
numRounds = 19
numTests = rep(2,length(xvals))
burnin=.1*MCMCiter
newTestNum = 6
xvalsStand = xvals/max(xvals)
xNew = seq(min(xvalsStand),max(xvalsStand),length=100)
varVals1 = matrix(nrow=numRounds,ncol=length(xvals))
varVals2 = matrix(nrow=numRounds,ncol=length(xNew))


for(totalCount in 1:numRounds){

probFunction = overallCount

###############
### Priors ###
###############

# alpha follows a normal(meanAlpha,sdAlpha)
meanAlpha = 0
sdAlpha = 100
alphaVec = rep(0,MCMCiter)
alphaVec[1] = 1
```

```
# beta follows a normal(meanBeta,sdBeta)
meanBeta = 0
sdBeta = 100
betaVec = rep(0,MCMCiter)
betaVec[1] = 1

####################################
### Complete Conditional Function ###
####################################

aFunction = function(data,alpha,beta){
m = length(data[,1])
n = data[,2]
y = data[,1]
x = data[,3]
prob=1/(1+exp(-(alpha+beta*x)))
result = sum(dbinom(y,n,prob,log=T)) + dnorm(alpha,meanAlpha,sdAlpha,log=T
   )
result
}
bFunction = function(data,alpha,beta){
m = length(data[,1])
n = data[,2]
y = data[,1]
x = data[,3]
prob=1/(1+exp(-(alpha+beta*x)))
result = sum(dbinom(y,n,prob,log=T)) + dnorm(beta,meanBeta,sdBeta,log=T)
result
}


if(probFunction == 1){fx = fx1}
if(probFunction == 2){fx = fx2}
if(probFunction == 3){fx = fx3}
if(probFunction == 4){fx = fx4}

if(totalCount==1){
xx = seq(0,1,length=100)
prob = fx(xx)
#plot(xx,prob,col='grey',type='l',ylim=c(0,1))
#abline(h=c(.5))
}




# Concentrations
if(totalCount==1){
xUnstand = c(rep(xvals[1],numTests[1]),rep(xvals[2],numTests[2]),rep(xvals
   [3],numTests[3]),rep(xvals[4],numTests[4]),rep(xvals[5],numTests[5]),
   rep(xvals[6],numTests[6]))
x = xUnstand/max(xUnstand)
}else{
```

```
xUnstandAdd = c(rep(xvals[1],newGroups[1]),rep(xvals[2], newGroups[2]),rep
   (xvals[3], newGroups[3]),rep(xvals[4], newGroups[4]),rep(xvals[5],
   newGroups[5]),rep(xvals[6], newGroups[6]))
xAdd = xUnstandAdd/max(xUnstand)
x = c(x,xAdd)
}

# Creating Tags
tags = rep(0,sum(numTests))
uVals = unique(x)
groups = seq(1,length(numTests),by=1)
for(i in 1:sum(numTests)){
   for(j in 1:length(xvals)){
      if(x[i]==uVals[j]){tags[i] = groups[j]}
   }
}

########################
### Data Generation ###
########################

if(totalCount==1){
prob = fx(x)
m = sum(numTests)
y = rbinom(m,1,prob)
data = cbind(y,1,x)
}else{
prob = fx(xAdd)
mAdd = length(xAdd)
yAdd = rbinom(mAdd,1,prob)
m = sum(numTests)
y = c(y,yAdd)
data = cbind(y,1,x)
}

############################
### Initializing Values ###
############################

n = length(y)
csAlpha = c(1.5,1.5,2,1.5)
csBeta = c(1.5,1.5,2,1.5)
countAlpha=0
countBeta=0
probMat = matrix(nrow=n,ncol=MCMCiter)
probPredMat = matrix(nrow=length(xNew),ncol=MCMCiter)

################################
### Gibbs/Metroplis Hastings ###
################################

if(sum(totalCount == c(1,3,11,19))>0){ # only look at desired sample sizes

for(i in 2:MCMCiter)
```

```
{
   candAlpha = rnorm(1,alphaVec[i-1],csAlpha[overallCount])
      alphaOld = aFunction(data,alphaVec[i-1],betaVec[i-1])
      alphaNew = aFunction(data,candAlpha,betaVec[i-1])
      if ((alphaNew - alphaOld) > log(runif(1,0,1)))
      {
         alphaVec[i] = candAlpha
         countAlpha= countAlpha + 1
      }else
      {
         alphaVec[i] = alphaVec[i-1]
      }
   betaVec[i] = betaVec[i-1]
   candBeta = rnorm(1, betaVec[i-1],csBeta[overallCount])
      betaOld = bFunction(data,alphaVec[i],betaVec[i-1])
      betaNew = bFunction(data,alphaVec[i],candBeta)
      if ((betaNew - betaOld) > log(runif(1,0,1)))
      {
         betaVec[i] = candBeta
         countBeta = countBeta + 1
      }else
      {
         betaVec[i] = betaVec[i-1]
      }

   probMat[,i] = exp(alphaVec[i]+betaVec[i]*x)/(1+exp(alphaVec[i]+betaVec[
      i]*x))
   probPredMat[,i] = exp(alphaVec[i]+betaVec[i]*xNew)/(1+exp(alphaVec[i]+
      betaVec[i]*xNew))
}

aRate = countAlpha/MCMCiter
bRate = countBeta/MCMCiter
print(paste('Round ',totalCount,'prob function',probFunction,' finished at
   ',date()))

################################
### Summerizing Information ###
################################
finalAlpha = alphaVec[burnin:MCMCiter]
finalBeta = betaVec[burnin:MCMCiter]
pFunction = function(alpha,beta,x){
result = exp(alpha+beta*x)/(1+exp(alpha+beta*x))
result
}
xx = seq(0,1,length=100)
qFun1 = function(vec){quantile(vec,.025)}
qFun2 = function(vec){quantile(vec,.975)}
finalP = rep(0,length(xx))
finalPUpper = rep(0,length(xx))
finalPLower = rep(0,length(xx))

for(k in 1:length(xx)){
   finalP[k] = mean(pFunction(finalAlpha,finalBeta,xx[k]))
```

```
   finalPLower[k] = qFun1(pFunction(finalAlpha,finalBeta,xx[k]))
   finalPUpper[k] = qFun2(pFunction(finalAlpha,finalBeta,xx[k]))
}


#################
### Updating ###
#################

# Finding variances per concentration
uTags = unique(tags)
for(i in 1:ncol(varVals1)){
   index = matrix(1*(tags==uTags[i]))
         interval = which(index!=0)
         tagMatrix = probMat[interval,(burnin:MCMCiter)]
         varVals1[totalCount,i] = var(as.vector(tagMatrix))
}

for(i in 1:ncol(varVals2)){
   varVals2[totalCount,i] = var(probPredMat[i,(burnin:MCMCiter)])
}

# Calculating ED50
ED50 = (log((.5)/(1-.5))-mean(finalAlpha)) / mean(finalBeta)
ED90 =  (log((.9)/(1-.9))-mean(finalAlpha)) / mean(finalBeta)
EDval1 = ED50
EDval2 = ED90
EDval = EDval1
if(EDval<0 || EDval>1){EDval = NA}

# Saving information to a list
fileName=paste('Log_Non_Adapt_Trial',totalCount,'Function',probFunction,'
   ED',EDTag)
l = list(x=x,finalP=finalP,finalPUpper=finalPUpper,finalPLower=finalPLower
   ,varVals1=varVals1,varVals2=varVals2,numTests = numTests,aRate=aRate,
   bRate=bRate,EDval=EDval)
save(l,file=fileName)

} # end if statement

# Creating new data
newGroups = rep(1,length(unique(xvals)))
numTests = numTests + newGroups

}
numTests = numTests - newGroups

}
```

B.2 Results

```
###################################
########  ANALYZING RESULTS ########
###################################
```

```
# Initializing Arrays and Vectors
biasArray = array(0,dim=c(3,3,4)) # 3 sample sizes (24,72,120),3
   probability curves, 4 scenarios (GP Adapt, GP Non Adapt, Log Adapt, Log
    Non Adapt)
varVal1Array = array(0,dim=c(19,6,4,4)) #19 interim steps,6 concentrations
   , 4 probability curves, 4 scenarios (GP Adapt, GP Non Adapt, Log Adapt,
    Log Non Adapt)
varVal2Array = array(0,dim=c(19,100,4,4)) #19 interim steps,6
   concentrations, 4 probability curves, 4 scenarios (GP Adapt, GP Non
   Adapt, Log Adapt, Log Non Adapt)
numTestArray = array(0,dim=c(19,6,4,4))#19 interim steps,6 concentrations,
    4 probability curves, 4 scenarios (GP Adapt, GP Non Adapt, Log Adapt,
   Log Non Adapt)
trueC50 = c(0.18839377008567554,0.5802775,.4)
sampleSizes = c(3,11,19) # 24,72,120
##############################
#### Initializing Numbers ###
##############################

numSampleSizes = 3
numProbCurves = 4
numScenario = 4
numAdaptTrials = 19

######################
## GAUSSIAN PROCESS ##
###### Adaptive ######
######################
ensemble = 1
for(i in 1:numSampleSizes){
   for(j in 1:numSampleSizes){
      fName = paste('GP_Adapt_Trial',sampleSizes[j],'Function',i,'ED',2)
      load(fName)
      biasArray[j,i,ensemble] = abs(l$EDval-trueC50[i])
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('GP_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   varVal1Array[i,,k,ensemble] = l$varVals1[i,]
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('GP_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   varVal2Array[i,,k,ensemble] = l$varVals2[i,]
   }
}
```

```
for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('GP_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   numTestArray[i,,k,ensemble] = l$numTests
   }
}

for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Adapt_Trial',j,'Function',1,'ED',2)
   load(fName)
   if(j==sampleSizes[1]){
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',col=i,main='Comparing
         Sample Size',xlab='Concentration',ylab='Probability of Detection
         ')
      abline(h=.5)
   }else{
      lines(xx,l$finalPred,col=i)
   }
}
lines(xx,fx1(xx),lwd=3,col='grey')
legend('topleft',c('30','60','90','120'),col=c(1,2,3,4),lwd=c(2,2,2,2))

for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Adapt_Trial',j,'Function',1,'ED',2)
   load(fName)
   if(j==sampleSizes[1]){
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',col=i,main='Comparing
         Sample Size',xlab='Concentration',ylab='Probability of Detection
         ')
      abline(h=.5)
      lines(xx,l$finalPredUpper,lty=2,col=i)
      lines(xx,l$finalPredLower,lty=2,col=i)
   }else{
      lines(xx,l$finalPred,col=i)
      lines(xx,l$finalPredUpper,lty=2,col=i)
      lines(xx,l$finalPredLower,lty=2,col=i)
   }
}
lines(xx,fx1(xx),lwd=3,col='grey')
legend('topleft',c('30','60','90','120'),col=c(1,2,3,4),lwd=c(2,2,2,2))

for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Adapt_Trial',j,'Function',1,'ED',2)
   load(fName)
   if(j==sampleSizes[1]){
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',col=i,main='Comparing
         Sample Size',xlab='Concentration',ylab='Probability of Detection
         ')
      fName = paste('Log_Adapt_Trial',j,'Function',1,'ED',2)
      load(fName)
```

```
      lines(xx,l$finalP,col=i,lty=2)
      abline(h=.5)
   }else{
      lines(xx,l$finalPred,col=i)
      fName = paste('Log_Adapt_Trial',j,'Function',1,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=i,lty=2)
   }
}
lines(xx,fx1(xx),lwd=3,col='grey')
legend('topleft',c('24','72','120'),col=c(1,2,3),lwd=c(2,2,2))

######################
## GAUSSIAN PROCESS ##
### Non - Adaptive ###
######################
ensemble = 2
for(i in 1:numSampleSizes){
   for(j in 1:numSampleSizes){
      fName = paste('GP_Non_Adapt_Trial',sampleSizes[j],'Function',i,'ED
         ',2)
      load(fName)
      biasArray[j,i,ensemble] = abs(l$EDval-trueC50[i])
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Non_Adapt_Trial',j,'Function',k,'ED',2)
         load(fName)
   varVal1Array[j,,k,ensemble] = l$varVals1[j,]
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Non_Adapt_Trial',j,'Function',k,'ED',2)
         load(fName)
   varVal2Array[j,,k,ensemble] = l$varVals2[j,]
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('GP_Non_Adapt_Trial',j,'Function',k,'ED',2)
         load(fName)
   numTestArray[j,,k,ensemble] = l$numTests
   }
}

for(i in 1:numSampleSizes){
```

```
   j = sampleSizes[i]
   fName = paste('GP_Non_Adapt_Trial',j,'Function',1,'ED',2)
         load(fName)
   if(i==1){plot(xx,l$finalPred,ylim=c(0,1),type='lines',col=i)}else{
   lines(xx,l$finalPred,col=i)
   }
   }
lines(xx,fx1(xx),lwd=3)

#################
### LOGISTIC ###
### Adaptive ###
#################
ensemble = 3
for(i in 1:numSampleSizes){
   for(j in 1:numSampleSizes){
      fName = paste('Log_Adapt_Trial',sampleSizes[j],'Function',i,'ED',2)
      load(fName)
      if(l$EDval>0 && l$EDval<1){
      biasArray[j,i,ensemble] = abs(l$EDval-trueC50[i])
      }else{
      biasArray[j,i,ensemble] = 'NA'
      }
   }
}


for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('Log_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   varVal1Array[i,,k,ensemble] = l$varVals1[i,]
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('Log_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   varVal2Array[i,,k,ensemble] = l$varVals2[i,]
   }
}

for(k in 1:numProbCurves){
   for(i in 1:numAdaptTrials){
   fName = paste('Log_Adapt_Trial',i,'Function',k,'ED',2)
         load(fName)
   numTestArray[i,,k,ensemble] = l$numTests
   }
}

for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('Log_Adapt_Trial',j,'Function',1,'ED',2)
```

```
    load(fName)
    if(j==sampleSizes[1]){
        plot(xx,l$finalP,ylim=c(0,1),type='lines',col=i,main='Comparing
            Sample Size',xlab='Concentration',ylab='Probability of Detection
            ')
        abline(h=.5)
        lines(xx,l$finalPUpper,lty=2,col=i)
        lines(xx,l$finalPLower,lty=2,col=i)
    }else{
        lines(xx,l$finalP,col=i)
        lines(xx,l$finalPUpper,lty=2,col=i)
        lines(xx,l$finalPLower,lty=2,col=i)
    }
}
lines(xx,fx1(xx),lwd=3,col='grey')
legend('topleft',c('24','72','120'),col=c(1,2,3),lwd=c(2,2,2))

####################
##### LOGISTIC #####
### Non-Adaptive ###
####################
ensemble = 4
for(i in 1:numSampleSizes){
    for(j in 1:numSampleSizes){
        fName = paste('Log_Non_Adapt_Trial',sampleSizes[j],'Function',i,'ED
            ',2)
        load(fName)
        biasArray[j,i,ensemble] = abs(l$EDval-trueC50[i])
    }
}


for(k in 1:numProbCurves){
    for(i in 1:numSampleSizes){
    j = sampleSizes[i]
    fName = paste('Log_Non_Adapt_Trial',j,'Function',k,'ED',2)
          load(fName)
    varVal1Array[j,,k,ensemble] = l$varVals1[j,]
    }
}

for(k in 1:numProbCurves){
    for(i in 1:numSampleSizes){
    j = sampleSizes[i]
    fName = paste('Log_Non_Adapt_Trial',j,'Function',k,'ED',2)
          load(fName)
    varVal2Array[j,,k,ensemble] = l$varVals2[j,]
    }
}

for(k in 1:numProbCurves){
    for(i in 1:numSampleSizes){
    j = sampleSizes[i]
    fName = paste('Log_Non_Adapt_Trial',j,'Function',k,'ED',2)
```

```
        load(fName)
   numTestArray[j,,k,ensemble] = l$numTests
   }
}

for(i in 1:numSampleSizes){
   j = sampleSizes[i]
   fName = paste('Log_Non_Adapt_Trial',j,'Function',1,'ED',2)
        load(fName)
   if(i==1){plot(xx,l$finalP,ylim=c(0,1),type='lines',col=i)}else{
   lines(xx,l$finalP,col=i)
   }
   }
lines(xx,fx1(xx),lwd=3)

################################
### Plotting Posterior Curves ###
################################

par(mfrow=c(1,2))
j = sampleSizes[1] # can use 1,2,3 here
probFunc = 1
   fName = paste('GP_Adapt_Trial',j,'Function',probFunc,'ED',2)
   load(fName)
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',lwd=3,col=1,main='Non
         Monotone',xlab='Concentration',ylab='Probability of Detection')
      fName = paste('GP_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalPred,col=2,lwd=3)
      fName = paste('Log_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=3,lwd=3)
      fName = paste('Log_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=4,lwd=3)
      abline(h=.5)
   lines(xx,fx1(xx),lwd=3,lty=2,col='grey')
legend('bottomright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non
   Adapt'),col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

probFunc = 2
   fName = paste('GP_Adapt_Trial',j,'Function',probFunc,'ED',2)
   load(fName)
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',lwd=3,col=1,main='Slow
         Increasing',xlab='Concentration',ylab='Probability of Detection')
      fName = paste('GP_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalPred,col=2,lwd=3)
      fName = paste('Log_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=3,lwd=3)
      fName = paste('Log_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=4,lwd=3)
```

```
      abline(h=.5)
   lines(xx,fx2(xx),lwd=3,lty=2,col='grey')
legend('bottomright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non
   Adapt'),col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

probFunc = 3
   fName = paste('GP_Adapt_Trial',j,'Function',probFunc,'ED',2)
   load(fName)
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',lwd=3,col=1,main='Sharp
          Increasing',xlab='Concentration',ylab='Probability of Detection
         ')
      fName = paste('GP_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalPred,col=2,lwd=3)
      fName = paste('Log_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=3,lwd=3)
      fName = paste('Log_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=4,lwd=3)
      abline(h=.5)
   lines(xx,fx3(xx),lwd=3,lty=2,col='grey')
legend('bottomright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non
   Adapt'),col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

probFunc = 4
   fName = paste('GP_Adapt_Trial',j,'Function',probFunc,'ED',2)
   load(fName)
      plot(xx,l$finalPred,ylim=c(0,1),type='lines',lwd=3,col=1,main='Null
         ',xlab='Concentration',ylab='Probability of Detection')
      fName = paste('GP_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalPred,col=2,lwd=3)
      fName = paste('Log_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=3,lwd=3)
      fName = paste('Log_Non_Adapt_Trial',j,'Function', probFunc,'ED',2)
      load(fName)
      lines(xx,l$finalP,col=4,lwd=3)
      abline(h=.5)
   lines(xx,fx4(xx),lwd=3,lty=2,col='grey')
legend('bottomright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non
   Adapt'),col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))


######################
### Plotting Bias ###
######################

par(mfrow=c(1,1))
ll=c(24,72,120)
plot(ll,biasArray[,1,1],type='lines',lwd=3,ylim=c(0,.2),ylab='Absolute
   Bias',xlab='Sample Size')
lines(ll,biasArray[,1,2],col=2,lwd=3)
```

```
lines(ll,biasArray[,1,3],col=3,lwd=3)
lines(ll,biasArray[,1,4],col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

plot(ll,biasArray[,2,1],type='lines',lwd=3,ylim=c(0,.2),ylab='Absolute
   Bias',xlab='Sample Size')
lines(ll,biasArray[,2,2],col=2,lwd=3)
lines(ll,biasArray[,2,3],col=3,lwd=3)
lines(ll,biasArray[,2,4],col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

plot(ll,biasArray[,3,1],type='lines',lwd=3,ylim=c(0,.2),ylab='Absolute
   Bias',xlab='Sample Size')
lines(ll,biasArray[,3,2],col=2,lwd=3)
lines(ll,biasArray[,3,3],col=3,lwd=3)
lines(ll,biasArray[,3,4],col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lty=c(1,1,1,1),lwd=c(3,3,3,3))

##########################
### Plotting Variance ###
##########################

# look at variance at concentrations
par(mfrow=c(1,1))
criteria = 'mean'
plot(ll,apply(varVal1Array[sampleSizes,,1,1],1,criteria ),xlab='Sample
   Size',ylab='Mean Variance',main='Mean Variances - Non Monotone',type='
   lines',col=1,ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,2,1],1,criteria ),col=2,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,3,1],1,criteria ),col=3,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,4,1],1,criteria ),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

plot(ll,apply(varVal1Array[sampleSizes,,1,2],1,criteria ),xlab='Sample
   Size',ylab='Mean Variance',main='Mean Variances - Slow Increasing',type
   ='lines',col=1,ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,2,2],1,criteria ),col=2,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,3,2],1,criteria ),col=3,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,4,2],1,criteria ),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

plot(ll,apply(varVal1Array[sampleSizes,,1,3],1,criteria ),xlab='Sample
   Size',ylab='Mean Variance',main='Mean Variances - Sharp Increasing',
   type='lines',col=1,ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,2,3],1,criteria ),col=2,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,3,3],1,criteria ),col=3,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,4,3],1,criteria ),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))
```

```
plot(ll,apply(varVal1Array[sampleSizes,,1,4],1,criteria ),xlab='Sample
   Size',ylab='Mean Variance',main='Mean Variances - Null',type='lines',
   col=1,ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,2,4],1,criteria ),col=2,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,3,4],1,criteria ),col=3,lwd=3)
lines(ll,apply(varVal1Array[sampleSizes,,4,4],1,criteria ),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

# looking at variance throughout curve
par(mfrow=c(1,1))
criteria = 'mean'
plot(ll,apply(varVal2Array[sampleSizes,,1,1],1,criteria),xlab='Sample Size
   ',ylab='Mean Variance',main='Mean Variances - Non Monotone',type='lines
   ',col=1,ylim=c(0,.15),lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,2,1],1,criteria),col=2,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,3,1],1,criteria),col=3,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,4,1],1,criteria),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

plot(ll,apply(varVal2Array[sampleSizes,,1,2],1,criteria),xlab='Sample Size
   ',ylab='Mean Variance',main='Mean Variances - Slow Increasing',type='
   lines',col=1,ylim=c(0,.15),lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,2,2],1,criteria),col=2,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,3,2],1,criteria),col=3,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,4,2],1,criteria),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

plot(ll,apply(varVal2Array[sampleSizes,,1,3],1,criteria),xlab='Sample Size
   ',ylab='Mean Variance',main='Mean Variances - Sharp Increasing',type='
   lines',col=1,ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,2,3],1,criteria),col=2,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,3,3],1,criteria),col=3,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,4,3],1,criteria),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))

plot(ll,apply(varVal2Array[sampleSizes,,1,4],1,criteria),xlab='Sample Size
   ',ylab='Mean Variance',main='Mean Variances - Null',type='lines',col=1,
   ylim=c(0,.05),lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,2,4],1,criteria),col=2,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,3,4],1,criteria),col=3,lwd=3)
lines(ll,apply(varVal2Array[sampleSizes,,4,4],1,criteria),col=4,lwd=3)
legend('topright',c('GP Adapt','GP Non Adapt','Log Adapt','Log Non Adapt')
   ,col=c(1,2,3,4),lwd=c(3,3,3,3))
```