A NONLINEAR RESPONSE MODEL FOR SINGLE NUCLEOTIDE POLYMORPHISM DETECTION ASSAYS

by

Drew Philip Kouri

Submitted in partial fulfillment of the requirements

For the degree of Masters of Science

Thesis Adviser: Dr. Peter Thomas

Department of Mathematics

CASE WESTERN RESERVE UNIVERSITY

August, 2008

CASE WESTERN RESERVE UNIVERSITY

SCHOOL OF GRADUATE STUDIES

We hereby approve the thesis/dissertation of

Drew P. Kouri
____

candidate for the Masters of Science degree *.

(signed) Peter J. Thomas
(chair of the committee)

Daniela Calvetti

Peter A. Zimmerman

James Alexander

(date) March 31, 2008

*We also certify that written approval has been obtained for any proprietary material contained therein.

Contents

List of Tables

List of Figures

List of Abbreviations

SNP Single Nucleotide Polymorphism

PCR Polymerase Chain Reaction

LDR Ligase Detection Reaction

FMA Fluorescence Microsphere Assay

NLLS Nonlinear Least Squares

ML Maximum Likelihood

BFGS Broyden, Fletcher, Goldfarb, and Shanno

PNG Papua New Guinea

Pf *Plasmodium falciparum*

WT Wild Type

DR Drug Resistant

dhfr Dihydrofolate Reductase

dsDNA Double Strand DNA

ssDNA Single Strand DNA

iRBCs Infected Red Blood Cells

MFI Median Fluorescence Intensity

$\|\cdot\|_2 = \|\cdot\|$ The Euclidean Norm

A Nonlinear Response Model for Single Nucleotide Polymorphism Detection Assays

Abstract

by

DREW PHILIP KOURI

Malaria is a significant cause of mortality in the tropical regions of the world, such to Papua New Guinea (PNG). Efforts to combat malaria are impeded by the development of drug resistant mutants. We generated a model describing the chemical and molecular properties of the ligase detection reaction (LDR) fluorescent microsphere assay (FMA) for single nucleotide polymorphism (SNP) detection and employed numerical optimization techniques to determine the parameters of this model. First, we implemented the Levenberg-Marquardt Nonlinear Least Squares (NLLS) algorithm and estimated the model parameters from simulated data representing a control dilution/mixing experiment. Second, we used these parameters as well as parameters estimated from experimental data to generate possible distributions of parasite concentration. These distributions can, in principle, inform us of how drug resistance is distributed in a given sample and throughout PNG communities in general. Furthermore, they allow us to evaluate a drug's effectiveness in that community.

1 Introduction

Malaria is an ever-increasing problem in the tropical regions of the world. A total of 41% of the world's population lives in malaria endemic regions. The malaria parasite is responsible for about a million of deaths and up to 500 million infections each year, according to the Center for Disease Control (www.cdc.org). With the increase in the use of anti-malarial drugs, the malaria parasite genome is constantly mutating in order to survive. A mutation at a single nucleotide position in the malarial genome can confer drug resistance. In the dihydrofolate reductase (*dhfr*) gene, there are four such point mutations. Carnevale et al have developed a three part molecular assay for the detection of these drug resistant single nucleotide polymorphisms (SNP). The first step is polymerase chain reaction (PCR), in which the desired gene is amplified following a sequence of steps repeated 35 times. The second part is ligase detection reaction (LDR), in which allele-specific and common primers bind to the PCR product. The final part is fluorescent microsphere assay (FMA). In this procedure, fluorescent microspheres are bound to allele specific oligonucleotide tags and the relative abundance of fluorescence is measured by flow cytometry.

This measurement suffers several limitations. The assay's fluorescent output yields no information about the starting concentration of each allele. Thus, the assay only permits a qualitative (presence/abscence) rather than quantitative analysis of the data. We can determine whether a sample is infected with the wild type (WT) or drug resistant (DR) strain, has a mixed infection, or has no infection, but we cannot determine the relative abundance of each parasite. Also, as with any measurement process, there is an error associated with the output data. In fact, performing the assay on negative controls (i.e. samples with no malarial DNA) yields a nonzero fluorescence measurement, typically about 100 median fluorescence intensity (MFI) units. Furthermore, this background noise has a specific structure. We observe that if only one allele is present in a sample, as the fluorescence for that allele rises, the background signal also rises. This background affects the analysis of the data and complicates determination of infection types.

There have been multiple techniques proposed to analyze a given data set and to understand the background noise. The classic technique is to find the mean and standard deviation of the negative controls and use the mean plus two or three standard deviations to generate two thresholds. These thresholds separate the single strain infections. Carnevale et al proposed a second technique, a heuristic method that uses a fitness function to determine the optimal horizontal and vertical thresholds. Neither of these two methods takes the rising background signal into account. Since the thresholds used in both methods are parallel to the axes, there is the possibility of misdiagnosing single strain infections as mixed infections. We proposed a third method of analyzing this data. Our histogram-based method involves a polar transformation of the data and determines three thresholds: two angular thresholds and a magnitude threshold ([38]). This method, detailed in the Appendix, accounts for the structure of the data and yields a better analysis.

Although our histogram-based method improves the accuracy in the identification of the type of infection in a given sample, it has several limitations. First, it is a strictly phenomenal model, exploiting the structure of naturally occurring clustering in the empirical population field data to make inferences about individual infections. Second, it does not give any insight into the distribution of parasitemias within the infected population. Parasitemia is known to vary by many orders of magnitude during the course of an infection, ranging from levels too low to be reliably detected to levels high enough to cause mortality ([83]).

In order to move beyond a purely phenomenological, data-driven approach, we propose a quantitative model for the molecular assay, which integrates three models describing the three steps of the assay (PCR, LDR and FMA) and has eight parameters that must be estimated. Given estimates of the parameters, we can employ Bayes' Theorem to produce estimates for the parasite concentration distributions of a given sample's fluorescence measurement. The model we propose and the sequence of steps leading to quantitative estimates of individual and population parasitemias are intended to provide a practical tool for improving assessment of drug resistance in the malaria and human populations wherever malaria is found.

Estimating the model parameters requires a two-stage process. First one performs a dilution/mixing experiment starting from laboratory samples of malarial DNA. Two control strains, each known to have either the DR or the WT allele present at locus 59 in *dhfr*, are used to prepare several different dilutions. Then we mix each dilution in the series, generating every combination of dilutions and mixtures. Applying the assay to this control dilution/mixing series, gives us input data (DNA concentrations) and output data (fluorescence intensity measurements) for which to fit our model. To illustrate this procedure we employed simulated data generated by an instance of the model with nominal parameter values. The second stage of the procedure requires numerical estimation of the model parameters from either a real (if available) or simulated data set. We used the Levenberg-Marquardt nonlinear least squares algorithm to estimate the parameters and applied this method to our simulated data set. Given the parameter estimates for the molecular assay model, we apply Bayes' theorem to generate distributions of the likelihoods of different parasitemias given individual fluorescence measurements. These parasitemia distributions in turn allow us to characterize the distribution of parasitemias within the sampled population.

2 Laboratory Methods

Obtained from the Malaria Research and Reference Reagent Resource (MR4, ATCC Manassas, Virginia), the control Pf strains used in this study were 3D7 (MRA-102G) and Dd2 (MRA-150G). These strains were grown in vitro as previously described in McNamara et al. Parasitemia for both the 3D7 (1.98% parasitemia) and Dd2 (2.84% parasitemia) controls was determined using light microscopy (that is, total number of infected red blood sell per microliter divided by the total number of red blood cells per microliter). The concentration of DNA in both controls was also determined via optical density (concentration of DNA in 3D7 was 25.025 g/ml and concentration of DNA in Dd2 was 18.209 g/ml). Seven ten-fold dilutions were performed on the controls. These dilutions are of the form one part control DNA to 10^k parts dH20, where k=0,1,2,3,4,5,6. Each dilution of 3D7 was mixed with each dilution from Dd2 to form every possible combination of DNA mixtures.

For the PCR amplication of these mixtures, all reactions (25 μl) were performed in a buffer containing 3 pmoles of the *dhfr* upstream and downstream primers, 67 mM Tris-HCl, pH 8.8, 6.7 mM MgSO4, 16.6 mM (NH4)2SO4, 10 mM 2-mercaptoethanol, 100 μM dATP, dGTP, dCTP, and dTTP, and 2.5 units of thermostable DNA polymerase. The samples were then amplified in a Peltier Thermal Cycler, PTC-225 (MJ Research, Watertown, MA). The specific primers and thermocycling conditions used to amplify the Pf *dhfr* target sequence were described in Carnevale et al.

The following is a brief description of the post-PCR LDR-FMA process, for a complete description (i.e. specific LDR primers/probes or reaction solutions) see [9]. Following PCR amplification, products were combined into a multiplex LDR where allele-specific upstream primers ligate to conserved sequence downstream primers. Upstream, allele-specific primers include 5' extensions of unique "TAG" sequences. Downstream, conserved sequence primers are modified by 5' phosphorylation and 3' biotinylation. The 5' ends of the LDR products receive "classification" labeling in a second multiplex reaction where hybridization occurs between the TAG sequences added to the allele-specific primers and anti-TAG (complementary sequence) oligonucleotide probes bound to fluorescent microspheres. Following this hybridization reaction, products are incubated in a solution containing streptavidin-R-phycoerythrin (SA-PE) to allow "reporter" labeling through binding to the 3'-biotin on the conserved sequence primers. Detection of doubly-labeled ligation products occurs through dual fluorescence flow cytometry in the Bio-Plex array reader (Bio-Rad Laboratories, Hercules, CA) and leads to collection of "reporter" signal in unique allele-specific bins (see Figure 1).

3 The Model

The molecular typing assay described above was created to determine whether or not the double-stranded DNA (dsDNA) in a sample provides evidence for infection by a drug resistant (DR) or wild type (WT) strain of Pf. Here "or" is used inclusively, giving four possible infection states $WT^{\pm}/DR^{\pm}$. This section of the thesis develops a mathematical model of the molecular reactions occurring during the assay, in order to facilitate more accurate analysis of data from samples collected in the field.

3.1 Polmerase Chain Reaction (PCR)

During PCR, the dsDNA is denatured to form two single stand DNA molecules (ssDNA). Let us denote one ssDNA as s_1 and its complimentary ssDNA as s_2. Then the upstream PCR primer binds to s_1, while the downstream primer binds to s_2. After the primers anneal, the free nucleotides bind to the open nucleotide positions on s_1 and s_2, elongating the two ssDNA/primer complexes to two complete dsDNA molecules. Thus, one dsDNA molecule produces two new dsDNA molecules after each cycle. Repeating this process for 35 cycles, one dsDNA molecule will produce 2^{35} dsDNA copies of the target sequence. Therefore, the theoretical yield of a 35 cycle PCR is $C_0 2^{35}$, where C_0 is the initial number of dsDNA. Since the quantity of each reagent is finite, there is a possibility of depleting one or multiple reagents during the 35 cycles. This depletion creates a saturating nonlinearity. We also notice that as the cycles increase, the total number of dsDNA increases, but the quantity of each reagent decreases, thus the probability of an ssDNA molecule binding to its complement increases, while the probability of an ssDNA molecule binding to a primer decreases. This effect could cause the PCR not to double at each cycle and thus reduce the yield. We could represent this process with the following piecewise linear function:

$$f(x) = \begin{cases} x\alpha^{35} & \text{if } x < C_\theta \\ \\ C_{MAX} & \text{if } x \geq C_\theta \end{cases} \tag{3.1.1}$$

where $\alpha \in (1, 2]$ is the replication factor, C_{MAX} is the maximum output of the PCR, and C_θ is the threshold for which PCR attains C_{MAX}. From this model, we have three parameters, but we notice that C_θ depends on both α and C_{MAX} in the following relation

$$C_\theta = \frac{C_{MAX}}{\alpha^{35}}. \tag{3.1.2}$$

However, this piecewise defined function is an upper bound for the PCR output. As we described above, the probability of creating a new dsDNA molecule decreases as the cycles progress or if the initial concentration is large, thus the PCR curve is lies below (3.1.1). Therefore, we chose to adopt the following Hill function to model the PCR step:

$$f(x) = \frac{C_{MAX}x}{x + C_\theta}, \tag{3.1.3}$$

where C_θ defines the point in which $f(x)$ attains half of its maximum. This model will initially grow at a rate of α^{35} just as equation (3.1.1), then will saturate at $f(x) = C_{MAX}$, as desired.

3.2 Ligase Detection Reaction (LDR)

The LDR phase is much simpler then the PCR phase. As described in the above section, given one dsDNA molecule of the target sequence, the dsDNA denatures forming two ssDNA molecules. The LDR process concerns itself with one ssDNA molecule and not its compliment, thus the common and the allele-specific primers bind to only one ssDNA molecule. After one cycle, one dsDNA molecule forms one LDR product, hence after 32 cycles, one dsDNA theoretically yields 32 LDR products. In practice, this does not always occur. Since we are looking at diallelic SNPs, each the dsDNA molecule only varies at one nucleotide position. Thus, there is a nonzero probability of an allele-specific primer binding to the incorrect ssDNA. With these mismatches occurring, we generate a background "crosstalk" signal (that is, if no dsDNA of one allele is present initially, after the LDR phase, there could nevertheless be LDR product present identified with that allele present). To model this situation, suppose that the probability of any ssDNA molecule binding to a primer is 1, p_1 is the probability of perfect match binding for the drug sensitive allele, and p_2 is the probability of perfect match binding for the drug resistant allele. These probabilities completely define the LDR phase. That is, given PCR product $\vec{x} = [x_1, x_2]^T$, the resulting LDR product is merely a linear combination of these probabilities and the vector $\vec{x}$:

$$g(\vec{x}) = 32 \begin{bmatrix} p_1 & (1 - p_2) \\ (1 - p_1) & p_2 \end{bmatrix} \vec{x}. \tag{3.2.1}$$

3.3 Fluorescent Microsphere Assay (FMA)

In the final step of the assay, fluorescent microspheres hybridize to the allele-specific primers. Each microsphere has multiple binding sites, thus multiple LDR products can bind to one microsphere. The fluorescence is then measured via flow cytometry. If we assume that the microspheres hybridize correctly, we only need to model the measurement process of the flow cytometry. Each LDR product hybridized to a microsphere has a common fluorescence as well as an allele-specific fluorescence. In order to measure the fluorescence, the Bio-plex separates out the allele-specific fluorescences and then measures the common fluorescence from each microsphere. Each microsphere has an associated maximum fluorescence, which creates another saturating nonlinearity. As seen in experimental data, there is a logistic shape to the curve describing the measurement process, thus we can model the FMA step as:

$$h(x) = \frac{\alpha x^p}{x^p + \beta^p}. \tag{3.3.1}$$

In this study, p is taken to be 2. In equation (3.3.1), α is the maximum fluorescence and β is the point in which $h(x)$ attains half of its maximum. Also, there is some measurement error associated with the flow cytometry which we model as additive lognormal noise.

Composing the three models described above we generate the molecular model describing the entire assay. We have to take some care in combining these models because when we run these experiments in the laboratory, we only use a portion of the PCR product for the LDR phase, and a portion of the LDR product for the FMA phase. For the LDR phase, we only require 1 microliter of PCR product from a 25 microliter reaction volume, thus we must divide $f(x)$ by 25. Similarly, for the FMA phase, we only require 1 microliter of LDR product from a 15 microliter reaction volume, thus we must divide $g(x)$ by 15. This yields the following complete molecular model:

$$M(x) = h\left(g\left(f\left(x|\alpha, C_{MAX}\right)/25|p_1, p_2\right)/15|\vec{\beta_1}, \vec{\beta_2}\right) + \vec{\eta} \qquad (3.3.2)$$

where $\vec{\eta} \sim LogN(\vec{\mu}, \vec{\sigma})$ is the additive noise (see figure 3).

4 Simulated Data Generation

As a model varification technique, we generate simulated data from the above model using a first principals parasite concentration distribution. The concentration vectors are drawn from a two dimensional, multimodal distribution. This distribution has four modes, one for each infection state. To select n vectors from this distribution, first generate a list of n uniformly distributed random numbers between 0 and 1. Each infection state has a certain likelihood of occurring. That is, if the probability of being infected by strain 1 is ρ_1 and the probability of being infected by strain 2 is ρ_2, then the probability of having a single strain infection is $\rho_i(1-\rho_j)$ for $i \neq j$, the probability of having a mixed infection is $\rho_1\rho_2$, and the probability of being uninfected is $(1-\rho_1)(1-\rho_2)$. Now, if a sample is in the uninfected state, set its parasite concentration to zero. If a sample is in either of the two single strain infection states, the infected component of the parasite concentration vector is drawn from a uniform distribution on the interval $(0, \alpha]$ for some α, while the uninfected component is set to zero. Finally, if a sample is the mixed state, both vector components, x_1 and x_2, are drawn from a uniform distribution on the interval $(0,\alpha]$ subject to the constraint, $x_1 + x_2 \leq \alpha$. This constraint on the mixed samples is due to the carrying capacity of parasite in the human body. For example, if the average malaria patient dies when parasite concentration reaches 100 infected red blood cells (iRBCs) per microliter (μL), then there can be no mixed infections that cause the parasite concentration to rise above 100 iRBCs per μL. This scheme gives the following probabilities of being in each state:

$$Pr(\vec{x} = [x_1, x_2]^T) = \begin{cases} (1-\rho_1)(1-\rho_2) & \text{if } \vec{x} = \vec{0} \\ \frac{\rho_1(1-\rho_2)}{\alpha} & \text{if } x_2 = 0 \text{ and } x_1 \in (0,\alpha] \\ \frac{\rho_2(1-\rho_1)}{\alpha} & \text{if } x_1 = 0 \text{ and } x_2 \in (0,\alpha] \\ \frac{2\rho_1\rho_2}{\alpha^2} & \text{if } x_1, x_2 \in (0,\alpha] \text{ and } x_1 + x_2 \leq \alpha. \end{cases} \tag{4.0.3}$$

These probabilities give rise to the following parasite concentration probability density function:

$$\begin{aligned} \pi(\vec{x}) = {} & \delta(x_1)\delta(x_2)(1-\rho_1)(1-\rho_2) \\ & + \chi_{(0,\alpha]}(x_1)\delta(x_2)\frac{\rho_1(1-\rho_2)}{\alpha} \\ & + \delta(x_1)\chi_{(0,\alpha]}(x_2)\frac{(1-\rho_1)\rho_2}{\alpha} \\ & + \chi_{\left\{\vec{x}\in(0,\alpha]^2 : \|\vec{x}\|_1 \leq \alpha\right\}}(\vec{x})\frac{2\rho_1\rho_2}{\alpha^2} \end{aligned} \tag{4.0.4}$$

(see figure 2). To generate the simulated data, the random parasite concentration vector is then input into the above molecular model. It should be noted that a similar density function can be employed to generate parasitemia vectors rather than the parasite concentration vectors, but the model does not input parasitemia. In order to use the parasitemia vectors with the molecular model, the parasitemia must be transformed to concentration. That is, parasitemia is defined as the percent of parasite per microliter of blood. On average, there are about 5,000,000 red blood cells per microliter of blood in the human body, thus given a parasitemia $\beta \in [0, 1]$, the resulting concentration is about $5,000,000\beta$.

5 Parameter Estimation

Parameter estimation is an interesting and challenging problem for any model. Having well choosen parameters can shed light on certain physical aspects of the modeled system as well as help generate more realistic results. The first step in parameter estimation is determining the best optimization framework for a given model, then adapting that framework to best fit the model. Once the optimization framework has been chosen, the next step is to decide how to make a globally convergent numerical routine. The parameters to be estimated in the molecular model include the PCR replication factor, the max PCR output, the LDR binding probabilities, the midpoint of the FMA, and the max flourescence signal. In this thesis, the optimization framework used in the first phase of parameter estimation was the Levenberg-Marquardt nonlinear least squares (NLLS) algorithm. This technique is a modification of the quasi-Newton method for nonlinear least squares and is easily adapted to determine parameters for the molecular model. Another possible optimization technique for the first phase is to adapt the NLLS framework to accommodate natural constraints on the parameters.

The typical nonlinear least squares problem is analogous to the minimization of an objective function of the form $f : \mathbb{R}^n \to [0, \infty)$ defined as $f(x) = \frac{1}{2}R(x)^T R(x)$ for the vector $R(x) = [r_i(x) - y_i]$ where r_i is the model evaluated at t_i, y_i is the observed data, and x is the parameter vector. In the two dimensional SNP situation, the previously described objective function formulation is not valid because $R(x)$ is a matrix, not a vector. Therefore, the nonlinear response model requires an adaptation of the classical NLLS scheme. To reformulate the objective function, we define the $f(x) = \frac{1}{2}\left\|R(x)\right\|_F^2$. Using the Frobenius norm is equivalent to stacking both allele n-vectors into one combined $2n$-vector where the first alleles' response function corresponds to the first n elements and the second alleles' response function corresponds to the second n elements, then minimizing the Euclidean norm squared of this $2n$-vector.

5.1 Nonlinear Least Squares

This section follows from "Numerical Methods for Unconstrained Optimization and Nonlinear Equations" by John Dennis Jr. and Robert Schnabel [16].

Suppose M is the nonlinear response model and (t_i, y_i) is the experimental data to which the model needs to be fitted. The main goal is to minimize the distance between $M(t_i)$ and y_i. Defining this distance to be $R(t_i) = M(t_i) - y_i$, then the best fit parameter vector is the solution to the minimization problem:

$$\min_x f(x) = \min_x \frac{1}{2}R(x)^T R(x). \tag{5.1.1}$$

Here x represents the vector of unknown parameters to be fit using the data. For a simple solution to this minimization problem, one can employ the Gauss-Newton method. The Gauss-Newton method makes use of an affine model approximation of

R. The approximate solution to (5.1.1) is then the solution to the following equation:

$$m_c(x) = R(x_c) + J(x_c)(x - x_c) = 0 \tag{5.1.2}$$

where J is the Jacobian of R and x_c is the current step. Linear least squares techniques are employed to solve this equation, yielding the solution:

$$\begin{aligned} x_+ &= x_c - (J(x_c)^T J(x_c))^{-1} J(x_c)^T R(x_c) \\ &= x_c - (J(x_c)^T J(x_c))^{-1} \nabla f(x_c). \end{aligned} \tag{5.1.3}$$

5.1.1 The Levenberg-Marquardt Method

The above method converges rather quickly if the initial parameter guess is close to the actual parameter values, but in general will not converge globally. A simple extension of the Gauss-Newton method is the Levenberg-Marquardt method, which is the Gauss-Newton method modified as a trust region problem (see [53, 82, 23]). The Levenberg-Marquardt nonlinear least squares problem may be reformulated as:

$$\min_x \frac{1}{2} \left\| J(x_c)(x_c - x) + R(x_c) \right\|, \text{ subject to } \|x_c - x\| \leq \Delta_c. \tag{5.1.4}$$

By lemma 6.4.1 in [16], this minimization problem is solved by finding λ_c such that

$$(J(x_c)^T J(x_c) - \lambda_c I)(x_c - x) = -J(x_c)^T R(x_c). \tag{5.1.5}$$

and

$$\left\| -(J(x_c)^T J(x_c) - \lambda_c I)^{-1} J(x_c)^T R(x_c) \right\| = \Delta_c. \tag{5.1.6}$$

Equation (5.1.5) is simply a linear system and thus can be solved using standard linear least squares. The solution to (5.1.4) is the best choice for the trust region step and is updated each iteration. Also, the Levenberg-Marquardt parameter, λ, and the bound on the step size, Δ, are updated each iteration using either a locally optimal hook step or a double dogleg step.

5.1.2 Stopping Criterion

The NLLS algorithm for the molecular model incorporates six stopping tests similar to those presented in [13]: absolute function convergence, relative function convergence, relative gradient convergence, relative step convergence, maximum iteration, and false convergence. The first stopping test simply tells us if the objective function f is minimized, i.e. $f(x_+) < \epsilon_f$. The relative gradient stopping occurs when the gradient of the objective function is about zero, or we have reached a stationary point. This stopping implies that the algorithm has reached an extreme point. In order to determine this, we use the following formula:

$$\max_{1\le i\le n}\left|\frac{\nabla f(x)_i \max\{|x_i|, typx_i\}}{\max\{|f(x)|, typf\}}\right| \le \epsilon_{rg} \tag{5.1.7}$$

where *typx* is the typical size of x and *typf* is the typical size of the function f. Maximum iteration stopping occurs when our algorithm has iterated a maximum number of iterations. The final three convergences are related. Relative function convergence is when the difference between the current function value and the previous function value is small. That is,

$$\frac{f(x_c) - q_c(x_c - H_c^{-1}\nabla f(x_c))}{f(x_c)} \le \epsilon_{rf}, \tag{5.1.8}$$

where H_c^{-1} is the inverse of the current Hessian matrix. A very important convergence test is relative step size test. This criterion tests the size of each step and stops the algorithm if the step size has become significantly small. Formally, a step that satisfies the following expression is considered to be "small enough:"

$$\frac{\max_i |(x_+ - x_c)_i|}{\max_j(|(x_c)_j| + |(x_+)_j|)} \leq \epsilon_{rx}. \tag{5.1.9}$$

As in [13], we only perform stopping criterion (5.1.7), (5.1.8), (5.1.9) when the current function values satisfy:

$$f(x_c) - f(x_+) \leq 2\left[f(x_c) - q_c(x_+)\right], \tag{5.1.10}$$

where q_c is the current quadratic model of the residual function. This tests whether or not the new step yields at most two times the predicted function decrease. Also, we employ a final stopping criterion that test for false convergence test. This test refers to relative function convergence and is only applied if condition (5.1.10) does not hold. In this situation, the algorithm has a possibility to converge to a noncritical point. To prevent this from happening, we set a lower bound, ϵ_F, for both relative function and relative step size convergence. This yields:

$$\epsilon_F \leq \frac{f(x_c) - q_c(x_c - H_c^{-1}\nabla f(x_c))}{f(x_c)} \leq \epsilon_{rf}. \tag{5.1.11}$$

If this test results in a value less than ϵ_F, the algorithm will not terminate. In our algorithm, we set the tolerances to $\epsilon_f = \epsilon_{rg} = \epsilon_{rf} = \epsilon_{rx} = (\epsilon_{machine})^{1/3}$ and $\epsilon_F = (\epsilon_{machine})^{2/3}$.

5.1.3 Scaling

In our nonlinear response model, the typical parameter values range over 12 orders of magnitude, thus it is necessary to scale the parameters during each iteration. If the parameters are not scaled, the algorithm may ignore the smaller parameters. This is a simple addition to the Levenberg-Marquardt method, only changing the problem slightly. Let $\tilde{x} = Dx$ where D is a positive diagonal scaling matrix with the reciprocal of the typical parameter sizes on the diagonal, then we reformulate our problem as:

$$\min_x \frac{1}{2} \left\| J(\tilde{x_c})(\tilde{x_c} - \tilde{x}) + R(\tilde{x_c}) \right\|, \text{ subject to } \|(\tilde{x_c} - \tilde{x})\| \leq \Delta_c. \tag{5.1.12}$$

This minimization problem leads to the following normal equation formulation:

$$(J(\tilde{x_c})^T J(\tilde{x_c}) - \lambda_c D^2)(\tilde{x_c} - \tilde{x}) = -\nabla f(\tilde{x_c}). \tag{5.1.13}$$

This modification merely transforms our trust region from circular to elliptical. The addition of the diagonal scaling matrix also may affect the stopping conditions. The two main stopping conditions that are affected are absolute residual convergence and relative step size convergence. Given the magnitude of the output of the molecular model (at most ≈ 22000), the residual function may be quite large. Large residuals make absolute residual convergence nearly impossible, thus the absolute residual criterion is adjusted to:

$$\frac{f(x_+)}{typf} < \epsilon_f. \tag{5.1.14}$$

In addition, the parameter estimates must be scaled in the relative step size criterion so that no value is overlooked; i.e.

$$\frac{\max_i |D_i(x_+ - x_c)_i|}{\max_j (D_j |(x_c)_j| + |(x_+)_j|)} \leq \epsilon_{rx} \tag{5.1.15}$$

where D_i denotes the i^{th} component of the diagonal of the scaling matrix.

5.2 Global Convergence

In order to obtain appropriate parameters for the molecular model, the optimization algorithm is required to find a global minimum of the residual function. To do this, the algorithm combines two global convergence strategies: a line search and a trust region method. The reason for the combination is that the line search is quick, but less accurate, while the trust region method is not as quick, but more accurate. A routine method for combining these two algorithms is to first perform line search for a set number of iterates (≈ 20), then move to the trust region method. The combined scheme used in the is thesis was presented in [57]. This method first attempts a trust region step. If the trust region step does not reduce the local model of the residual function, the step is rejected and the algorithm performs a line search to determine the new step. The algorithm used in this study has the option of either using a hook step trust region method or a double dog leg trust region method.

5.2.1 Finite Differences Jacobian Approximation

In this thesis, we use a finite differences approximation of the Jacobian matrix of the molecular model. This Jacobian estimate is then used to approximate the Hessian matrix as $J(x_c)^T J(x_c)$ in the Gauss-Newton case and $J(x_c)^T J(x_c) + \lambda_c I$ in the quasi-Newton case.

5.2.2 The Line Search

The line search algorithm is merely a reduction of the (possibly n-dimensional) optimization problem to the one-dimensional optimization problem of finding the minimum of:

$$\phi(\alpha) = f(x_c + \alpha p_c). \tag{5.2.1}$$

To do this, it is logical to first try the full Newton step ($\alpha_c = 1$) and then, if the full Newton step is unsatisfactory, to backtrack along the Newton step to find an optimal value for α_c. This algorithm requires certain conditions to ensure convergence. There are two types of conditions that can be considered for a line search algorithm: the Wolfe conditions and the Goldstein conditions (see [58, 16, 34]). The Wolfe conditions consist of the Armijo condition which ensures sufficient decrease in the objective function and the curvature condition which rules out unacceptably short steps. The Armijo condition is given by the inequality:

$$f(x_c + \alpha_c p_c) \leq f(x_c) + c_1 \alpha_c \nabla f(x_c)^T p_c \tag{5.2.2}$$

where $c_1 = 1 \times 10^{-4}$. The curvature condition requires α_c to satisfy the following inequality:

$$\nabla f(x_c + \alpha_c p_c)^T p_c \geq c_2 \nabla f(x_c)^T p_c \tag{5.2.3}$$

where $c_2 = 0.9$. These conditions are called the weak Wolfe conditions. A slightly stronger version of these conditions (the strong Wolfe conditions) are as follows

$$f(x_c + \alpha_c p_c) \leq f(x_c) + c_1 \alpha_c \nabla f(x_c)^T p_c, \tag{5.2.4}$$

$$\left|\nabla f(x_c + \alpha_c p_c)^T p_c\right| \geq c_2 \left|\nabla f(x_c)^T p_c\right|. \tag{5.2.5}$$

The Goldstein conditions also ensures a sufficient decrease in the step length, but, in addition, requires that the new step is not too short. These conditions are as follows:

$$f(x_c + \alpha_c p_c) \leq f(x_c) + c \alpha_c \nabla f(x_c)^T p_c, \tag{5.2.6}$$

$$f(x_c + \alpha_c p_c) \geq f(x_c) + (1-c)\alpha_c \nabla f(x_c)^T p_c \tag{5.2.7}$$

for $c \in (0, \frac{1}{2})$. Equation (5.2.6) ensures that the Newton step is sufficiently decreased; while equation (5.2.7) ensures that the new step is not too small.

The idea behind the backtracking algorithm mentioned above is to start with the full Newton step and then "backtrack" along the Newton step until a sufficient step value is found. In order to do this, the most recent information about ϕ is used to model ϕ and then find α_c such that the model is minimized. The initial information is $\phi(0) = f(x_c)$ and $\phi'(0) = \nabla f(x_c)^T p_c$. Also, after computing the next step, $\phi(1) = f(x_c + p_c)$ and thus, the quadratic model is

$$m_q(\alpha) = [\phi(1) - \phi(0) - \phi'(0)]\alpha^2 + \phi'(0)\alpha + \phi(0). \tag{5.2.8}$$

To minimize this, first differentiate with respect to α to obtain

$$m_q'(\alpha) = 2[\phi(1) - \phi(0) - \phi'(0)]\alpha + \phi'(0) \tag{5.2.9}$$

and solve for $m_c'(\alpha) = 0$; which has solution

$$\alpha = \frac{-\phi'(0)}{2[\phi(1) - \phi(0) - \phi'(0)]}. \tag{5.2.10}$$

To determine whether or not this solution is a minimum, compute the second derivative

$$m_q''(\alpha) = 2[\phi(1) - \phi(0) - \phi'(0)] > 0, \tag{5.2.11}$$

since $\phi(1) > \phi(0) + \phi'(0)$. In the situation that $\phi(1) >> \phi(0)$, α can be very small and thus suggests that our quadratic model is not a good representation of ϕ. If this occurs, model ϕ with a cubic equation and search for a minimizer. This cubic model is

$$m_c(\alpha) = a\alpha^3 + b\alpha^2 + \phi'(0)\alpha + \phi(0) \tag{5.2.12}$$

where a and b are estimated by additional information. Given α_0 and α_1, the previous two selections of α, determine the cubic and quadratic coefficients as follows

$$\begin{bmatrix} a \\ \\ b \end{bmatrix} = \frac{1}{\alpha_1 - \alpha_0} \begin{bmatrix} \frac{1}{\alpha_1^2} & \frac{-1}{\alpha_0^2} \\ \\ \frac{-\alpha_0}{\alpha_1^2} & \frac{\alpha_1}{\alpha_0^2} \end{bmatrix} \begin{bmatrix} \phi(\alpha_1) - \phi(0) - \phi'(0)\alpha_1 \\ \\ \phi(\alpha_0) - \phi(0) - \phi'(0)\alpha_0 \end{bmatrix}; \tag{5.2.13}$$

which gives the minimization solution

$$\alpha = \frac{-b + \sqrt{b^2 - 3a\phi'(0)}}{3a}. \tag{5.2.14}$$

The α found in this way may lead to a step that is too small or too large. Therefore in case both quadratic and cubic interpolation fail to yield sufficient values of α, we impose upper and lower bounds on α. In the implementation in this study, the upper bound is u=0.5 and the lower bound is l=0.1. These bounds reduce the number of computations necessary and thus make the algorithm more efficient.

5.2.3 The Hook Step

The locally constrained hook step (see [16]) is exactly the solution to the Levenberg-Marquardt adaptation of nonlinear least squares. That is, define the function:

$$s(\mu) = -(J(x_c)^T J(x_c) + \mu I)^{-1} \nabla f(x_c). \tag{5.2.15}$$

The hook step problem is to find a μ such that $\|s(\mu)\| \cong \delta_c$. In other words, the solution to (5.2.15) is the root of the equation:

$$\Phi(\mu) = \|s(\mu)\| - \delta_c = 0. \tag{5.2.16}$$

Consider the local model of (5.2.15)

$$m_c(\mu) = \frac{\alpha_c}{\beta_c + \mu} - \delta_c \tag{5.2.17}$$

with the two parameters α and β. These two parameters are required to satisfy two conditions:

$$m_c(\mu) = \frac{\alpha_c}{\beta_c + \mu} - \delta_c = \Phi(\mu) = \|s(\mu)\| - \delta_c \tag{5.2.18}$$

and

$$\begin{aligned} \frac{d}{d\mu}[m_c(\mu)] &= -\frac{\alpha_c}{(\beta_c + \mu)^2} \\ &= \frac{d}{d\mu}[\Phi(\mu)] \\ &= -\frac{s(\mu)^T (J(x_c)^T J(x_c) + \mu I)^{-1} s(\mu)}{\|s(\mu)\|}, \end{aligned} \tag{5.2.19}$$

which leads to the following parameter values:

$$\alpha = -\frac{(\Phi(\mu) + \delta)^2}{\Phi'(\mu)} \tag{5.2.20}$$

and

$$\beta = -\frac{(\Phi(\mu) + \delta)}{\Phi'(\mu)} - \mu. \tag{5.2.21}$$

As stated above, the root of (5.2.16) is desired. We obtain the solution:

$$\begin{aligned} \mu_+ &= \frac{\alpha}{\delta} - \beta \\ &= \mu - \frac{\|s(\mu)\|}{\delta} \left[\frac{\Phi(\mu)}{\Phi'(\mu)} \right]. \end{aligned} \tag{5.2.22}$$

In order to make this algorithm practical and efficient, only approximate solutions to equations (5.2.16) and (5.2.22) are used. Therefore the algorithm must search for a range of solutions, i.e. $\mu_+ \in [l_+, u_+]$ for some lower and upper bounds l_+ and u_+ updated every iteration, and $\|s(\mu)\| \in [\frac{3\delta}{4}, \frac{3\delta}{2}]$.

5.2.4 The Double Dogleg

The double dogleg method (see [16]) is another algorithm for approximating the solution of the trust region subproblem, similar to the hook step. The basic idea is to find a point $x_+ = x_c + s(\mu_c)$ by approximating the "dogleg" curve $s(\mu)$ with a piecewise linear function connecting the minimizer of the local quadratic model and the full Newton step. The minimizer of the local quadratic model pointing in the steepest descent direction is called the Cauchy point. The double dogleg algorithm chooses x_+ to be the point where the arc of the trust region intersects the line segment connecting the Cauchy point and some point in the Newton direction; that is, $\|x_+ - x_c\| = \delta$. In the case that $\|H_c^{-1}\nabla f(x_c)\| < \delta$, the new step, x_+, is merely the full Newton step.

The algorithm begins with the computation of the full Newton Step and then the computation of the Cauchy point. The minimizer of the local quadratic model is given by:

$$\lambda_* = \frac{\|\nabla f(x_c)\|^2}{\nabla f(x_c)^T H_c \nabla f(x_c)}. \tag{5.2.23}$$

Thus, this solution and the steepest descent direction, $-\nabla f(x_c)$ is used to determine the Cauchy point:

$$x_+^{CP} = x_c - \lambda_* \nabla f(x_c). \tag{5.2.24}$$

If the Cauchy point lies outside of the trust region radius, i.e. $\delta \leq \lambda_* \|\nabla f(x_c)\|$, there is no intersection between the arc of the trust region and the line connecting the Cauchy point and the line in the Newton direction. In this case we take the Cauchy point as a step of length δ in the steepest descent direction. Now, the algorithm must determine what point on the Newton direction will yield an optimal solution. This point is given by

$$N = x_c - \eta H_c^{-1} \nabla f(x_c) \tag{5.2.25}$$

where $\gamma \leq \eta \leq 1$ and γ satisfies $\|s_{CP}\| \leq \gamma \|s_N\| \leq \|s_N\|$. In the above inequality, s_{CP} and s_N are the Cauchy step and the Newton step respectively. Thus, the dogleg curve begins at x_c, travels along the steepest descent direction until it reaches the Cauchy point, changes direction and travels towards N, intersecting the trust region boundary in the process, reaches N and travels along the Newton direction until it reaches the full Newton step. The next step is thus chosen on the line segment between the Cauchy point and N, and has the form:

$$x_+ = x_c + s_{CP} + \lambda(\eta s_N - s_{CP}). \tag{5.2.26}$$

5.3 Maximum Likelihood

Another parameter estimation technique utilized in this study is maximum likelihood. The ML problem is formulated as follows: suppose that $Y = y_i$ is a given data set and that $\{f_\theta | \theta \in \Theta\}$ is some family of probability distributions that generates this data set and $\Theta \subseteq \mathbb{R}^n$ is the parameter space. The goal of ML is to choose parameters such that the likelihood function is maximized. That is, we seek a maximum of the following form:

$$\max_\theta(\mathcal{L}(\theta)) = \max_\theta(f_\theta(y_1, y_2, \ldots)). \tag{5.3.1}$$

Assuming that Y are identically and independently distributed, (5.3.1) reduces to

$$\max_\theta(\mathcal{L}(\theta)) = \max_\theta \prod_i f_\theta(y_i) \tag{5.3.2}$$

and since the maximum is preserved under monotone transformations, a useful trick is to define the log-likelihood as:

$$\max_{\theta}(\mathcal{L}^*(\theta)) = \max_{\theta}(\log(\mathcal{L}(\theta))) = \max_{\theta}\sum_{i}\log(f_\theta(y_i)). \tag{5.3.3}$$

and maximize this resulting equation. In order to solve equation (5.3.2), we find the values of θ for which the derivative of (5.3.3) is equal to zero, then classify these critical values using the second derivative.

Even though the NLLS scheme looks to minimize the objective function and ML looks the maximize the objective function, the same code can be used because finding the maximum of a function f is equivalent to finding the minimum of $-f$.

6 Analysis and Results

In order to use the molecular model to analyze field data, the NLLS parameters must be estimated. To do this, a controlled dilution/mixing experiment should be run using two laboratory adapted strains of Pf of known concentration/parasitemia. This experiment requires ten serial dilutions plus one blank well for each of the two different lab adapted strains. Each strain dilution is then mixed with all eleven of the opposite strain dilutions to generate 121 samples with varying concentrations of each strain (see figure 4). These parameters, along with the molecular model, are then utilized to estimate distributions of parasitemia or parasite concentrations in population data. In order to do this, we employ Bayes' formula along with the maximum likelihood techniques described above. Using "π" to represent probability distributions functions, Bayes' formula is:

$$\pi(\vec{x}|\vec{y}) = \frac{\pi(\vec{x})\pi(\vec{y}|\vec{x})}{\pi(\vec{y})}. \tag{6.0.4}$$

In terms of our inverse problem (see [8, 71]), we can rewrite this equation as:

$$\pi(\vec{x}|\vec{y}) = \frac{\pi_{conc}(\vec{x})\pi_{noise}(\vec{y} - f(\vec{x}))}{\pi(\vec{y})}, \tag{6.0.5}$$

where f is the molecular assay, $\vec{y}$ is the observed fluorescence data, and $\vec{x}$ is the input data. The concentration distribution represents the allele concentration distribution described in the surrogate data section. For this distribution, the probabilities of having one allele present, p_1 and p_2, can be estimated from population data using the histogram method (see Appendix). That is, p_i is equal to the number of samples with the i allele present divided by the total number of samples. The third parameter, α, is estimated later. The noise distribution from (6.0.5) is the bivariate lognormal distribution from the FMA stage of the model. The parameters for the noise distribution can be estimated from control/blank data, by computing the mean and variance of the log of the data.

To estimate α, we employ ML with respect to the noise distribution. In the one dimensional space, the log normal distribution is given by:

$$\rho(z) = \frac{1}{z\sigma\sqrt{2\pi}} \exp\left[\frac{-(\ln(z) - \mu)^2}{2\sigma^2}\right]. \tag{6.0.6}$$

We want to optimize ρ for x and, in order to do so, we will investigate the log likelihood of ρ:

$$\mathcal{L}(z) = \ln(\rho(z)) = -\ln(z\sigma\sqrt{(}2\pi)) - \frac{(\ln(z) - \mu)^2}{2\sigma^2}. \tag{6.0.7}$$

To maximize this function, we must find where the derivative of ϕ is zero, i.e.

$$\mathcal{L}'(z) = -\frac{1}{z} - \frac{\ln(z) - \mu}{z\sigma^2} = 0. \tag{6.0.8}$$

This equation has solutions at $z = \exp(\mu - \sigma^2)$ and $z = \infty$, but if $z = \infty$, then $\mathcal{L}(z) = 0$ and thus not a maximum. Hence, the most likely value for z is the mode, $z = \exp(\mu - \sigma^2)$. For our noise distribution, we have $\vec{z} = \vec{y} - f(\vec{x})$ and thus, the most likely solution for the input data is $\vec{x} = f^{-1}(\vec{y} - \exp(\vec{\mu} - \vec{\sigma}^2)$. With these most likely values of $\vec{x}$, we can compute the third concentration distribution parameter as the maximum of the sum of the elements of $\vec{x_i}$ for each i:

$$\alpha = \max_i \|\vec{x_i}\|_1 \tag{6.0.9}$$

by analogy with the maximum likelihood estimator for the univariate uniform distribution.

Now we have two of the three components of the conditional probability distribution, $\pi(\vec{x}|\vec{y})$. To compute $\pi(\vec{y})$, we notice that

$$\begin{aligned} 1 &= \int_0^\alpha \frac{\pi_{conc}(\vec{x})\pi_{noise}(\vec{y} - f(\vec{x}))}{\pi(\vec{y})} \mathrm{d}\vec{x} \\ &= \frac{1}{\pi(\vec{y})} \int_0^\alpha \pi_{conc}(\vec{x})\pi_{noise}(\vec{y} - f(\vec{x}))\mathrm{d}\vec{x} \end{aligned} \tag{6.0.10}$$

and $\pi(\vec{y})$ is merely the normalization constant for any given $\vec{y}$. Thus,

$$\pi(\vec{y}) = \int_0^\alpha \pi_{conc}(\vec{x})\pi_{noise}(\vec{y} - f(\vec{x})). \tag{6.0.11}$$

With this distribution, the conditional probability distribution is complete and may be utilized to determine the distribution of possible parasitemias associated with any observed fluorescence data $\vec{y_i}$.

6.1 Nonlinear Least Squares Parameter Estimates

We simulated the proposed dilution/mixing experiment by generating a 121 by 2 matrix of concentrations as described above. Suppose that the control DNA, before dilution, has a concentration of 1,000,000 infected red blood cells (iRBCs) per microliter (μL). Then the dilution series of this controlled DNA consists of 10^{-i} times 1,000,000 iRBCs per μL, where $i = 0, 1, ..., 9$. To generate a more realistic dilution experiment each element of the concentration vector has a stochastic term added. This stochastic term is a uniform random number with magnitude one order of magnitude less than the dilution sample. We input this surrogate dilution/mixing experiment into the model using predetermined "reasonable" parameters to generate surrogate fluorescence data. These parameters were chosen by visually fitting the model parameters to locus 59 in the *dhfr* gene from RVL field data from PNG. Using the NLLS algorithm, we fit parameters to the model output fluorescence. The NLLS algorithm terminated after 13 iterations returning a relative residual (i.e. $residual/typf$) of 0.08. The total run time to fit these parameters was 3.93 seconds.

6.2 Conditional Probability Distribution Analysis

Using the techniques described in the surrogate data section, we generated $n = 264$ fluorescence samples based on locus 59 in the *dhfr* gene from RVL field data from PNG. To determine parasitemia distribution parameters, we analyzed the locus 59 RVL field data with the histogram SNP analysis algorithm (see appendix) to determine the number of samples in each infection state. This analysis yielded the following parameters: $p_1 = 0.3144$ and $p_2 = 0.2273$. Also, we estimated the lognormal distribution parameters using the fluorescence output of 70 uninfected samples (no malarial DNA). The maximum likelihood lognormal parameters were $\mu_1 = 4.4153$, $\sigma_1 = 0.4185$, $\mu_2 = 4.6019$, and $\sigma_2 = 0.3543$. See Figure 5. Given the surrogate fluorescence data, along with the NLLS parameters and the log normal parameters, we estimated the third parasite concentration distribution parameter as $\alpha = 110.4945$. Finally, to generate the conditional probability distribution for any given $\vec{y}$, we approximate the $\pi(\vec{y})$ distribution by approximating the integral equation (6.0.10). Figure 6 shows typical conditional probability parasite concentration distributions.

7 Discussion

The overall goal of the project is to provide a framework for improved diagnosis at the individual and population level for drug-resistant malaria mutants.

Double serial dilution control experiments described in Laboratory Methods were carried out by J. DaRe. Unfortunately, technical difficulties developing the stain for prepared gels made it impossible to incorporate empirically determined PCR product measurements. This situation forced us to use surrogate data in place of empirical data in order to give a complete description of the overall data analysis framework.

8 Conclusions

The molecular model described in this paper should prove useful to the biomedical community, particularly to those focused on single nucleotide polymorphism (SNP) detection using the LDR-FMA platform or any microarray SNP platform. With well chosen parameters, community fluorescence data can be analyzed quickly via the inverse of the molecular model, yielding parasitemia distributions for each allele in the drug resistance conferring SNPs. The knowledge of how these SNPs are dispersed throughout a community has great significance in the field of public health and policy. Being able to chart the changes in drug resistance will allow policy makers to change drug treatment plans for that region before the drug becomes obsolete. Also, being able to determine the relative abundance of these drug resistance strains will help to map the evolutionary spread of these point mutations or even determine the rate of mutation. Future directions for this study are to implement a bound constrained optimization algorithm for the NLLS problem. Another direction is obtaining the proposed dilution/mixing experiment data to estimate the biological and chemical parameters of the molecular model. Advancing technology is rapidly introducing many new molecular and laboratory techniques. Providing a framework for interpreting the data sets now becoming available is one of the ongoing challenges at the intersection of applied mathematical and biomedical research today. First principles models and analytic techniques such as those proposed in this paper will potentially have a significant impact on the scientific community and the world population in general.

9 Appendix

9.1 Histogram SNP Analysis Algorithm

The topic of this section has appeared as partial requirement of Biology 388 (advisory Peter J. Thomas) and will be appearing in a scientific journal soon.

There are four infection states at any given diallelic SNP, i.e. uninfected for both alleles, infected with a single allele, or infected with both alleles (00, 01, 10, 11). Given LDR-FMA output data, we would like to generate a principled partition of the data into these four infection states. Previous methods by Carnevale Et. Al. have generated two thresholds using a heuristic technique. These thresholds have been verticle and horizontal lines. Upon inspection of the output data, we notice that that these thresholds do not take into account the intrinsic structure of the data. As signal for one allele increases, the background signal for the other allele also increases. Thus, we see the single strain infections pulling away form the axes. We propose a more robust three threshold technique in which we exploit this intrinsic structure of the data. This technique generates a quarter circle threshold seperating uninfected from infected and two rays that seperate single infection from mixed infection.

The technique consists of six parts: data transformation to polar coordinates, generate histograms, find minima, data tranformation back to Cartesian coordinates, bootstrapping confidence intervals, and estimating total confidence in the thresholds. The first step is transforming the LDR-FMA data into its angle and magnitude components. Let X and Y represent the two alleles, then the magnitude and angle are given by:

$$r = \sqrt{X^2 + Y^2} \tag{9.1.1}$$

$$\theta = \arctan\left(\frac{Y}{X}\right). \tag{9.1.2}$$

We next generate histograms of both the magnitude and angle. To do so, we find the minimum and the maximum magnitudes and angles, $r_- = \min(r)$, $r_+ = max(r)$, $\theta_- = \min(\theta)$, and $\theta_+ = \max(\theta)$. We then generate 100 equally spaced bins for the magnitude and 64 equally spaced bins for the angle. The spacing is given by:

$$r_{spacing} = \left\lceil \frac{r_+ - r_-}{100} \right\rceil \tag{9.1.3}$$

$$\theta_{spacing} = \left\lceil \frac{\theta_+ - \theta_-}{64} \right\rceil. \tag{9.1.4}$$

Now, we employ a simple search algorithm to determine the first minimum after the initial maximum of the magnitude vector, and to determine the first mimimum after the initial maximum and the final minimum before the final maximum of the angle vector. This follows from inspection of the LDR-FMA data. We notice that a majority of the samples are uninfected and thus have small magnitude. This means that there is a relatively large density of samples near about 200 MFI in the magnitude histogram. Similarly, if someone is infected with the allele that lies near the horizontal axis, the angle between the sample and the horizontal axis will be near 0 radians, while if someone is infected with the allele that lies near the verticle axis, the angle between the sample and the verticle axis will be near $\frac{\pi}{2}$ radians. This method generates three numbers. The first is the threshold between infected and uninfected and the second and third are the angles of the rays that separate the single strain infections from the mixed infections. To transform these numbers into thresholds, we have:

$$y_r(x) = \sqrt{\tilde{r}^2 - x^2} \tag{9.1.5}$$

$$y_{\theta_i}(x) = x\tan(\tilde{\theta}_i) \tag{9.1.6}$$

where $\tilde{r}$ and $\tilde{\theta}_i$ are the thresholds determined above.

In order to generate confidence intervals for this method, we employ a statistical bootstrapping method in which we resample the LDR-FMA data allowing for replacement and run the histrogram threshold detection algorithm on the resampled data. We repeat this process 1000 to 10000 times and store the threshold estimates from each run. If $(1-\alpha)$ is our desired confidence interval, then we use the α and $(1-\alpha)$ quantiles from each list of parameter estimates as the confidence intervals. With these confidence estimates on the thresholds, we can determine our total confidence in algorithm. Suppose $\rho(r,\theta)$ is the density of infection and $f(r,\theta)$ is the confidence as a function of position, then the probability of misclassification is:

$$Pr = 1 - \int_0^\infty \int_0^{\frac{\pi}{2}} \rho(r,\theta) f(r,\theta) r \mathrm{d}r \mathrm{d}\theta. \tag{9.1.7}$$

From this, we can approximate the total confidence (TC) by descritising for each sample in the LDR-FMA data. That is,

$$TC = 1 - PR \approx \frac{1}{N}\sum_{i=1}^{N} \alpha_i \rho(r_i, \theta_i) \tag{9.1.8}$$

where α_i is the confidence interval in which the i^{th} sample lies and N is the number of samples. In analyzing population, we determined total confidence of about 98 to 99% in our method.

9.2 Thermodynamic Analysis

One source of background in the SNP detection assay is due to mismatched binding of the allele specific LDR primers. For each locus, there are two primers (since we are investigating diallelic SNPs) and these primers only differ at one nucleotide. Thus, suppose that the primer is n nucleotides in length, then $n-1$ of the nucleotides are the same in each primer. This allows for an increased probability of mismatched binding. In order to investigate these mismatches, one can investigate the thermodynamic properties of Watson-Crick and non Watson-Crick basepairing. This method is a rather simple method based on properties of Gibbs equation:

$$\Delta G(T)^{\circ} = -RT \ln \left(\frac{C}{K} \right) \tag{9.2.1}$$

where T is the temperature, R is the Boltzmann constant, C is the concentration, and K is the equilibrium constant. By rearranging this equation and solving for C, we can determine the probability of binding. Therefore,

$$C = K \exp \left[\frac{-\Delta G^{\circ}}{RT} \right] \tag{9.2.2}$$

gives the probability of a single nucleotide binding to another. In the two dimensional locii case, there are two probabilities of binding (excluding the case in which neither primer binds). These two cases are perfect match (PM) and mismatch (MM). In the PM situation, the correct primer binds to the PCR product, in the MM situation the incorrect primer binds to the PCR product. Using the above equations, we can calculate the probabilities of binding for both the MM and the PM case. There have been numerous nucleic acids research papers written on the Gibbs free energy,

$\Delta G(T)^\circ$, associated with Watson-Crick and non Watson-Crick basepairing [1, 3]. One can use values for the Gibbs free energy as determined in these papers to determine these probabilities. By taking the ratios of the probabilities of PM and MM, one determines the probability of the difference of the two situations. That is, if C_{PM} is the probability of PM and C_{MM} is the probability of MM, then:

$$\frac{C_{PM}}{C_{MM}} = \frac{\exp\left[\frac{-\Delta G^\circ_{PM}}{RT}\right]}{\exp\left[\frac{-\Delta G^\circ_{MM}}{RT}\right]} = \exp\left[\frac{-(\Delta G^\circ_{PM} - \Delta G^\circ_{MM})}{RT}\right]. \tag{9.2.3}$$

Relating this value with the SNP detection assay, for a two dimensional locus, C^v_{PM}/C^v_{MM} (where C^v_i is the probability of binding for $i = MM, PM$ along the vertical axis) is an approximation of the slope of the fluorescence of the allele along the vertical axis; while, C^h_{MM}/C^h_{PM} (where C^h_i is the probability of binding for $i = MM, PM$ along the horizontal axis) is an approximation of the slope of the fluorescence of the allele along the horizontal axis.

9.3 Optimization Matlab Code

9.3.1 Levenberg-Marquardt Driver

```
% LEVENBERG-MARQUARDT IMPLEMENTATION OF NONLINEAR LEAST SQUARES
% DREW KOURI
% INPUT: function, parameter guess, trust region parameters, and tolerance
% parameters
%
% OUTPUT: best fit parameters, best fit residual, termination code, and
% time
%
% DESCRIPTION: Employ both linesearch and trust region techniques to
% optimize the objective function defined as follows:
%
% m(xi|t) = model
% t = parameters
% (xi,yi) = data
% R(t) = m(xi|t)-yi residual function
% f(t) = 1/2 R(t)*R(t) objective function
%
% J = Jacobian Matrix
% g = -J*R(t) Gradient Vector
% H = J*J Hessian Matrix
%
% Find x+ such that (H - L D^2) x+ = -g, where D is scaling matrix, and L
% L is the Levenberg-Marquardt Parameter
% -----------------------------------------------------------------------


% -----------------------------------------------------------------------
% SECTION 1: INITIALIZE SYSTEM AND INTRODUCTORY DISPLAY


clear all       % Clear all saved data
close all       % Close all open windows
clc             % Clear the home screen
```

```
format long g   % Output format

macheps = MACHINEPS(); % Compute Machine Epsilon

% Introduction Display
disp('Nonlinear Least Squares Parameter Fitting')
disp('Levenberg-Marquardt Algorithm')
disp('Drew Kouri')
disp('...')
% -------------------------------------------------------------------------

% -------------------------------------------------------------------------
% SECTION 2: USER DEFINED INPUTS

% Input Initial Parameter Guess
x0 = [2;2e12;1;1;1e4;1e4;1e10;1e10;4;4];
% dataSize = 2*121;

ds = 264;
dataSize = 2*ds;
% Input Function to be Optimized
[M,f,R,y,t,M1] = assayFunc(.4,.3,100,ds);

% Choose Global Step Type
% Hook Step = 1
% Double Dogleg = 2
% Line Search = 3 only if scheme = 2
steptype = 1;

% Choose Global Scheme
% Combined = 1
% Single = 2
scheme = 1;
```

```
% Choose Problem Type
% General Optimization = 1
% Least Squares = 2
probtype = 2;

% Choose Cholesky Factorization Method
% Revised Perturbed Cholesky Factorization = 1
% Standard Perturbed Cholesky Factorization = 2
choltype = 1;

% Tolerance Parameters
maxiter = 400;               % Number of Iterations
residtol = macheps^(1/3);    % Residual Tolerance
relresidtol = macheps^(1/3); % Residual Tolerance
gradtol = macheps^(1/3);     % Gradient Tolerance
steptol = macheps^(1/3);     % Stepsize Tolerance
falsetol = macheps^(2/3);    % False Convergence Tolerance
typx = [1;1e12;1;1;1e4;1e4;1e10;1e10;1;1];
typf = 1e8;                  % Typical f(x) size

% Trust Region Parameters
del = -1;     % Initial Trust Region Radius
lam = 0;      % Initial Levenberg-Marquardt Parameter
delprev = 0;  % Initial Previous TR Radius
phi = 0;      % Initial Phi Estimate
phipr = 0;    % Initial Phi' Estimate
% -----------------------------------------------------------------------------


% -----------------------------------------------------------------------------
%SECTION 3: MINIMIZATION
tic
[x,termcode,m,i,ls]=minDriver(f,x0,scheme,steptype,probtype,...
                              choltype,maxiter,residtol,relresidtol,...
                              gradtol,steptol,falsetol,typx,typf,del,...
                              lam,delprev,phi,phipr,R,M);
toc
% ---------------------------------------------------------------------------


% ---------------------------------------------------------------------------
% SECTION 4: DISPLAY OUTPUT


% Display Termination Code
disp(strcat('Total # of Iterations--',num2str(i)))
disp(strcat('Total # of Line Search Iterations--',num2str(ls)))
disp(strcat('NLLS Fit Residual=',num2str(m)))
disp(strcat('NLLS Fit Relative Residual=',num2str(m/typf)))
switch termcode
    case 1
        disp(strcat('termination code #',num2str(termcode),...
            ': residual less than residtol'));
    case 2
        disp(strcat('termination code #',num2str(termcode),...
            ': relative residual less than relresidtol'));
    case 3
        disp(strcat('termination code #',num2str(termcode),...
            ': norm scaled gradient less than gradtol'));
    case 4
        disp(strcat('termination code #',num2str(termcode),...
            ': scaled distance between last two steps less than steptol'));
    case 5
        disp(strcat('termination code #',num2str(termcode),...
            ': iteration limit exceeded'));
    case 6
        disp(strcat('termination code #',num2str(termcode),...
            ': algorithm converging to a noncritical point'));
end
disp('...')
```

```
% Display Parameters
P = ['Rep Factor:     '
     'Max PCR:        '
     'Binding Prob 1: '
     'Binding Prob 2: '
     'Max FMA 1:      '
     'Max FMA 2:      '
     'Midpoint FMA 1: '
     'Midpoint FMA 2: '
     'FMA Noise 1:    '
     'FMA Noise 2:    '];


disp('               Best Fit Values')
disp([P num2str(x,5)])
% -------------------------------------------------------------------------

% -------------------------------------------------------------------------
% SECTION 5: PLOTTING

xlog = x;
xlog(9) = log(xlog(9))+.8106^2/2;
xlog(10) = log(xlog(10))+.5696^2/2;
y1 = M1(xlog);
figure
scatter(y(1:dataSize/2),y(dataSize/2+1:dataSize),'bo')
hold on
scatter(y1(1:dataSize/2),y1(dataSize/2+1:dataSize),'r.')
xlabel('ALLELE 1')
ylabel('ALLELE 2')
legend('Experimental Data','Surrogate Data')
% -------------------------------------------------------------------------
```

9.3.2 Minimization Function

```
function [x,termcode,m,i,ls]=minDriver(f,x0,scheme,steptype,probtype,...
    choltype,maxiter,residtol,relresidtol,gradtol,steptol,falsetol,...
    typx,typf,del,lam,delprev,phi,phipr,R,M)
% Minimization Driver
% Drew Kouri
% minDriver runs a combined trust region/linesearch algorithm to fine the
% minimum of a nonlinear equation.
% ------------------------------------------------------------------------
% Required Input:
% f = Objective Function
% R = Residual Function
% ------------------------------------------------------------------------
% Optional Input:
% scheme = Global Scheme: 1=Combined; 2=Single
% steptype = Global Step Type: 1=Hook Step; 2=Double Dogleg; 3=Linesearch
% probtype = Problem Type: 1=General Minimization; 2=Least Squares
% choltype = Cholesky Factorization Type: 1=New Method; 2=Standard Method
% maxiter = Iteration Limit
% residtol = Residual Tolerance
% relresidtol = Relative Residual Tolerance
% gradtol = Gradient Tolerance
% steptol = Step Size Tolerance
% falsetol = False Convergence Tolerance
% typx = Typical x Size
% typf = Typical f Size
% del = Trust Region Parameter
% lam = Levenberg-Marquardt Parameter
% delprev = Previous Trust Region Parameter
% phi = estimate of phi
% phipr = estimate of phi'
% M = Model
% x0 = Initial Parameter Guess
% ------------------------------------------------------------------------
% Output:
% x = Solution to Mimization Problem
% termcode = Termination Code
% m = Objective Function Value
% i = Iteration Count
% ls = linesearch iteration count
% ------------------------------------------------------------------------

numParam = length(x0); % Compute Number of Parameters
macheps = MACHINEPS(); % Compute Machine Epsilon

% Check Input Parameters
if(nargin<20), probtype = 1;                end
if(nargin<19), phipr = 0;                   end
if(nargin<18), phi = 0;                     end
if(nargin<17), delprev = 0;                 end
if(nargin<16), lam = 0;                     end
if(nargin<15), del = -1;                    end
if(nargin<14), typf = 1;                    end
if(nargin<13), typx = ones(numParam,1);     end
if(nargin<12), falsetol = macheps^(2/3);    end
if(nargin<11), steptol = macheps^(1/3);     end
if(nargin<10), gradtol = macheps^(1/3);     end
if(nargin<9),  relresidtol = macheps^(1/3); end
if(nargin<8),  residtol = macheps^(1/3);    end
if(nargin<7),  maxiter = 400;               end
if(nargin<6),  choltype = 2;                end
if(nargin<5),  probtype = 1;                end
if(nargin<4),  steptype = 1;                end
if(nargin<3),  scheme = 1;                  end

% Cannot Use combined method with line search
if(scheme == 1 && steptype == 3)
    scheme = 2;
```

```
end

% For NLLS problems
if(probtype==2)
    dataSize = length(R(x0)); % Compute Length of Data Vector
    % Quadratic Model of f - Objective Function
    mq = @(x1,x2,J) f(x1)+(x2-x1)'*J'*R(x1)+1/2*(x2-x1)'*J'*J*(x2-x1);
end

m = f(x0);                                  % Compute Initial Residual
D = diag(1./typx);                          % Scaling Matrix
maxstep = 1e3*max(norm(D*x0),norm(1./typx));% Max Step Tolerance

% Initial Iteration Counts
i = 0;        % Initialize Iteration Number
ls = 0;       % Total Line Search Iterations
retcode = 0;  % Initialize Return Code for Global Optimizers

% Compute Initial Gradiant via Finite Differences
g = finiteDifferenceGradient(numParam,x0,f(x0),f,typx,1e-7);
x = x0; % Set Optimization Variable to Initial Guess

% Decide Whether or Not to Stop Algorithm
termcode = UMSTOP0(x0,f(x0),g,diag(D),typf,gradtol);

% ---------------------------------------------------------------------
while(termcode == 0)
    i  = i+1; % Update Iteration Count

    % Either Compute Finite Difference Gradiant/Hessian
    % or Compute the NLLS Gradiant/Hessian
    if(probtype == 2) % Compute Gradient/Hessian via Jacobian Approximation
        J = finiteDifferenceJacobian(dataSize,numParam,x,M(x),M,typx,1e-7);
        g = J'*R(x);          % Compute Gradiant Vector Approximation
        H = J'*J;             % Compute Hessian Matrix Approximation
    else % Compute Gradient/Hessian via Finite Differences
        g = finiteDifferenceGradient(numParam,x,f(x),f,typx,1e-7);
        H = finiteDifferenceHessian(numParam,x,f(x),f,typx,1e-7);
    end

    % Compute Cholesky Decomposition of H+D
    L = choleskyDecomp(H,choltype,0);

    d = cholsolve(L,g);   % Determine Full Newton Step
    xp = x;               % Store Previous Step

    % Compute Best Step
    if(scheme == 1) % Use Combined Trust Region/Line Search Method
        [x,del,lam,delprev,phi,phipr,ls] = globaldriver(d,x,m,f,H,L,...
            g,D,del,lam,i,numParam,delprev,phi,phipr,maxstep,ls,...
                steptype);
    else % Use Single Global Method
        if(steptype == 1)
            % Use Trust Region Hooke Step Algorithm
            [x,del,lam,delprev,phi,phipr] = hookdriver(d,x,m,f,H,L,g,D,...
                del,lam,i,numParam,delprev,phi,phipr,maxstep);
        elseif(steptype == 2)
            % Use Trust Region Double Dogleg Algorithm
            [x,m,retcode,maxtaken,del] = dogdriver(numParam,x,f(x),f,g,...
                L,H,d,D,maxstep,del);
        else
            % Use Linesearch Backtracking Algorithm
            x = linesearch(x,g,d,f,typx);
        end
    end

    % Compute New Residual
    m = f(x);
```

```
    % Decide Whether or Not to Stop Algorithm
    if(probtype == 2) % For NLLS
        termcode = UMSTOPNLLS(x,xp,m,M,f,mq,g,J,i,D,typf,typx,residtol,...
            maxiter,relresidtol,falsetol,gradtol,steptol);
    else % For General Optimization
        termcode = UMSTOP(x,xp,f(xp),g,typx,typf,retcode,gradtol,steptol,...
            i,maxiter);
    end
end
```

9.3.3 Assay Function

```
function [M,f,R,y,t,M1] = assayFunc(p11,p12,p21,z)

% ------------------------------------------------------------------------

% DESCRIPTION: Generate surrogate data from molecular model.

% ------------------------------------------------------------------------

% INPUT:

% p11 = probability of being uninfected

% p12 = probability of being infected by allele 1

% p21 = probability of being infected by allele 2

% z   = number of samples

% ------------------------------------------------------------------------

% OUTPUT:

% M  = stochastic model function

% f  = objective function

% R  = residual function

% y  = surrogate fluorescence data

% t  = surrogate parasitemia data

% M1 = mean value model function

% ------------------------------------------------------------------------


% ------------------------------------------------------------------------

% Dilution/Mixing Experiment

% z=121;

% d = 1e6.*10.^(-(1:10)+1);

% d(11)=0;

% d=d';

% d = flipud(d);

% for i = 1:11

%     t1((i-1)*11+1:i*11)=d(i);

% end

% t2 = [d;d;d;d;d;d;d;d;d;d;d];

% t = [t1',t2];

%

% for i = 1:121
%     for j = 1:2
%         t(i,j)=t(i,j)+1e-1*t(i,j)*rand();
%     end
% end

% ------------------------------------------------------------------------
% Parasitemia Distribution
t1 = zeros(z,1);
t2 = zeros(z,1);
r = rand(z,1);
for j = 1:z
    if(r(j)<p11)
        t1(j) = 0;
        t2(j) = 0;
    elseif(r(j)>p11 && r(j)<p12+p11)
        t1(j) = 100*rand;
        t2(j) = 0;
    elseif(r(j)>p12+p11 && r(j)<p21+p12+p11)
        t1(j) = 0;
        t2(j) = 100*rand;
    else
        s = rand;
        t1(j) = s*100*rand;
        t2(j) = (1-s)*100*rand;
%             t1(j) = 100*rand;
%             t2(j) = 100*rand;
    end
end
t = [t1,t2];

% ------------------------------------------------------------------------
% Generate Assay Model
% PCR
PCR = @(x,b) b(2)*x./(x+b(2)/b(1)^35);
```

```
% LDR

LDR = @(x,p) (32*[p(1),(1-p(2));(1-p(1)),p(2)]*x')';


% FMA for stochastic model

FMA = @(x,b) [b(1)*x(:,1).^2./(x(:,1).^2+(b(3))^2),...

    b(2)*x(:,2).^2./(x(:,2).^2+(b(4))^2)];

% FMA for mean value model

FMA2 = @(x,b) [b(1)*x(:,1).^2./(x(:,1).^2+(b(3))^2);

               b(2)*x(:,2).^2./(x(:,2).^2+(b(4))^2)];


% Stochastic model

Assay = @(x,b) FMA(LDR(PCR(x,[b(1);b(2)])/25,[b(3);b(4)])/15,...

    [b(5);b(6);b(7);b(8)])+[lognrnd(b(9),.8106,z,1),lognrnd(b(10),...

    .5696,z,1)];

% Mean value model

mAssay = @(x,b) FMA2(LDR(PCR(x,[b(1);b(2)])/25,[b(3);b(4)])/15,...

    [b(5);b(6);b(7);b(8)])+[repmat(exp(b(9)+.8106^2/2),z,1);...

    repmat(exp(b(10)+.5696^2/2),z,1)];


% Important functions

M1 = @(b) Assay(t,b); % Stochastic model with input initial data

M = @(b) mAssay(t,b); % Mean value model with input initial data


% Parameter definitions

repfactor = 1.95;

maxPCR = 1.806e12;

bindProb1 = .94;

bindProb2 = .92;

maxFMA1 = 9000;

maxFMA2 = 11000;

midFMA1 = 2.3e10;

midFMA2 = 2.5e10;

FMAnoise1 = 4.3158;
```

```
FMAnoise2 = 4.7504;


param = [repfactor
         maxPCR
         bindProb1
         bindProb2
         maxFMA1
         maxFMA2
         midFMA1
         midFMA2
         FMAnoise1
         FMAnoise2];


% Stack allele vectors
y1 = M1(param);
y = [y1(:,1);y1(:,2)];


% --------------------------------------------------------------------------
% Output functions
R = @(b) M(b) - y;       % Residual function
f = @(x) 1/2*R(x)'*R(x); % Objective function
```

9.3.4 Initial Stopping Conditions

```
function termcode = UMSTOP0(x0,f0,g,D,typf,gradtol)


if(max(g.*max(x0,1./D)/max(abs(f0),typf))<=1e-3*gradtol)

    termcode = 3;

else

    termcode = 0;

end
```

9.3.5 General Stopping Conditions

```
function termcode = UMSTOP(xc,x,f,g,typx,typf,retcode,gradtol,steptol,...

                           icount,maxiter)

termcode = 0;

if(retcode==1)

    termcode = 3;

elseif(max(abs(g).*max(abs(x),1./typx)/max(abs(f),typf))<=gradtol)

    termcode = 1;

elseif(max(abs(x-xc)./max(abs(x),1./typx))<=steptol)

    termcode = 2;

elseif(icount>=maxiter)

    termcode = 4;

end
```

9.3.6 NLLS Stopping Conditions

```
function termcode = UMSTOPNLLS(x,xp,m,M,f,mq,g,J,i,D,typf,typx,residtol,...

    maxiter,relresidtol,falsetol,gradtol,steptol)

termcode = 0;

% Stopping Criterion

if(1/typf*m<residtol)

    termcode = 1; % Absolute Residual

elseif(i>=maxiter)

    termcode = 5; % Maximum Iterations

elseif(f(xp)-f(x)<=2*(f(xp)-mq(xp,x,J)))

    H = pinv(J'*J);

    if((f(xp)-mq(xp,xp-H*g,J))/f(xp)<relresidtol)

        if((f(xp)-mq(xp,xp-H*g,J))/f(xp)>falsetol)

            termcode = 2; % Relative Residual

        end

    end

    if(max(abs(g)*max(max(abs(x),typx))/max(max(abs(M(x)),typf)))<gradtol)

        termcode = 3; % Norm Scaled Gradient

    end

    if(max(abs(D*(x-xp)))/max(D*(abs(x)+abs(xp)))<steptol)

        termcode = 4; % Relative Step Size

    end

else

    if(max(abs(D*(x-xp)))/max(D*(abs(x)+abs(xp)))<falsetol)

        termcode = 6; % False Convergence

    end

end
```

9.3.7 Global Step Driver

```
function [xp,del,lam,delprev,phi,phipr,ls] = ...

globaldriver(sN,xc,fc,f,H,L,g,D,del,lam,i,n,delprev,phi,phipr,maxstep,ls,steptype)

function [xp,del,lam,delprev,phi,phipr,ls] = globaldriver(sN,xc,fc,f,H,...

    L,g,D,del,lam,i,n,delprev,phi,phipr,maxstep,ls,steptype)

% Determine New Step, trust region radius, and Levenberg-Marquardt

% Parameter

% -------------------------------------------------------------------------

% Input:

% sN = Newton Step

% xc = current x

% fc = f(xc)

% f = objective function

% H = Hessian Matrix

% L = lower triangular Cholesky Decomposition of H

% g = gradient

% D = scaling matrix

% del = trust region radius

% lam = Levenberg-Marquardt Parameter

% i = iteration

% n = length of x

% delprev = previous trust region radius

% phi = estimate of phi

% phipr = estimate of phi'

% ls = linesearch iteration count

% steptype = 1 if Hook Step, 2 if Double Dog Leg

% -------------------------------------------------------------------------

% Output:

% xp = new step

% del = new trust region radius

% lam = new Levenberg-Marquardt Parameter

% delprev = previous trust region radius

% phi = new estimate of phi

% phipr = new estimate of phi'
% ls = linesearch iteration count
% ---------------------------------------------------------------------------

xpprev = xc;   % Define xpprev
fpprev = fc;   % Define fpprev
Dvec = diag(D);% Store scaling matrix diagonal
retcode = 4;   % Initialize Return Code
Newtlen = norm(D*sN); % Determine Newton Length

if(steptype == 1)
    phiprinit = 0; % Initial Phi' Estimate
    firsthook = 1; % First Hook Step
    % Determine Initial trust region radius
    if(i == 1 || del == -1)
        lam = 0;
        if(del == -1)
            alpha = norm(1./Dvec.*g)^2;
            beta = 0;
            for j = 1:n
                temp = sum(L(j:n,j).*g(j:n)./(Dvec(j:n).^2));
                beta = beta + temp*temp;
            end
            del = alpha*alpha^(1/2)/beta;
            if(del>maxstep)
                del = maxstep;
            end
        end
    end
else
    firstdog = 1;
end

% Compute new step, trust region radius, and Levenberg-Marquardt Parameter
while(retcode>=2)
```

```
    if(steptype == 1)

        % Use Hook Step

        [s,del,lam,phi,phipr,firsthook,phiprinit,Newttaken] = hookstep(...

            n,g,L,H,sN,D,Newtlen,delprev,del,lam,phi,phipr,firsthook,...

            phiprinit);

    else

        % Use Double Dog Leg

        if(firstdog == 1)

            ssD = zeros(n,1);

            v = zeros(n,1);

            Cauchylen = 0;

            eta = 0;

        end

        [del,firstdog,Cauchylen,eta,ssD,v,s,Newttaken] = dogstep(n,g,L,...

            sN,D,Newtlen,maxstep,del,firstdog,ssD,v,eta,Cauchylen);

    end

    % Decide whether to use Line Search and Update trust region radius

    if(f(xc+s)>=f(xc))

        xp = linesearch(xc,g,s,f,1./Dvec); % Use Linesearch Backtracking Algorithm

        del = max(norm(xp-xc),0.75*del);

        ls = ls+1;

        retcode = 1;

    else

        delprev = del;

        [xp,fp,maxtaken,del,retcode,xpprev,fpprev] = trustregup(n,xc,fc,...

            f,g,s,D,Newttaken,maxstep,H,L,del,retcode,xpprev,fpprev,...

            steptype);

    end

end
```

9.3.8 Line Search Code

```
function xp = linesearch(xc,dfc,pc,f,typx)

% Line Search Algorithm

% xc = current step

% dfc = current gradient value

% pc = Quasi-Newton step

% f = objective function


% Tolerance Parameters

maxstep = 1e3*max(norm(xc),1); % Max Step Size

steptol = eps^(2/3); % Step tolerance

alpha = 1e-4;

n = length(xc);


% Find Current Values

fc = f(xc); % Current function value

Newtlen = norm(pc); % Newton Step Length

initslope = dfc'*pc; % Initial Slope

rellength = max(abs(pc)./max(abs(xc),1)); % Relative Step Length

minlambda = steptol/rellength; % Minimum Line Search Value


% Update Newton Length based on Maximum Step Length

if(Newtlen>maxstep)

    pc = pc*(maxstep/Newtlen);

    Newtlen = maxstep;

end


lambda = 1; % Initial Line Search Value

retcode = 0; % Stopping Condition

while(retcode == 0)

    xp = xc+lambda*pc; % Determine New Step

    fp = f(xp); % Determine New Objective Function Value

    % Line Search Using Directional Derivative

    if(fp <= fc+alpha*lambda*initslope)
```

```
dfp = finiteDifferenceGradient(n,xp,f(xp),f,typx,1e-7);

beta = 0.9;

newslope = dfp'*pc; % Determine New Slope

if(newslope < beta*initslope)

    if(lambda == 1 && Newtlen < maxstep)

        maxlambda = maxstep/Newtlen; % Update Max Lambda

        stopcode = 0;

        while(stopcode == 0)

            lprev = lambda;

            fpprev = fp;

            lambda = min(2*lambda,maxlambda);

            xp = xc+lambda*pc; % Update Step

            fp = f(xp); % Updata Function Value

            if(fp <= fc+alpha*lambda*initslope)

                dfp = finiteDifferenceGradient(n,xp,f(xp),f,typx,1e-7);

                newslope = dfp'*pc; % Update Slope

            end

            % Exit Loop Conditions

            if(fp <= fc+alpha*lambda*initslope)

                stopcode = 1;

            elseif(newslope < beta*initslope)

                stopcode = 2;

            elseif(lambda < maxlambda)

                stopcode = 3;

            end

        end

    end

    if((lambda < 1) || (lambda>1 && fp > fc+alpha*lambda*initslope))

        llo = min(lambda,lprev); % Lower Estimate of Lambda

        ldiff = abs(lprev-lambda); % Change in Lambda

        if(lambda < lprev)

            flo = fp; % Lower Estimate of Objective Function

            fhi = fpprev; % Upper Estimate of Objective Function

        else
```

```
                flo = fpprev; % Lower Estimate of Objective Function
                fhi = fp; % Upper Estimate of Objective Function
            end
            while(newslope < beta*initslope && ldiff > minlambda)
                % Increment Lambda
                lincr = (-newslope*ldiff^2)/(2*(fhi-(flo+newslope*ldiff)));
                if(lincr < 0.2*ldiff)
                    lincr = 0.2*ldiff; % Update Lambda Increment
                end
                lambda = llo+lincr; % Update Lambda
                xp = xc+lambda*pc; % Update Step
                fp = f(xp); % Update Objective Function
                if(fp > fc+alpha*lambda*initslope)
                    ldiff = lincr; % Change in Lambda
                    fhi = fp; % Upper Estimate of Objective Function
                else
                    dfp = finiteDifferenceGradient(n,xp,f(xp),f,typx,1e-7);
                    newslope = dfp'*pc; % Update Slope
                    if(newslope < beta*initslope)
                        llo = lambda; % Lower Estimate of Lambda
                        ldiff = ldiff-lincr; % Change in Lambda
                        flo = fp; % Lower Estimate of Objective Function
                    end
                end
            end
            if(newslope < beta*initslope)
                fp = flo; % Update Objective Function
                xp = xc+llo*pc; % Update Step
            end
        end
    end
    retcode = 1; % Satisfactory Step Found
elseif(lambda < minlambda)
    xp = xc; % Update Step
```

```
            retcode = 2; % No Satisfactory Step Found
        else
            if(lambda == 1)
                % Quadratic Interpolation
                ltemp = -initslope/(2*(fp-fc-initslope));
            else
                % Cubic Interpolation
                A = 1/(lambda-lprev)*[1/lambda^2 -1/lprev^2;...
                    -lprev/lambda^2 lambda/lprev^2]*[fp-fc-lprev*initslope;...
                    fpprev-fc-lprev*initslope];
                disc = A(2)^2-3*A(1)*initslope;
                if(A(1) == 0)
                    ltemp = -initslope/(2*A(2));
                else
                    ltemp = (-A(2)+sqrt(disc))/(3*A(1));
                end
                if(ltemp > 0.5*lambda)
                    ltemp = 0.5*lambda;
                end
            end
            lprev = lambda; % Update Previous Lambda
            fpprev = fp; % Update Previous Function Values
            if(ltemp <= 0.1*lambda)
                lambda = 0.1*lambda; % Update Lambda
            else
                lambda = ltemp; % Update Lamdba
            end
        end
    end
```

9.3.9 Hook Step Driver

```
function [xp,del,lam,delprev,phi,phipr] = hookdriver(sN,xc,fc,f,H,L,g,D,del,lam,i,n,delprev,phi,phipr,maxstep)

% Determine New Step, trust region radius, and Levenberg-Marquardt

% Parameter

% ------------------------------------------------------------------------

% Input:

% sN = Newton Step

% xc = current x

% fc = f(xc)

% f = objective function

% H = Hessian Matrix

% L = lower triangular Cholesky Decomposition of H

% g = gradient

% D = scaling matrix

% del = trust region radius

% lam = Levenberg-Marquardt Parameter

% i = iteration

% n = length of x

% delprev = previous trust region radius

% phi = estimate of phi

% phipr = estimate of phi'

% ------------------------------------------------------------------------

% Output:

% xp = new step

% del = new trust region radius

% lam = new Levenberg-Marquardt Parameter

% delprev = previous trust region radius

% phi = new estimate of phi

% phipr = new estimate of phi'

% ------------------------------------------------------------------------


phiprinit = 0; % Initial Phi' Estimate

xpprev = xc;   % Define xpprev

fpprev = fc;   % Define fpprev
```

```
Dvec = diag(D);% Store scaling matrix diagonal

retcode = 4;   % Initialize Return Code

firsthook = 1; % First Hook Step


Newtlen = norm(D*sN)^2; % Determine Newton Length


% Determine Initial trust region radius

if(i == 1 || del == -1)

    lam = 0;

    if(del == -1)

        alpha = norm(1./Dvec.*g)^2;

        beta = 0;

        for i = 1:n

            temp = sum(L(i:n,i).*g(i:n)./(Dvec(i:n).^2));

            beta = beta + temp*temp;

        end

        del = alpha*alpha^(1/2)/beta;

        if(del>maxstep)

            del = maxstep;

        end

    end

end


% Compute new step, trust region radius, and Levenberg-Marquardt Parameter

while(retcode>=2)

    [s,del,lam,phi,phipr,firsthook,phiprinit,Newttaken] = hookstep(n,g,...

        L,H,sN,D,Newtlen,delprev,del,lam,phi,phipr,firsthook,phiprinit);

    delprev = del;

    [xp,fp,maxtaken,del,retcode,xpprev,fpprev] = trustregup(n,xc,fc,f,g,...

        s,D,Newttaken,maxstep,H,L,del,retcode,xpprev,fpprev,1);

end
```

9.3.10 Hook Step Code

```
function [s,del,lam,phi,phipr,firsthook,phiprinit,Newttaken] =...

    hookstep(n,g,L,H,sN,D,Newtlen,delprev,del,lam,phi,phipr,firsthook,...

    phiprinit)

% Determine New Step, trust region radius, and Levenberg-Marquardt

% Parameter

% ---------------------------------------------------------------------------

% Input:

% n = length of x

% g = gradient

% L = lower triangular Cholesky Decomposition of H

% H = Hessian Matrix

% sN = Newton Step

% D = scaling matrix

% Newtlen = Newton Step Length

% delprev = previous trust region radius

% del = trust region radius

% lam = Levenberg-Marquardt Parameter

% phi = estimate of phi

% phipr = estimate of phi'

% firsthook = whether or not first hook step

% phiprinit = initial phi' estimate

% ---------------------------------------------------------------------------

% Output:

% s = new step

% del = new trust region radius

% lam = new Levenberg-Marquardt Parameter

% phi = new estimate of phi

% phipr = new estimate of phi'

% firsthook = updated whether or not first hook step

% phiprinit = updated initial phi' estimate

% Newttaken = whether or not Newton Step has been taken

% ---------------------------------------------------------------------------
```

```
hi = 1.5; % upper bound on trust region

lo = 0.75; % Lower bound on trust region


% Decide Whether or not to take full Newton Step

if(Newtlen<=hi*del)

    Newttaken = 1;

    s = sN;

    lam = 0;

    del = min(del,Newtlen);

else

    Newttaken = 0;

    if(lam>0)

        lam = lam-((phi+delprev)/del)*(((delprev-del)+phi)/phipr);

    end

    phi = Newtlen-del;

    if(firsthook == 1)

        firsthook = 0;

        tempvec = D'*D*sN;

        tempvec = Lsolve(tempvec,L,0);

        phiprinit = -norm(tempvec)^2/Newtlen;

    end

    lamlo = -phi/phiprinit;

    lamup = norm(inv(D)*g)/del;

    done = 0;

    while(done == 0)

        if((lam<lamlo) || (lam>lamup))

            lam = max((lamlo*lamup)^(1/2),1e-3*lamup);

        end

        for i = 1:n

            H(i,i) = H(i,i)+lam*D(i,i)*D(i,i);

        end

        L = choleDecomp(H,0);

        s = cholsolve(L,g);

        for i = 1:n
```

```
            H(i,i) = H(i,i)-lam*D(i,i)*D(i,i);

        end

        steplen = norm(D*s);

        phi = steplen-del;

        tempvec = D'*D*s;

        tempvec = Lsolve(tempvec,L,0);

        phipr = -norm(tempvec)^2/steplen;

        if(((steplen>=lo*del) && (steplen<=hi*del)) ||...

                ((lamup-lamlo)<=eps^(1/3)))

            done = 1;

        else

            lamlo = max(lamlo, lam-(phi/phipr));

            if(phi<0)

                lamup = lam;

            end

            lam = lam-(steplen/del)*(phi/phipr);

        end

    end

end
```

9.3.11 Double Dogleg Driver

```
function [xp,fp,retcode,maxtaken,del] = dogdriver(n,xc,fc,f,g,L,H,sN,D,...

                                                  maxstep,del)

retcode = 4;

xpprev = xc;

fpprev = fc;

firstdog = 1;

Newtlen = norm(D*sN);

while(retcode >= 2)

    [del,firstdog,Cauchylen,eta,ssD,v,s,Newttaken] = dogstep(n,g,L,sN,D,...

        Newtlen,maxstep,del,firstdog);

    [xp,fp,maxtaken,del,retcode,xpprev,fpprev] = trustregup(n,xc,fc,f,g,...

        s,D,Newttaken,maxstep,H,L,del,retcode,xpprev,fpprev,2);

end
```

9.3.12 Double Dogleg Code

```
function [del,firstdog,Cauchylen,eta,ssD,v,s,Newttaken] = dogstep(n,g,L,...

    sN,D,Newtlen,maxstep,del,firstdog,ssD,v,eta,Cauchylen)

% Determine New Step, trust region radius, and Levenberg-Marquardt

% Parameter

% ------------------------------------------------------------------------

% Input:

% n = length of x

% g = gradient

% L = lower triangular Cholesky Decomposition of H

% sN = Newton Step

% D = scaling matrix

% Newtlen = Newton Step Length

% maxstep = maximum step length

% del = trust region radius

% firstdog = whether or not first doubld dog leg step

% ssD = steepest decent direction step

% v = Difference between scaled Newton and Cauchy Steps

% eta <= 1

% Cauchylen = length of the Cauchy step

% ------------------------------------------------------------------------

% Output:

% del = new trust region radius

% firstdog = updated whether or not first dog step

% Cauchylen = length of the Cauchy step

% eta <= 1

% ssD = steepest decent direction step

% s = new step

% Newttaken = whether or not Newton Step has been taken

% ------------------------------------------------------------------------


Dinv = diag(1./diag(D)); % compute inverse of scaling matrix

S = diag(D); % Scaling vector
```

```
if(Newtlen <= del)

    % If full Newton step is acceptible, s = sN

    Newttaken = 1;

    s = sN;

    del = Newtlen;

else

    Newttaken = 0;

    if(firstdog == 1)

        % If first double dog leg

        firstdog = 0;


        % Compute alpha

        alpha = norm(Dinv*g)^2;


        % Compute beta

        beta = 0;

        for i = 1:n

            temp = sum((L(i:n,i).*g(i:n)./(S(i:n).^2)));

            beta = beta+temp^2;

        end


        % Compute steepest descent direction

        ssD = -(alpha/beta)*Dinv*g;


        % Compute Cauchy Step Length

        Cauchylen = alpha*sqrt(alpha)/beta;


        % Determine eta <= 1

        eta = 0.2+(0.8*alpha^2/(beta*abs(g'*sN)));


        % Determine difference between steepest descent and Newton step in

        % scaled metric

        v = eta*D*sN-ssD;
        % If first iterate, set trust region radius

        if(del == -1)

            del = min(Cauchylen,maxstep);

        end

    end


    % Determine new step

    if(eta*Newtlen <= del)

        % Scaled Newton step is acceptible

        s = (del/Newtlen)*sN;

    elseif(Cauchylen >= del)

        % Cauchy step is acceptible

        s = (del/Cauchylen)*(Dinv)*ssD;

    else

        % Deterime convex combination s = Dinv*(ssD + lambda*v)

        temp = v'*ssD;

        tempv = v'*v;

        lambda = (del^2-Cauchylen^2)/(temp+sqrt(temp^2-tempv*...

            (Cauchylen^2-del^2)));

        s = (Dinv)*(ssD+lambda*v);

    end

end
```

9.3.13 Trust Region Update

```
function [xp,fp,maxtaken,del,retcode,xpprev,fpprev] = trustregup(n,xc,...

    fc,f,g,s,D,Newttaken,maxstep,H,L,del,retcode,xpprev,fpprev,steptype)

% Determine whether or not the new step is acceptable and update trust

% region radius

% -----------------------------------------------------------------------

% Input:

% n = length of x

% xc = current x

% fc = f(xc)

% f = objective function

% g = gradient

% s = new step

% D = scaling matrix

% Newttaken = whether or not Newton step has been taken

% maxstep = max step tolerance

% H = Hessian Matrix

% del = trust region radius

% retcode = return code

% xpprev = previous x value

% fpprev = f(xpprev)

% -----------------------------------------------------------------------

% Output:

% xp = new step

% fp = f(xp)

% maxtaken = decide whether max step size has been taken

% del = new trust region radius

% retcode = updated return code

% xpprev = previous x value

% fpprev = f(xpprev)

% -----------------------------------------------------------------------


maxtaken = 0;        % Max step has not been taken

steptol = eps^(2/3); % Relative step tolerance
```

```
alpha = 1e-4;        % Slope parameter


steplen = norm(D*s); % Determine New Step Length

xp = xc+s;           % update step

fp = f(xp);          % update f

df = fp-fc;          % determine change in f

initslope = g'*s;    % determine initial slope of f


% Decide whether step is sufficient and update trust region parameter

if((retcode == 3) && ((fp >= fpprev) || (df > alpha*initslope)))

    retcode = 0;

    xp = xpprev;

    fp = fpprev;

    del = del/2;

elseif(df >= alpha*initslope)

    rellength = max(abs(s)./max(abs(xp),1./diag(D)));

    if(rellength < steptol)

        retcode = 1;

        xp = xc;

    else

        retcode = 2;

        deltemp = (-initslope*steplen)/(2*(df-initslope));

        if(deltemp < 0.1*del)

            del = 0.1*del;

        elseif(deltemp >0.5*del)

            del = 0.5*del;

        else

            del = deltemp;

        end

    end

else

    dfpred = initslope;

    if(steptype==1)

        for i = 1:n
```

```
            temp = 1/2*H(i,i)*s(i)^2+sum(H(i,i+1:n)'.*s(i).*s(i+1:n));

            dfpred = dfpred+temp;

        end

    else

        for i = 1:n

            temp = sum(L(i:n,i).*s(i:n));

            dfpred = dfpred+temp;

        end

    end

    if((retcode ~= 2) && ((abs(dfpred-df)<=0.1*abs(df)) ||...

            (df <= initslope)) && (Newttaken == 0) && (del <= 0.99*maxstep))

        retcode = 3;

        xpprev = xp;

        fpprev = fp;

        del = min(2*del,maxstep);

    else

        retcode = 0;

        if(steplen > 0.99*maxstep)

            maxtaken = 1;

        end

        if(df >= 0.1*dfpred)

            del = del/2;

        elseif(df <= 0.75*dfpred)

            del = min(2*del,maxstep);

        end

    end

end
```

9.3.14 Perturbed Cholesky Decomposition

```
function L = choleskyDecomp(H,choltype,maxoffl)

% -------------------------------------------------------------------------

% Cholesky Factorization of H+D

% H = Input Matrix

% choltype = 1 for standard Cholesky; 2 for Modified

% L = Lower Triangle Cholesky Decomposition Matrix

% --------------------------------------------------------------------------

% Revised Modified Cholesky Decompostion Algorithm

% choltype = 2

% By Drew Kouri

% Based on Algorithm presented in SIAM Journal of Optimization

% Volume 9, Issue 4, "H Revised Modified Cholesky Factorization Algorithm"

% RB Schnabel and Elizabeth Eskow

% Goal:

% Given H (symmetrix nxn matrix), find H+E = LL' for E>=0

% --------------------------------------------------------------------------


% Check Input

if(nargin<3), maxoffl=0;   end

if(nargin<2), choltype=2; end


if(choltype==2)

    n = length(H); % Determine Size of H

    L = zeros(n);  % Initialize L

    minl = eps^(1/4)*maxoffl; % Determine Threshold for Perturbation


    % Determine Threshold

    if(maxoffl==0)

        maxoffl = sqrt(max(diag(H)));

    end


    minl2 = eps^(1/2)*maxoffl; % Determine Threshold for Perturbation

    maxadd = 0;
    for j = 1:n

        % Compute jth Column of L

        L(j,j) = H(j,j)-sum(L(j,1:j-1).^2);

        minljj = 0;

        for i = j+1:n

            L(i,j) = H(j,i)-L(i,1:j-1)*L(j,1:j-1)';

            minljj = max(abs(L(i,j)),minljj);

        end


        % Decide if H requires perturbation

        minljj = max(minljj/maxoffl,minl);

        if(L(j,j))>minljj^2;

            L(j,j) = sqrt(L(j,j));

        else

            if(minljj<minl2)

                minljj = minl2;

            end

            maxadd = max(maxadd,minljj^2-L(j,j));

            L(j,j) = minljj;

        end

        for i = j+1:n

            L(i,j) = L(i,j)/L(j,j);

        end

    end

else

    % Compute Machine Epsilon

    macheps = MACHINEPS();


    % Tolerance Parameters

    tau = macheps^(1/3);

    tauBar = macheps^(2/3);

    mu = 0.1;

    gamma = max(diag(H));
```

```
delprev = 0;

% Initialize Phase and Iteration Count
phaseone = 1;
j = 1;

% Determine matrix size
n = length(H);

% Initialize Output
L = zeros(n,n);

% Initialize Gerschgorin Bounds
g = zeros(n,1);

% Phase One: H is potentially positive definite
while(j<n && phaseone == 1)
    [maxAjj,i] = max(diag(H(j:n,j:n)));
    minAjj = min(diag(H(j:n,j:n)));
    if(maxAjj<tauBar*gamma || minAjj<-mu*maxAjj)
        phaseone = 0;
    else
        % Pivot on maximum diagonal of remaining submatrix
        if(i~=j)
            irow = H(i,:);
            jrow = H(j,:);
            icolumn = H(:,i);
            jcolumn = H(:,j);
            H(i,:) = jrow;
            H(j,:) = irow;
            H(:,i) = jcolumn;
            H(:,j) = icolumn;
        end
        if(min(diag(H(j+1:n,j+1:n))-H(j+1:n,j).^2./H(j,j))<-mu*gamma)
```

```
            phaseone = 0;
        else
            % Perform jth iteration of factorization
            L(j,j) = sqrt(H(j,j));
            for i = j+1:n
                L(i,j) = H(i,j)/L(j,j);
                for k = j+1:i
                    H(i,k) = H(i,k)-L(i,j)*L(k,j);
                end
            end
            j = j+1;
        end
    end
end

 % Phase Two: H is not positive Definite
 if(phaseone == 0 && j == n)
    del = -H(n,n)+max(tau*(-H(n,n))/(1-tau),tauBar*gamma);
    H(n,n) = H(n,n)+del;
    L(n,n) = sqrt(H(n,n));
 end

if(phaseone == 0 && j < n)
    k = j-1;
    % Calculate lower Gerschgorin bounds in phase one
    for i = k+1:n
        g(i) = H(i,i)-sum(abs(H(i,k+1:i-1)))-sum(abs(H(i+1:n,i)));
    end
    % Modified Cholesky Decompostion
    for j = k+1:n-2
        % Pivot on maximum lower Gerschgorin bound estimate
        [gmax,i] = max(g(j:n));
        if(i~=j)
            irow = H(i,:);
            jrow = H(j,:);

            icolumn = H(:,i);

            jcolumn = H(:,j);

            H(i,:) = jrow;

            H(j,:) = irow;

            H(:,i) = jcolumn;

            H(:,j) = icolumn;

        end

        % Calculate Ejj and add to diagonal

        normj = sum(abs(H(j+1:n,j)));

        del = max([0,-H(j,j)+max(normj,tauBar*gamma),delprev]);

        if(del>0)

            H(j,j) = H(j,j)+del;

            delprev = del;

        end

        % Update Gerschgorin bound estimate

        if(H(j,j)~=normj)

            temp = 1-normj/H(j,j);

            for i = j+1:n

                g(i) = g(i)+abs(H(i,j))*temp;

            end

        end

        % Perform jth iteration of factorization

        L(j,j) = sqrt(H(j,j));

        for i = j+1:n

            L(i,j) = H(i,j)/L(j,j);

            for k = j+1:i

                H(i,k) = H(i,k)-L(i,j)*L(k,j);

            end

        end

        % Final 2x2 submatrix

        lambda = eigs([H(n-1,n-1) H(n,n-1);H(n,n-1) H(n,n)]);

        llo = lambda(2);

        lhi = lambda(1);
```

```
            del = max([0,-llo+max(tau*(lhi-llo)/(1-tau),tauBar*gamma),delprev]);

            if(del>0)

                H(n-1,n-1) = H(n-1,n-1)+del;

                H(n,n) = H(n,n)+del;

                delprev = del;

            end

            L(n-1,n-1) = sqrt(H(n-1,n-1));

            L(n,n-1) = H(n,n-1)/L(n-1,n-1);

            L(n,n) = sqrt(H(n,n)-L(n,n-1)^2);

        end

    end

end
```

9.3.15 Cholesky Solver

```
function x = cholsolve(L,g)

% Solve L*L'*s = -g for s

% L = Cholesky Factorization H = L*L'

% g = Solution Vector


% Solve L*y = g

y = Lsolve(g,L,0);


% Solve L'*s = y

x = Lsolve(y,L,1);


x = -x;
```

9.3.16 Triangular Matrix Solver

```
function y = Lsolve(b,L,state)

function y = Lsolve(b,L,state)

% Solve Ly = b for y

% b = Solution Vector

% L = lower triangular matrix

% state: 0 = lower triangular L

% state: 1 = upper triangular L'


n = length(L);

y = zeros(n,1);


if(state == 0) % Lower Triangular L

    y(1) = b(1)/L(1,1);

    for i = 2:n

        y(i) = (b(i)-sum(L(i,1:i-1)'.*y(1:i-1)))/L(i,i);

    end

elseif(state == 1) % Upper Triangular L'

    y(n) = b(n)/L(n,n);

    for i = n-1:-1:1

        y(i) = (b(i)-sum(L(i+1:n,i).*y(i+1:n)))/L(i,i);

    end

end
```

9.3.17 Finite Difference Jacobian

```
function J = finiteDifferenceJacobian(l,n,xc,Fc,FVEC,Sx,eta)

% Drew Kouri

% Compute the finite difference Jacobian Matrix


% INPUT PARAMETERS:

% l = size of data

% n = number of parameters

% xc = initial parameter vector

% Fc = model evaluated at xc

% FVEC = model

% Sx = 1/(typical value of FVEC(x))

% eta = 1e-DIGITS, where DIGITS is the number of reliable digits of FVEC(x)



 J = zeros(l,n); % Initialize Jacobian Matrix

 sqrteta = sqrt(eta);


 % Calculate the Jacobian Matrix

 for j = 1:n

     % Calculate column j of the Jacobian Matrix

     stepsizej = sqrteta*max(abs(xc(j)),1/Sx(j))*sign(xc(j));

     tempj = xc(j); % Store Initial xc(j)

     xc(j) = xc(j) + stepsizej; % Horizontl Difference

     stepsizej = xc(j) - tempj; % Reduces Finite Precision Errors

     J(:,j) = (FVEC(xc)-Fc)/stepsizej; % Total Difference

     xc(j) = tempj; % Reset xc(j)

 end
```

9.3.18 Finite Difference Gradient

```
function g = finiteDifferenceGradient(n,x,fc,f,Sx,eta)

% Finite Difference Gradient Approximation

% Drew Kouri


sqrteta = sqrt(eta);

g = zeros(n,1);


for j = 1:n

    ss = sqrteta*max(abs(x(j)),1/Sx(j))*sign(x(j));

    temp = x(j);

    x(j) = x(j)+ss;

    ss = x(j)-temp;

    fj = f(x);

    g(j) = (fj-fc)/ss;

    x(j) = temp;

end
```

9.3.19 Finite Difference Hessian

```
function H = finiteDifferenceHessian(n,x,fc,f,Sx,eta)

% Finite Difference Hessian Approximation

% Drew Kouri


cubeta = (eta)^(1/3);

H = zeros(n,n);

ss = zeros(n,1);

fn = zeros(n,1);


for i = 1:n

    ss(i) = cubeta*max(abs(x(i)),1/Sx(i))*sign(x(i));

    tempi = x(i);

    x(i) = x(i)+ss(i);

    ss(i) = x(i)-tempi;

    fn(i) = f(x);

    x(i) = tempi;

end

for i = 1:n

    tempi = x(i);

    x(i) = x(i)+2*ss(i);

    fii = f(x);

    H(i,i) = ((fc-fn(i))+(fii-fn(i)))/(ss(i)*ss(i));

    x(i) = tempi+ss(i);

    for j = (i+1):n

        tempj = x(j);

        x(j) = x(j)+ss(j);

        fij = f(x);

        H(i,j) = ((fc-fn(i))+(fij-fn(j)))/(ss(i)*ss(j));

        x(j) = tempj;

    end

    x(i) = tempi;

end
```

9.3.20 Compute Machine Epsilon

```
function macheps = MACHINEPS()

macheps = 1;

while(macheps+1 ~= 1)

    macheps = macheps/2;

end

macheps = 2*macheps;
```

References

[1] Aboul-ela F, Koh D, Tinoco IJ. *Base-base mismatches. Thermodynamics of double helix formation for dCA3XA3G + dCT3YT3G (X,Y = A,C,G,T).* Nucleic Acids Research. Vol 13 (1985).

[2] Alekseev AK, Navon IM, Steward JL. *Comparison of Advanced Large-Scale Minimization Algorithms for the Solution of Inverse Ill-Posed Problems.*

[3] Allawi HT, SantaLucia J. *Thermodynamics and NMR of Internal GT Mismatches in DNA*. Biochemistry. Vol 36 (1997).

[4] Bertsekas, DP. *On the Goldstein-Levitin-Polyak Gradient Projection Method.* IEEE Transactions on Automatic Control. Vol AC-21. No 2 (April 1976).

[5] Brown KM, Dennis JE. *Derivative Free Analogues of the Levenberg-Marquardt and Gauss Algorithms for Nonlinear Least Squares Approximation.* Numerical Mathematics. Vol 18 (1972).

[6] Byrd RH, Et. Al. *A Limited Memory Algorithm for Bound Constrained Optimization.* Technical Report NAM-08. May 1994.

[7] Byrd RH, Schnabel RB, Shultz GA. *A Trust Region Algorithm for Nonlinearly Constrained Optimization.* SIAM Journal on Numerical Analysis. Vol 24. No 5 (Oct 1987).

[8] Calvetti D, Somersalo E. “Introduction to Bayesian Scientific Computing: Ten Lectures on Subjective Computing.” Springer: New York. 2007.

[9] Carnevale EP, Et. Al. *A Multiplex Ligase Detection Reaction-Fluorescent Microsphere Assay for Simultaneous Detection of Single Nucleotide Polymorphisms Associated with Plasmodium falciparum Drug Resistance*. Journal of Clinical Microbiology. Vol 45. No 3 (2007).

[10] Chylla RA, Marley JL. *Theory and Application of the Maximum Likelihood Principle to NMR Parameter Estimation of Multidimensional NMR Data*. Journal of Biomolecular NMR. Vol 5 (1995).

[11] Conn AR, Gould NIM, Toint PL. *Global Convergence of a class of Trust Region Algorithms for Optimization with Simple Bounds*. SIAM Journal on Numerical Analysis. Vol 25. No 2 (April 1988).

[12] Conn AR, Gould NIM, Toint PL. *Testing a Class of Methods for Solving Minimization Problems with Simple Bounds on the Variables*. Mathematics of Computation. Vol 50. No 182 (April 1988).

[13] Dennis JE, Gay DM, Welsch RE. *An Adaptive Nonlinear Least-Squares Algorithm*. ACM Transactions on Mathematical Software. Vol 7. No 3 (1981).

[14] Denis JE, Mei HHW. *Two New Unconstrained Optimization Algorithms which Use Functions and Gradient Values*. Journal of Optimization Theory and Applications. Vol 28. No 4 (Aug 1979).

[15] Dennis JE, Welsch RE. *Techniques for Nonlinear Least-Squares and Robust Regression*. Communications in Statistics-Simulation and Computation. Vol 7. No 4 (1978).

[16] Dennis JE, Schnabel RB. “Numerical Methods for Unconstrained Optimization and Nonlinear Equations.” Society of Industrial and Applied Mathematics: Philadelphia. 1996.

[17] Dirks RM, Pierce NM. *A Partition Function Algorithm for Nucleic Acid Secondary Structure Including Pseudoknots*. Journal of Computation Chemistry. Vol 24 (2003).

[18] Dirks RM, Pierce NM. *An Algorithm for Computing Nucleic Acid Base-Pairing Probabilities Including Pseudoknots*. Journal of Computation Chemistry. Vol 25 (2004).

[19] Dirks RM, Et. Al. *Thermodynamic Analysis of Interacting Nucleic Acid Strands*. SIAM Review. Vol 49. No 1 (2007).

[20] Excoffier L, Slatkin M. *Maximum-Likelihood Estimation of Molecular Haplotype Frequencies in Diploid Population*. Molecular Biology and Evolution. Vol 12. No 5 (1995).

[21] Facchinei F, Lucidi S. *Quadratically and Superlinearly Convergent Algorithms for the Solution of Inequality Constrained Minimization Problems*. Journal of Optimization Theory and Applications. Vol 85. No 2 (May 1995).

[22] Fan J, Pan J. *Convergence Properties of a Self-Adaptive Levenberg-Marquardt Algorithm Under Local Error Bound Condtion*. Computational Optimization and Applications. Vol 34 (2006).

[23] Fan J, Yuan Y. *On the Quadratic Convergence of the Levenberg-Marquardt Method without Nonsingularity Assumption*. Computing. Vol 74 (2005).

[24] Fang L, Gossard DC. *Multidimensional Curve Fitting to Unorganized Data Points by Nonlinear Minimization*. Computer-Aided Design. Vol 27. No 1 (1995).

[25] Forsgren A, Gill PE, Murray W. *Computing Modified Newton Directions Using a Partial Cholesky Factorization*. SIAM Journal of Scientific Computing. Vol 16. No 1 (1995).

[26] Gertz EM, Gill PE. *A Primal-Dual Trust Region Algorithm for Nonlinear Optimization*. Mathematical Programming. Vol 87. No 2 (April 2000).

[27] Gill PE, Murray W. *Algorithms for the Solution of the Nonlinear Least-Squares Problem*. SIAM Journal on Numerical Analysis. Vol 15. No 5 (Oct 1978).

[28] Gill PE, Leonard MW. *Limited-Memory Reduced-Hessian Methods for Large-Scale Unconstrained Optimization*. SIAM Journal on Optimization. Vol 14 . No 2 (2003).

[29] Gill PE, Et. Al. *Procedures for Optimization Problems with a Mixture of Bounds and General Linear Constraints*. AMC Transactions on Mathematical Software. Vol 10. No 3 (Sept 1984).

[30] Gill PE, Leonard MW. *Reduced-Hessian Quasi-Newton Methods for Unconstrained Optimization*. SIAM Journal of Optimization. Vol 12. No 1 (2001).

[31] Hanson MA, Mond B. *Necessary and Sufficient Conditions in Constrained Optimization.* Mathematical Programming. Vol 37 (1987).

[32] Jagers P, Klebaner F. *Random Variation and Concentration Effects in PCR.* Journal of Theoretical Biology. Vol 224 (2003).

[33] Kelley CT. "Iterative Methods for Linear and Nonlinear Equations." Society of Industrial and Applied Mathematics: Philadelphia. 1995.

[34] Kelley CT. "Iterative Methods for Optimization." Society of Industrial and Applied Mathematics: Philadelphia. 1999.

[35] Kelley CT, Sachs EW. *Local Convergence of the Symmetric Rank-One Iteration.* Computational Optimization and Applications. Vol 9 (1998).

[36] Koop G, Pesaran MH, Potter SM. *Impulse Response Analysis in Nonlinear Multivariate Models.* Journal of Econometrics. Vol 74 (1996).

[37] Korenberg M, Billings SA, Liu YP, McIlroy PJ. *Orthogonal Parameter Estimation Algorithm for Nonlinear Stochastic Systems.* International Journal of Control. Vol 48. No 1 (July 1988).

[38] Kouri DP, Thomas PJ, Zimmerman PA *Transforming Cartesian X,Y Data to Polar Coordinates Differentiates Drug Resistant from Sensitive Alleles of Plasmodium falciparum.* BMC Genetics. Submitted January, 2008.

[39] Lalam N. *Estimation of the Reaction Efficiency in Polymerase Chain Reaction.* Journal of Theoretical Biology. Vol 242 (2006).

[40] Lee JY, Et. Al. *Simulation and Real-Time Monitoring of Polymerase Chain Reaction for its Higher Efficiency.* Biochemical Engineering Journal. Vol 26 (2006).

[41] Leontaritis IJ, Billings SA. *Input-Output Parametric Models for Nonlinear Systems Part II: Stochastic Nonlinear Systems.* International Journal of Control. Vol 41. No 2 (Feb 1985).

[42] Levenberg, K. *A Method for the Solution of Certain Nonlinear Problems in Least Squares.* Quarterly of Applied Mathematics. Vol 2 (1944).

[43] Li ZF, Osborne MR, Prvan T. *Adaptive Algorithm for Constrained Least-Squares Problems.* Journal of Optimization Theory and Applications. Vol 114. No 2 (2002).

[44] Liew, CK. *Inequality Constrained Least-Squares Estimation.* Journal of the American Statistical Association. Vol 71. No 355 (Sept 1976).

[45] Lin C, More JJ. *Newton's Method for Large Bound-Constrained Optimization Problems.* SIAM Journal on Optimization. Vol 9. No 4 (1999).

[46] Lindstrom P, Wedin P. *Gauss-Newton Based Algorithms for Constrained Nonlinear Least Squares Problems.* citeseer.ist.psu.edu/237590.html.

[47] Lloyd AL. *Destabilization of Epidemic Models with the Inclusion of Realistic Distributions of Infectious Periods.* Proceedings of the Royal Society of London: Biology. Vol 268 (2001).

[48] Maheshri N, Schaffer DV. *Computational and Experimental Analysis of DNA Shuffling*. Proceedings of the National Academy of Sciences of the United States of America. Vol 100 (2003).

[49] Marquardt, DW. *An Algorithm for Least-Squares Estimations of Nonlinear Parameters*. Journal of the Society of Industrial and Applied Mathematics. Vol 11. No 2 (1963).

[50] McNamara DT, Et Al. *Diagnosing Infection Levels Of Four Human Malaria Parasite Species By A Polymerase Chain Reaction/Ligase Detection Reaction Fluorescent Microsphere-Based Assay*. American Journal of Tropical Medicine and Hygiene. Vol 74 (2006).

[51] Moorhead M, Et. Al. *Optimal Genotype Determiniation in Highly Multiplexed SNP Data*. European Journal of Human Genetics. Vol 14 (2006).

[52] More JJ, Garbow BS, Hillstrom KE. *Testing Unconstrained Optimization*. ACM Transactions on Mathematical Software. Vol 7. No 1 (March 1981).

[53] More, JJ. *The Levenberg-Marquardt Algorithm: Implementation and Theory*. Lecture Notes in Mathematics, 1977.

[54] More, JJ. *Trust Regions and Projected Gradients*. Lecture Notes on Control and Information Science. Vol 113 (1998).

[55] Murugan R. *A Stochastic Model on DNA Renaturation Kinetics*. Biophysical Chemistry. Vol 104 (2003).

[56] Myung, IJ. *Tutorial on Maximum Likelihood Estimation*. Journal of Mathematical Psychology. Vol 47 (2003).

[57] Nocedal J, Yuan Y. *Combining Trust Region and Line Search Techniques*. Technical Report OTC 98/04. (March 1998).

[58] Nocedal J, Wright SJ. "Numerical Optimization." Springer: New York. 2006.

[59] Panier ER, Tits AL. *A Superlinearly Convergent Feasible Method for the Solution of Inequality Constrained Optimization Problems*. SIAM Journal of Control and Optimization. Vol 25. No 4 (July 1987).

[60] Panier ER, Tits AL. *On Combining Feasibility, Descent and Superlinear Convergence in Inequality Constrained Optimization*. Mathematical Programming. Vol 59 (1993).

[61] Plagnol V, Et. Al. *A Method to Address Differential Bias in Genotyping in Large-Scale Association Studies*. PLOS Genetics. Vol 3. No 5 (May 2007).

[62] Ruhe A, Wedin PA. *Algorithms for Separable Least Squares Problems*. SIAM Review. Vol 22. No 3 (1980).

[63] Rustem, B. *Equality and Inequality Constrained Optimization Algorithms with Convergent Stepsizes*. Journal of Optimization Theory and Applications. Vol 76. No 3. (March 1993).

[64] Schnabel RB, Koontz JE, Weiss BE. *A Modular System of Algorithms for Unconstrained Minimization.* ACM Transactions on Mathematical Software. Vol 11. No 4 (Dec 1985).

[65] Schnabel RB, Eskow E. *A Revised Modified Cholesky Factorization Algorithm.* SIAM Journal on Optimization. Vol 9. No 4 (1999).

[66] Shepard, D. *A Two-Dimensional Interpolation Function for Irregularly Spaced Data.* Proceedings of the 1968 23rd ACM National Conference. (1968).

[67] Shultz GA, Schnabel RB, Byrd RH. *A Family of Trust-Region-Based Algorithms for Unconstrained Minimization with Strong Global Convergence Properties.* SIAM Journal on Numerical Analysis. Vol 22. No 1 (Feb 1985).

[68] Sorensen DC. *Newton's Method with a Model Trust Region Modification.* SIAM Journal on Numerical Analysis. Vol 19. No 2 (April 1982).

[69] Stolovitzky G, Cecchi G. *Efficiency of DNA Replication in the Polymerase Chain Reaction.* Proceedings of the National Academy of Sciences of the United States of America. Vol 93 (1996).

[70] Tarantola A, Valette B. *Generalized Nonlinear Inverse Problems Solved Using the Least Squares Criterion.* Reviews of Geophysics and Space Physics. Vol 20. No 2 (1982).

[71] Tarantola A. "Inverse Problem Theory and Methods for Model Parameter Estimation." Society of Industrial and Applied Mathematics: Philadelphia. 2005.

[72] Teschke G, Ramlau R. *An Iterative Algorithm for Nonlinear Inverse Problems with Joint Sparsity Constraints in Vector-Valued Regimes and an Application to Color Imagae Inpainting*. Inverse Problems. Vol 23 (2007).

[73] Velazquez L, Et. Al. *Selective Search for Global Optimization of Zero or Small Residual Least-Squares Problems: A Numerical Study*. Computation Optimization and Applications. Vol 20 (2001).

[74] Velikanov MV, Kapral R. *Polymerase Chain Reaction: A Markov Process Approach*. Journal of Theoretical Biology. Vol 201 (1999).

[75] Vikalo H, Hassibi B, Hassibi A. *ML Estimation of DNA Initial Copy Number in Polymerase Chain Reaction (PCR) Processes*. Acoustics, Speech and Signal Processing. Vol 1 (April 2007).

[76] Volkman SK, Et. Al. *Genomic Heterogeneity in the Density of Noncoding Single Nucleotide and Microsatellite Polymorphisms in Plasmodium falciparum*. Gene. Vol 387 (2007).

[77] Weiss G, von Haeseler A. *A Coalescent Approach to the Polymerase Chain Reaction*. Nucleic Acids Research. Vol 25. No 15 (1997).

[78] Wenyu S, Yuan Y. "Optimization Theory and Methods: Nonlinear Programming." Springer: New York. 2006.

[79] Whitney SE, Et Al. *Principles of Rapid Polymerase Chain Reactions: Mathematical Modeling and Experimental Verification*. Computation Biology and Chemistry. Vol 28 (2004).

[80] Yuan, Y. *On a Subproblem of Trust Region Algorithms for Constrained Optimization*. Mathematical Programming. Vol 47 (1990).

[81] Zhang JZ, Xue Y, Zhang K. *A Structured Secant Method Based on a New Quasi-Newton Equation for Nonlinear Least Squares Problems*. BIT Numerical Mathematics. Vol 43 (2003).

[82] Zhang JZ, Chen LH. *Nonmonotone Levenberg-Marquardt Algorithms and Their Convergence Analysis*. Journal of Optimization Theory and Applications. Vol 92. No 2 (1997).

[83] Peter A. Zimmerman, Rajeev K. Mehlotra, Laurin J. Kasehagen and James W. Kazura. *Why do we need to know more about mixed Plasmodium species infections in humans?* TRENDS in Parasitology. Vol.20 No.9 September 2004.

[84] Zhu C, Et. Al. *Algorithm 778: L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization*. ACM Transaction on Mathematical Software. Vol 23. No 4 (Dec 1997).

Tables

Parameter Name	Actual	NLLS Fit	Difference (%)
Replication Factor	1.95	1.9431	0.356
Maximum PCR	1,806,000,000,000	1,767,866,083,613.87	2.112
Binding Probability 1	0.94	0.9447	0.505
Binding Probability 2	0.92	0.9201	7.112×10^{-3}
Maximum FMA 1	9000	8883.8280	1.291
Maximum FMA 2	11000	10790.5105	1.904
Midpoint FMA 1	23,000,000,000	20,376,474,855.7563	11.407
Midpoint FMA 2	25,000,000,000	22,163,646,347.0478	11.345

Table 1: This table contains the actual parameters used to generate the surrogate dilution/mixing experiment data and the NLLS fit parameters for the dilution/mixing experiment.

Figures

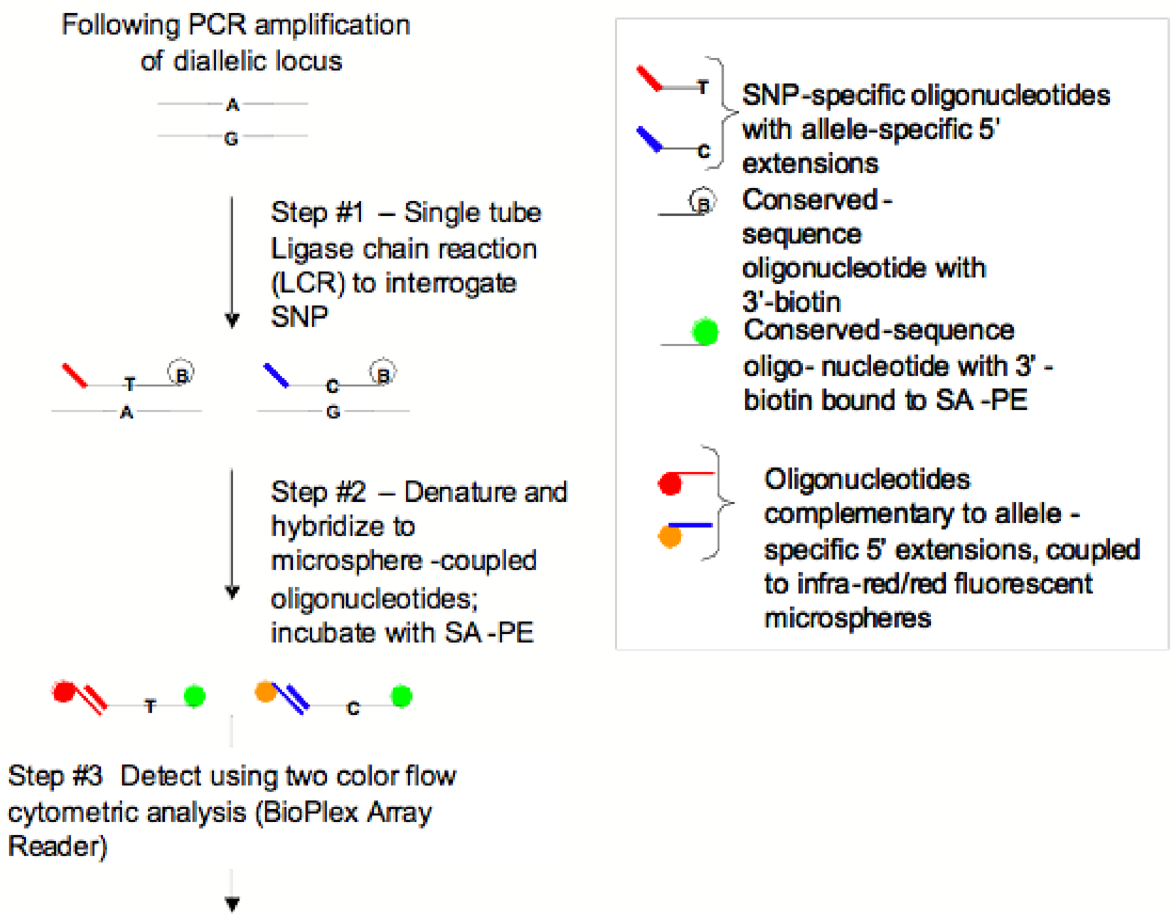
Following PCR amplification of diallelic locus
A
G
Step #1 – Single tube Ligase chain reaction (LCR) to interrogate SNP
T
B
A
C
B
G
Step #2 – Denature and hybridize to microsphere -coupled oligonucleotides; incubate with SA -PE
T
C
Step #3 Detect using two color flow cytometric analysis (BioPlex Array Reader)
T
C
SNP-specific oligonucleotides with allele-specific 5' extensions
B
Conserved - sequence oligonucleotide with 3'-biotin
Conserved-sequence oligo- nucleotide with 3' - biotin bound to SA -PE
Oligonucleotides complementary to allele - specific 5' extensions, coupled to infra-red/red fluorescent microspheres

Figure 1: Incorrect binding of the allele specific primers in the LDR stage results in increased background signal. This schematic shows how, in the ideal case, the LDR stage of the molecular assay works. First the double stranded DNA denatures, then allele specific and common primers binding to the single stranded DNA. Finally, the fluorescent microspheres bind to the allele specific primers.

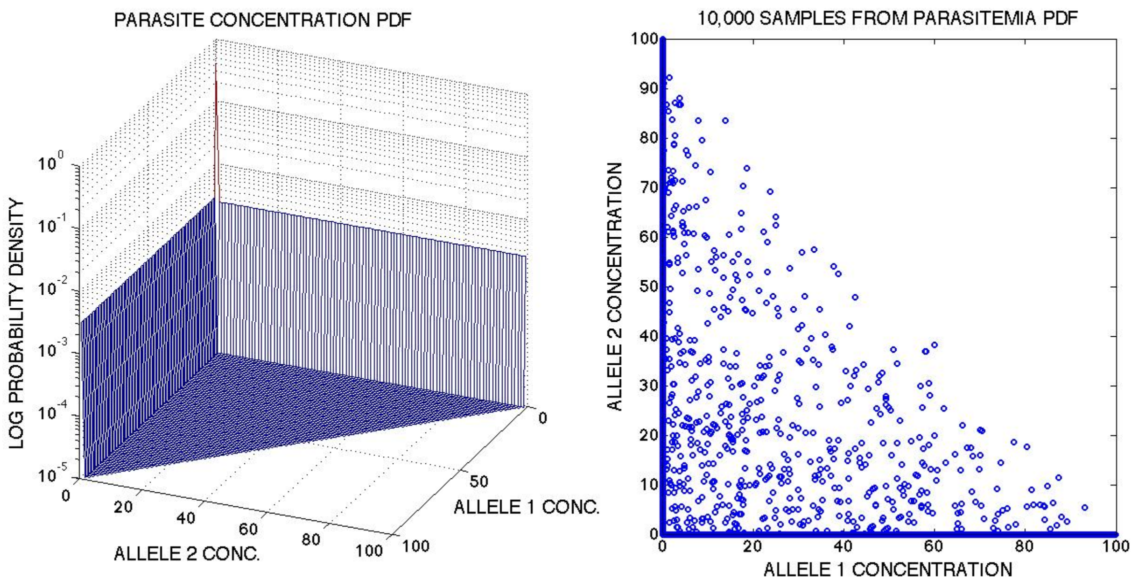

Figure 2: The image on the right is the parasitemia probability density function on a z axis log scale. The image on the left is the scatter plot of 10,000 samples drawn from the parasitemia PDF.

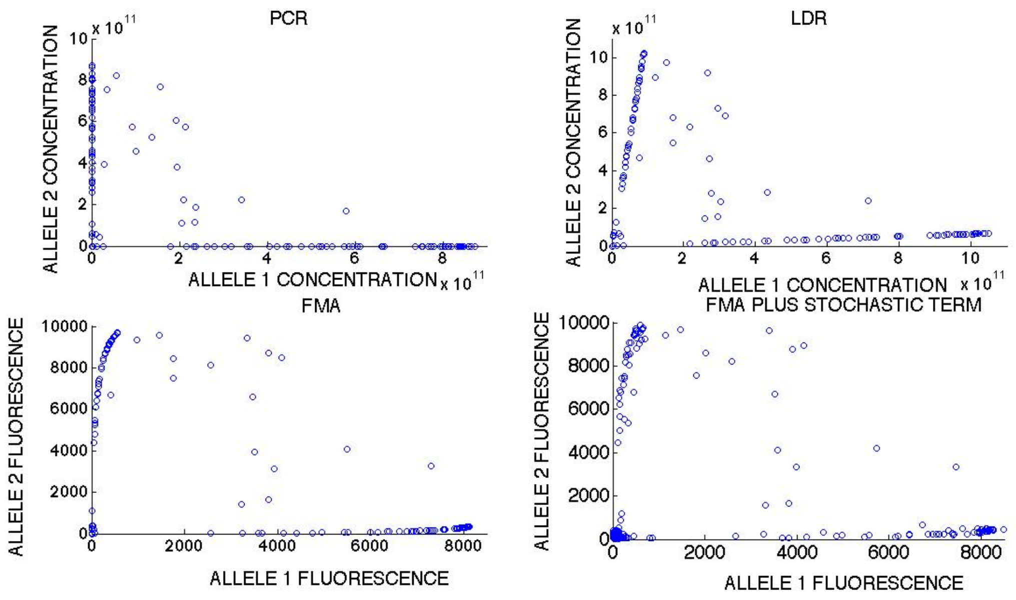

Figure 3: Beginning with 264 samples drawn from the parasitemia PDF, the top right graph is the 264 samples post PCR, the top left graph is the 264 samples post PCR and LDR, the bottom right graph is the 264 samples post PCR, LDR, and FMA without the stochastic term, the final graph is the 264 samples post PCR, LDR, and FMA with the stochastic term.

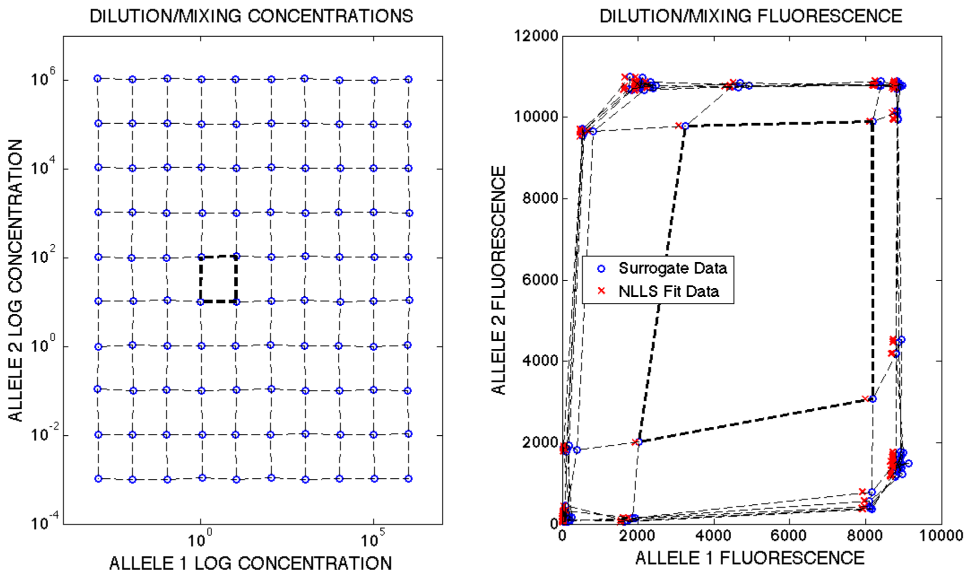

Figure 4: We ran a computer simulated dilution/mixing experiment, then applied the molecular model with a set of reasonable parameters to generate fluorescence data. Using NLLS, we estimated the parameters for the model and plotted the original and the estimated flourescence against each other.

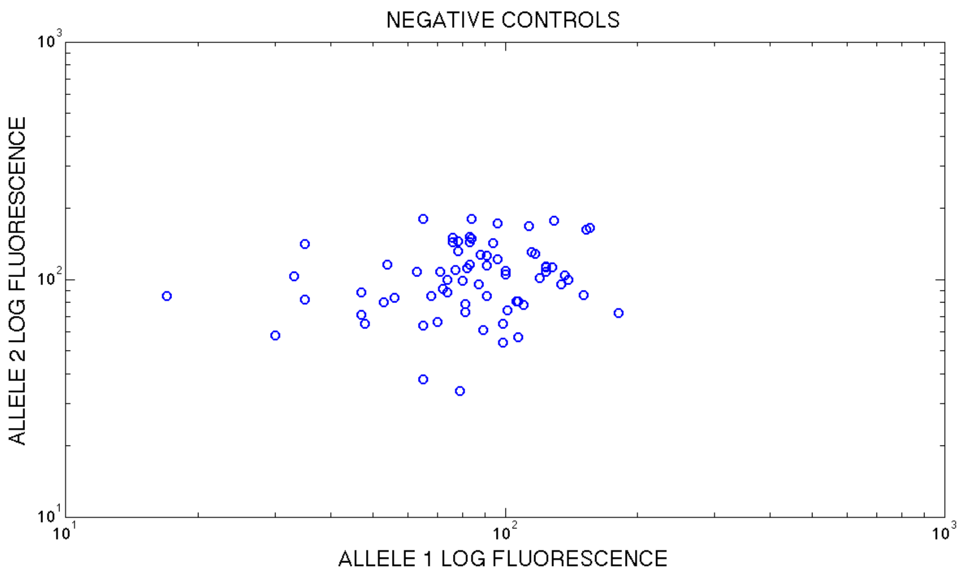

Figure 5: Scatter plot of fluorescence data for locus 59 in the *dhfr* gene of 70 uninfected North Americans. This data is used to compute the mean and variance of the bivariate lognormal distribution.

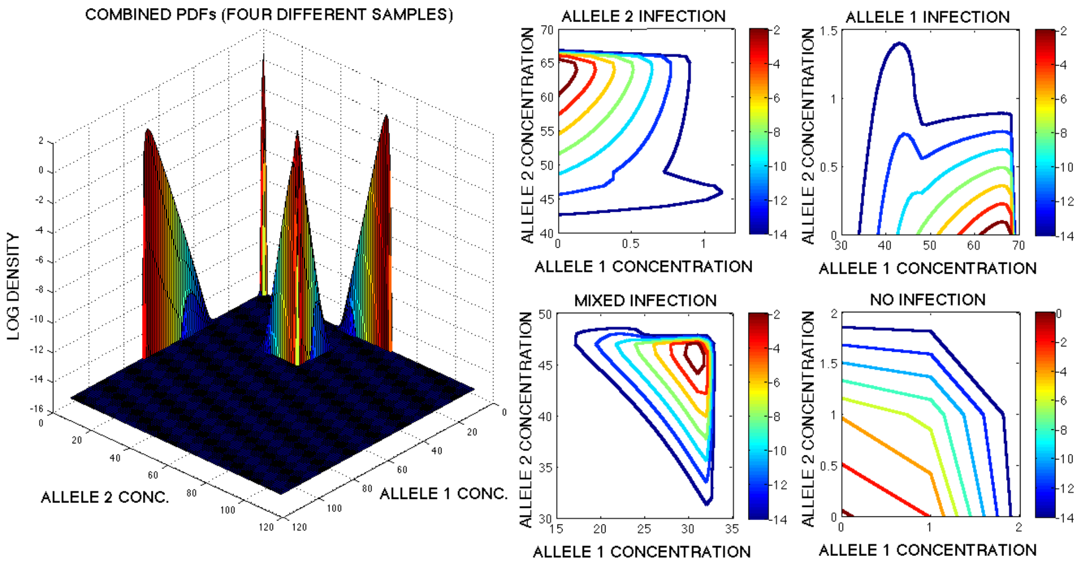

Figure 6: We generated conditional probability distributions using parameters estimated from surrogate controlled experiment data as well as from surrogate field data. These distributions described the possible input allele concentrations for a given fluorescence signal. In the upper left corner the fluorescence was $\vec{y} = [344, 8698]^T$, in the upper right corner the fluorescence was $\vec{y} = [2620, 121]^T$, in the lower left $\vec{y} = [5990, 7952]^T$, and in the lower right $\vec{y} = [87, 48]^T$. The colorbars have units of log density.