The "Fair" Triathlon: Equating Standard Deviations Using Non-Linear Bayesian Models

Steven McKay Curtis
Brigham Young University - Provo

THE "FAIR" TRIATHLON: EQUATING STANDARD DEVIATIONS USING NON-LINEAR BAYESIAN MODELS

by

S. McKay Curtis

A Project submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Department of Statistics

Brigham Young University

April 2004

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a Project submitted by

S. McKay Curtis

This Project has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Gilbert W. Fellingham, Chair
Date	C. Shane Reese
Date	G. Bruce Schaalje

ABSTRACT

THE “FAIR” TRIATHLON: EQUATING STANDARD DEVIATIONS USING NON-LINEAR BAYESIAN MODELS

S. McKay Curtis

Department of Statistics

Master of Science

The Ironman triathlon was created in 1978 by combining events with the longest distances for races then contested in Hawaii in swimming, cycling, and running. The Half Ironman triathlon was formed using half the distances of each of the events in the Ironman. The Olympic distance triathlon was created by combining events with the longest distances for races sanctioned by the major federations for swimming, cycling, and running. The relative importance of each event in overall race outcome was not given consideration when determining the distances of each of the races in modern triathlons. Thus, the swimming portion in the modern triathlon is underweighted. We present a nonlinear Bayesian model for triathlon finishing times that models time and standard deviation of time as a function of distance. We use this model to create “fair” triathlons by equating the standard deviations of the times taken to complete the swimming, cycling, and running events. Thus, in these “fair” triathlons, a one standard deviation improvement in any event has an equivalent impact on overall race time.

Acknowledgements

I want to thank Dr. Gilbert Fellingham for the many hours he spent in consultation with me to ensure that this project (and ultimately the attainment of my Master's degree) was successful. The success of this project also depended largely on Dr. Shane Reese, whose knowledge and experience in the field of Bayesian statistics helped me overcome many set backs. Randall Barber and Mark Vickers helped translate my R code into C code and thereby reduced the amount of time I would have spent writing and running code. This project would not have been possible without data, and Bill McGuire spent numerous hours searching the internet, cleaning data, and writing SAS code to get the data into a usable form. Most of all, I want to thank my wife, Stephanie, who not only has given me moral support, but has also helped with the "grunt work" of finding data on the internet, cleaning data, and reading data into SAS.

Contents

Chapter

1 Introduction 1

2 Review of Literature 5

2.1 Bayesian Methods 5

2.2 Nonlinear Models 12

2.3 Triathlon Research 14

3 The Model 16

3.1 The Fair Triathlon Defined 16

3.2 Data Description 19

3.3 Model 19

3.3.1 Sampling Density Model 19

3.3.2 Prior Distribution Model 26

3.3.3 Computation 33

3.3.4 Posterior Summaries 34

3.3.5 Model Checking 34

4 Results 35

4.1 Introduction 35

4.2 The Fair Triathlon Defined 37

4.3 Data Description 40

4.4 The Model 40

4.4.1 Sampling Density Model 42

4.4.2 Prior Distribution Model 49

4.5 Results 54

4.6 The Fair Triathlon 57

5 Further Research and Contributions to Statistical Practice 63

5.1 A Hierarchical Model 63

5.2 Covariance Structure 64

5.3 Sparsity of Data 65

5.4 Contributions to Statistical Practice 65

Appendix

A Data Sources 67

B Further Graphical Model Checks 69

C Code: Posterior Simulation 78

C.1 Swim Code 78

C.2 Bike Code 91

C.3 Run Code 103

D Code: Model Summaries 116

D.1 Swim Code 116

D.2 Bike Code 119

D.3 Run Code 123

D.4 Posterior Summaries 127

D.5 The Fair Triathlon Code 131

Bibliography 139

Tables

Table

3.1 Types of triathlons in the dataset and their respective distances in miles for the swim, bike, and run. “Number” is the number of triathlons of that type in the data set. “Triathletes” is the number of triathletes in the data set who recorded a finishing time in that type of triathlon. 20

3.2 Table of parameter values for prior distributions for all three portions of a triathlon. 28

3.3 Table of means and standard deviations for prior distributions of each parameter in the model. Distances δ_1 and δ_2 (in miles) are also listed for the parameters τ_1 and τ_2. 29

3.4 Possible values for β and their corresponding finishing times for a swim of 1.0 mile and 1.5 miles. The means of the prior distributions for τ_1 and τ_2 were used in equation(3.8) along with the values of β in the table to calculate the predicted finishing times. 32

4.1 Types of triathlons in the dataset and their respective distances in miles for the swim, bike, and run. “Number” is the number of triathlons of that type in the data set. “Triathletes” is the number of triathletes in the data set who recorded a finishing time in that type of triathlon. 41

4.2 Table of parameter values for prior distributions for all three portions of a triathlon. 51

4.3 Table of means and standard deviations for prior distributions of each parameter in the model. Distances δ_1 and δ_2 (in miles) are also listed for the parameters τ_1 and τ_2. 51

4.4 Possible values for β and their corresponding finishing times for a swim of 1.0 mile and 1.5 miles. The means of the prior distributions for τ_1 and τ_2 were used in equation(4.4.1.3) along with the values of β in the table to calculate the predicted finishing times. 54

4.5 Posterior means, posterior standard deviations, and credible intervals for each parameter in the model for each triathlon component. 56

4.6 Table of fair triathlon distances (in miles). Each triathlon is based on a run distance of 10, 15, 20, or 25 kilometers. Also listed are the ratios of the bike distances to the swim distances and the run distances to the swim distances. 60

A.1 Internet sources for the triathlons in the dataset. 68

Figures

Figure

3.1 Histograms of the finishing times for each of the three events in the Olympic, Half Ironman, and Ironman triathlons in the data set. Each histogram is fit with a density smoother to highlight the shape of the data. 22

3.2 Plots of the logarithm of event times versus distance. Scales on the x and y axes are different for each plot. 24

4.1 Histograms of the finishing times for each of the three events in the Olympic, Half Ironman, and Ironman triathlons in the data set. Each histogram is fit with a density smoother to highlight the shape of the data. 43

4.2 The three parameter lognormal distribution with three different values for the peakedness parameter α and $E[X] = 40$. 45

4.3 Plots of the logarithm of event times versus distance. Scales on the x and y axes are different for each plot. 47

4.4 Plots of the log of the sum of the posterior mean of αplus the finishing time for each triathlon component. The line on the plot is equation (4.8) with the posterior means of τ_1, τ_2, and β. 58

4.5 Normal probability plots of the residuals for each event. The residuals were calculated as the difference between the log of the sum of finishing time and the posterior mean for alpha and the predicted value for μ at a given distance. 59

B.1 Histograms of the residuals for the swim, cycle, and run. The thick solid line on each plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals. The thin line on each plot is a density smooth of the residuals. 70

B.2 Histograms of the residuals at each unique swim distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals at that distance. The thin line is a density smooth of the data. 72

B.3 Normal probability plots of the residuals at each unique swim distance in the data set. 73

B.4 Histograms of the residuals at each unique bike distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals at that distance. The thin line is a density smooth of the data. 74

B.5 Normal probability plots of the residuals at each unique bike distance in the data set. 75

B.6 Histograms of the residuals at each unique run distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance at that distance. The thin line is a density smooth of the data. 76

B.7 Normal probability plots of the residuals at each unique run distance in the data set. 77

Chapter 1

Introduction

According to popular myth[1], the first Ironman triathlon was conceived in a bar in Hawaii. Chatting over a few beers, Naval Commander John Collins debated with several others the following question: "Who are the toughest athletes—swimmers, cyclists, or runners." Collins suggested that to answer the question athletes from each discipline should compete in an event that consists of all three sports run back-to-back. Collins proposed a race comprised of the longest events in swimming, cycling, and running then contested in Hawaii—the 2.4 mile Waikiki Rough Water Swim, the 112 mile Around Oahu Bike Ride, and the 26.2 mile Honolulu Marathon. So, in 1978, Collins and a little over a dozen others participated in the first Ironman triathlon.

The popularity of the Ironman triathlon increased dramatically after the second Hawaii Ironman when *Sports Illustrated* ran a story on the 1979 winner Tom Warren and ABC began coverage of the Hawaii triathlon in 1980. Soon, other shorter-distance triathlons emerged. The Half-Ironman triathlon was formed by

[1] For Collins' comments and clarifications on the genesis of the Ironman triathlon see `http://vnews.ironmanlive.com/vnews//1043361628/?keywords=john:collins`

taking half the distances of each event in the Ironman triathlon (1.2 mile swim, 56 mile bike, and a 13.1 mile run). The International Triathlon Union (ITU) formed the Olympic triathlon by combining the longest event distances used in races sanctioned by each of the major federations for swimming (the International Federation for Swimming), cycling (International Cycling Union), and track (International Association of Athletics Federations).[2]The Olympic Triathlon consists of a 1.5 km (0.93 miles) swim, a 40 km (24.86 miles) bike, and a 10 km (6.21 miles) run.

Today, the three major triathlon types are the Olympic, Half-Ironman, and Ironman triathlons. However, triathletes can compete in a plethora of other triathlons of different distances. Sprint triathlons are composed of events of shorter distances than the Olympic triathlon. Triathletes also compete in Double and Triple Ironman triathlons, which—as their names imply—are twice and three times the distance of the Ironman triathlon.

As the history of the modern triathlon indicates, little consideration was given to the relative importance of each event when forming the various triathlons. In fact, empirical evidence indicates that the swimming portion of the modern triathlon is extremely underweighted. Dengel et al. (1989), in a study of eleven male triathletes, report that, "swimming time was not significantly related to overall triathlon time."

In a fair triathlon, a one standard deviation improvement in time in any of

[2] See `http://www.triathlon.org/tv/FAQs.htm`.

the three triathlon components results in the same improvement in overall time. As described in Section 3.1, the standard deviations in each component of a fair trithlon are equal. Because finishing times in each triathlon component increase with distance, a fair tritathlon can be constructed by finding distances for each event such that event standard deviations are the same. Thus, a statistical model for triathlon finishing times can be used to find the appropriate distances that equate the standard deviations in each triathlon component.

This project presents a Bayesian model for the distribution of triathlon times as a function of distance. The core feature of this model is unique in the statistics literature and consists of a nonlinear functional for the parameter μ in a lognormal distribution and use of a third parameter α to account for tail thickness in the data. This model makes it possible to equate standard deviations of the three race elements as a function of distance—thus, creating a "fair" triathlon.

This report details the model for triahtlon finishing times and begins in Chapter 2 by discussing the relevant literature of Bayesian estimation, nonlinear models, and triathlon research. Chapter 3 defines the fair triathlon and describes the model for triathlon finishing times as a function of distance. Chapter 4 is a paper, which was submitted to a student paper competition sponsored by the Western North American Region of the International Biometric Society (WNAR), that contains the results of the triathlon model, including a method for constructing the fair triathlon. Chapter 5 presents some drawbacks to the model, discusses possible areas for further research, and outlines the contributions to statistical practice.

Chapter 2

Review of Literature

This chapter is a review of the literature that describes the statistical methodology appropriate for modeling triathlon race times as a function of distance. This literature can be separated into three broad categories. The first category is Bayesian inference; the second category is nonlinear models; and the third category is general triathlon research.

2.1 Bayesian Methods

Bayesian methodology relies solely on Bayes' Theorem for estimation of model parameters. Although named after Thomas Bayes for work published posthumously in 1764 (see Bayes 1764), Bayes' Theorem first appeared in Hartley (1749) where Hartley claimed an "ingenious Friend has communicated to me of the solution of the inverse problem..." (see Stigler 1983). A modern version of Hartley's "Friend's" theorem is stated succinctly below:

Theorem 1 (Bayes' Theorem) For two events A and B

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

Bayes' Theorem can be used for parameter estimation simply by substituting the vector of model parameters θ for B and the vector of observed data y into the theorem to obtain

$$\pi(\theta|y) = \frac{f(y|\theta)\pi(\theta)}{f(y)}$$

or, alternatively,

$$\pi(\theta|y) = \frac{f(y|\theta)\pi(\theta)}{\int f(y|\theta)\pi(\theta)d\theta}, \tag{2.1}$$

where $f(y|\theta)$ is the sampling density of the data given the parameter θ, $\pi(\theta)$ is the prior distribution for the parameter θ, $f(y)$ is the marginal distribution of y, and $\pi(\theta|y)$ is the posterior distribution for the parameter θ.

Bayes (1764) is the first known example of parameter estimation using Bayes' Theorem. Bayes presented the problem of estimating a binomial parameter using Bayes' Theorem in a scenario where a ball is rolled across a table in such a way that the probability of it resting at any one horizontal position is uniform. If one edge of the table has a horizontal coordinate of zero and the other edge of the table has a horizontal coordinate of one, then the horizontal coordinate of the resting place of the ball θ is a number between zero and one. In Bayes' scenario another ball is rolled across the table $p+q$ times and the number of times the ball comes to rest to the left of θ is recorded as p and the number of times the ball comes to rest to the right of θ is recorded as q. Thus, if X is a binomial random variable of the number of times the ball comes to rest to the right of θ in $p+q$ trials, then the likelihood for X is given by

$$\binom{n}{p} \theta^p (1-\theta)^{n-p}.$$

And, by Bayes' Theorem,

$$P[a < \theta < b | X = p] = \frac{\int_a^b \binom{n}{p} \theta^p (1-\theta)^{n-p} d\theta}{\int_0^1 \binom{n}{p} \theta^p (1-\theta)^{n-p} d\theta}.$$

which assumes a uniform prior on θ (i.e. $\pi(\theta) = 1$). Bayes justified the use of this prior distribution in a scholium to his 1764 paper. (For an explanation and clarification of modern commentators' misperceptions of Bayes' scholium, see Stigler 1982. For a summary of Bayes' paper see Gelman et al. 2004, pp. 34–36 and Stigler 1982.)

Although most likely unacquainted with the work of Thomas Bayes (Stigler 1978), Laplace (1774, reprinted in 1986) also used Bayes Theorem to estimate a binomial parameter. Laplace used a hypothetical urn containing an infinite number of black and white tickets in an unknown ratio as his motivating example. Laplace also used a uniform prior distribution for the binomial parameter θ, although he gave no justification for the use of such a prior.

Laplace (1774, reprinted in 1986) also approached the problem of what "mean" of three observations should be used in estimating the true mean of a population. He proposed two "means"—the median of the posterior distribution and the quantity that minimizes the absolute error loss—and then proved that these two quantities were the same. (For a summary of Laplace 1774, reprinted in 1986, see Stigler 1986.)

Despite its beginnings in the 18^{th} century, Bayesian methodology did not gain widespread use because of computational difficulties. Even for very simple problems, such as estimating the mean of a normal population with unknown variance, the posterior distribution is intractable because the integral in the denominator of equation (2.1)—called the normalizing constant—is difficult or impossible to calculate. However, since the 1950's several methods have been devised for approximating distributions through simulation by using Markov chain Monte Carlo (MCMC) methods. Metropolis et al. (1953) devised a procedure to simulate from a target distribution $\pi(\theta|y)$ using Markov chains. The first step in the algorithm involves choosing a starting value of the parameter vector $\theta^{(0)}$ for the Markov chain. The next step is to simulate a "candidate" $\theta^{(cand)}$ from a "proposal" distribution, where the proposal distribution p must be symmetric—that is, $p(\theta_a|\theta_b) = p(\theta_b|\theta_a)$ for all θ_a and θ_b. Then, for the t^{th} iteration, the ratio $r = g(\theta^{(cand)})/g(\theta^{(t-1)})$ is calculated, where $g(\cdot)$ is the unnormalized posterior distribution of θ—that is,

$$g(\theta) = f(y|\theta)\pi(\theta)$$

is the numerator of equation (2.1) The t^{th} value of the Markov chain is then determined by the following:

$$\theta^{(t)} = \begin{cases} \theta^{(cand)} & \text{with probability } \min(r,1) \\ \theta^{(t-1)} & \text{otherwise.} \end{cases}$$

Metropolis' algorithm was generalized by Hastings (1970) and is commonly referred to as the Metropolis-Hastings algorithm. Hastings' version of the algorithm doesn't require the proposal density to be symmetric. Because the proposal density is no longer required to be symmetric, the ratio r becomes

$$r = \frac{g(\theta^{(cand)})/p(\theta^{(cand)}|\theta^{(t-1)})}{g(\theta^{(t-1)})/p(\theta^{(t-1)}|\theta^{(cand)})}$$

(See Chib and Greenberg 1995 for a more detailed introduction to the Metropolis-Hastings algorithm.)

Geman and Geman (1984) use a special case of the Metropolis-Hastings algorithm (Gelman et al. 2004, p. 293) for generating from a posterior distribution. The algorithm, which they named the Gibbs sampler, entails generating values of θ_i from the conditional distributions $\theta_i|\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_k$. As explained in Casella and George (1992), "knowledge of the conditional distributions is sufficient to determine a joint [posterior] distribution." Thus, as the chain of simulations converges, the conditional distribution of $\theta_i|\theta_1, \ldots, \theta_{i-1}, \theta_{i+1}, \ldots, \theta_k$ converges to the marginal distribution of θ_i; and the marginal distributions of θ_i for each i collectively form draws from the joint posterior density.

Although the Gibbs sampler gained popularity in image processing research, Gibbs sampling was not commonly used for more general statistical problems until Gelfand and Smith (1990). Gelfand and Smith present many applications of the Gibbs sampler in common statistical problems. These examples include the simplest case of a variance components model, where $y_{ij} = \theta_i + \epsilon_{ij}$ and θ and ϵ are random components, and the normal means model, where $Y_{ij} \sim N(\theta_i, \sigma_i^2)$, $\theta_i \sim N(\mu, \tau)$, and $\sigma^2 \sim IG(a, b)$ (i.e. each observation can potentially come from a different normal distribution).

The increasing use of MCMC methods for simulating from a posterior distribution has given rise to many practical issues such as assessing convergence, choosing the number of iterations, and choosing the burn-in length. Kass et al. (1998) is a discussion with several experts in Bayesian analysis on many of these practical issues. Their advice includes using the convergence diagnostic $\hat{R}$ (Gelman and Rubin 1992). The diagnostic $\hat{R}$ uses multiple MCMC sequences for the same parameter θ_i to compute within-sequence variation and between-sequence variation. $\hat{R}$ is a function of the two variances and converges to one as the sequence approaches the target density. Gelman et al. (2004, p. 297) recommend continuing simulation until all $\hat{R}$'s are close to one for each parameter.

Raferty and Lewis (1996) address the problems of the number of iterations and the length of the burn-in for a MCMC simulation. They motivate their solution with an appeal to the estimation of quantiles of the posterior distribution. In other words, they find the number of iterations and burn-in length that will provide an estimate of the q^{th} quantile of the posterior distribution to within $\pm r$ with probability s. Raferty and Lewis implement their solutions in their `gibbsit` software. For a given MCMC sequence, the `gibbsit` software computes the recommended burn-in length, the number of iterations after burn-in, and the "thin" (where a thin value of k requires that only every k^{th} iteration from the final chain be kept). The `gibbsit` software also provides the quantity I which "measures the increase in the [recommended] number of iterations due to dependence in the sequence" (Raferty and Lewis 1996, p. 119). Values near one are desirable, and values greater than five indicate “problems that might be alleviated by changing the implementation” of the MCMC sequence (Raferty and Lewis 1996, p. 119).

The field of statistics is generally concerned with creating mathematical models of data for use in inference and prediction. Bayesian statistics is no exception. Lindley and Smith (1972) outline the Bayesian linear model. They present two versions of the model; the first models the mean of the distribution of the data as a linear function of a vector of parameters,

$$\begin{aligned} \mathbf{y}|\,\theta_1 &\sim N(\mathbf{A}_1\theta_1, \mathbf{C}_1) \\ \theta_1 &\sim N(\mathbf{A}_2\theta_2, \mathbf{C}_2), \end{aligned}$$

and the second adds an additional level by modeling the mean of the distribution of the parameter θ_1 as a linear function of a vector of hyperparameters θ_2,

$$\begin{aligned} \theta_1|\,\theta_2 &\sim N(\mathbf{A}_2\theta_2, \mathbf{C}_2) \\ \theta_2 &\sim N(\mathbf{A}_3\theta_3, \mathbf{C}_3). \end{aligned}$$

Lindley and Smith use the two-factor effects model to demonstrate a practical application of the Bayesian linear model.

The hierarchical model is a natural fit for Bayesian estimation because the number of parameters in a hierarchical model usually exceeds the number of data points—making maximum likelihood estimation impossible. Reese et al. (2001) use a hierarchical model to estimate fetal growth and gestation for bowhead whales. Their model includes three parameters for every observation—a parameter for length of fetus at birth, date of birth, and date of conception. Using Bayesian estimation, Reese et al. estimate conception dates, parturition dates and fetal length as a function of fetal age. Graves et al. (2003) hierarchically model the finishing positions of race-car drivers. Using a hierarchical structure and Bayesian estimation, Graves et al. are able to estimate driver abilities and track-driver interactions.

2.2 Nonlinear Models

In the nonlinear and growth models literature, Chambers (1973) reviews several different numerical techniques—including Newton-Raphson—for estimating nonlinear regression parameters. Chambers describes a general approach to estimation whereby a value for θ is found that minimizes some objective function $F(\theta)$. Chambers also gives some practical advice for incorporating considerations of the numerical estimation procedure into the statistical model. Gallant (1975) gives a gentle introduction to nonlinear regression. Gallant discusses least-squares estimation of parameters, statistical properties of least-squares estimators, hypothesis testing, and confidence intervals. Sandland and McGilchrist (1979) present a general class of growth models based on the stochastic differential equation

$$(1/W)dW/dt = f[W(t), \alpha, \epsilon(t)],$$

where t is time, W is a measure of growth, α is a vector of parameters, and $\epsilon(t)$ is random error.

Schnute (1981) presents a general growth model under which several other popular growth models are special cases (e.g. Gompertz, von Bertalanffy, Richards, Logistic). Schnute bases his model on the following differential equations:

$$\begin{aligned} Z &= \frac{1}{Y}\frac{dY}{dt} && (2.2) \\ \frac{1}{Z}\frac{dZ}{dt} &= -(a - bZ) && (2.3) \end{aligned}$$

where Y is some biological measurement (e.g. length, weight) as a function of time. The quantity (2.2) is the relative growth rate, and the quantity (2.3) is the relative growth rate of the relative growth rate. Schnute solves the system of differential equations for different cases when a and b are equal to zero or not equal to zero. To solve the differential equations, Schnute makes some model constraints. For example, if at times $\tau_1$and τ_2, the biological measurement is constrained to equal y_1 and y_2 respectively, then the growth function based on (2.2) and (2.3) is

$$Y(t) = \left[y_1^b + (y_2^b - y_1^b)\frac{1 - e^{-a(t-\tau_1)}}{1 - e^{-a(\tau_2-\tau_1)}} \right]^{1/b}, \qquad (2.4)$$

when $a \neq 0$ and $b \neq 0$.

White (1981) discusses the consequences of least-squares estimation of non-linear regression parameters (namely inconsistent estimators) when the nonlinear model is misspecified. White demonstrates that, under certain conditions, the least squares estimator "converges strongly to the parameters of a (weighted) least squares approximation to the true model." White also shows that, under some additional conditions, the least squares estimators are asymptotically normally distributed. In addition, White presents a modified estimator of the covariance matrix that is robust to model misspecification. Cook and Tsai (1985) show that use of "ordinary" residuals in nonlinear models can be misleading. Cook and Tsai propose the use of “projected residuals” and illustrate their superiority to ordinary residuals.

In the literature on nonlinear models from the Bayesian perspective, Eaves (1983) derives an “objective” prior distribution for θ and σ in the model $y = g(\theta)+\sigma\epsilon$. Eaves demonstrates the use of his prior with an enzyme-kinetics example. Hills (1987) discusses two nonlinear models—the Michaelis-Menton model and the logistic model—with reference priors and unidentifiable parameters. Hills presents some model constraints that correct for the unidentifiablility in the parameters.

2.3 Triathlon Research

In the triathlon literature, Laursen and Rhodes (2001) discuss how predictors such as VO_{2max} and anaerobic threshold (AT) are correlated with performance in endurance events. However, Laursen and Rhodes also point out that VO_{2max} and AT are not good predictors of “ultra-endurance triathlons” (which they define as triathlons with finishing times greater than four hours). Laursen and Rhodes review several different factors—substrate utilization, fluid and electrolyte homeostasis, and cardiovascular drift—that could account for the poor prediction by VO_{2max} and AT. Laursen and Rhodes urge researchers to examine other threshold values (perhaps related to AT) that can be used for prediction of ultra-endurance triathlon performance.

Millet et al. (2002) compare the differences in swimming styles between triathletes and swimmers. Millet et al. investigated the stroke rate, and the stroke length, and the “index of coordination” (IdC), which they define as the difference in time between the end of the propulsion phase of one arm to the beginning of the propulsion phase of the other arm expressed as a percentage of the mean time of the entire stroke. They measured each of the responses on triathletes and swimmers at six different velocities (where the velocities were measured as a percentage of the athlete’s maximal velocity). Millet et al. do not find differences in IdC or in stroke rate between triathletes and swimmers. However, they report significant differences in the stroke length—triathletes had significantly shorter stroke lengths than swimmers.

This chapter has reviewed the statistical literature appropriate for modeling triathlon data. The next chapter builds on this foundation by detailing a model for triathlon finishing times.

Chapter 3

The Model

This chapter describes the model for triathlon finishing times. The chapter begins with a section that motivates the model by defining the fair triathlon. The next section describes the data used to fit the model. The final section presents the model for the fair triathlon, which includes a discussion of the sampling density, prior distributions, computation, and model checking.

3.1 The Fair Triathlon Defined

The fair triathlon is defined as a triathlon in which a one standard deviation improvement in time in any of the individual triathlon components results in the same improvement in overall triathlon time. This "fair" result can happen if the standard deviations in each of the triathlon components are the same—that is, if the standard deviations of finishing times in swimming, cycling, and running are equal. For example, the overall finishing time t_o for an arbitrary triathlete in an arbitrary triathlon can be written

$$t_o = t_s + t_b + t_r$$

where t_o is the triathlete's finishing time for the entire triathlon and t_s, t_b, and t_r are the triathlete's finishing times for swimming, cycling, and running respectively. Adding and subtracting the mean overall finishing time μ_o for all triathletes in the triathlon gives

$$t_o = \mu_o + t_s + t_b + t_r - \mu_s - \mu_b - \mu_r \tag{3.1}$$

where $\mu_o = \mu_s + \mu_b + \mu_r$ and μ_s, μ_b, and μ_r are the mean finishing times in swimming, cycling, and running for all triathletes the triathlon. In the fair triathlon, the standard deviations in each of the three events are equal. We represent this standard deviation by σ. Then rearranging the terms and multiplying by one $(\frac{\sigma}{\sigma})$ gives

$$\begin{aligned} t_o &= \mu_o + \frac{\sigma}{\sigma}\left(t_s - \mu_s + t_b - \mu_b + t_r - \mu_r\right) && (3.2)\\ t_o &= \mu_o + \sigma\left(\frac{t_s - \mu_s}{\sigma} + \frac{t_b - \mu_b}{\sigma} + \frac{t_r - \mu_r}{\sigma}\right) \\ t_o &= \mu_o + \sigma\left(Z_s + Z_b + Z_r\right). && (3.3) \end{aligned}$$

Thus, if a triathlete finishes one standard deviation faster than the mean time in any of the triathlon components, equation 3.3 becomes one of

$$\begin{aligned} t_o &= \mu_o + \sigma\left(-1 + 0 + 0\right) \\ t_o &= \mu_o + \sigma\left(0 - 1 + 0\right) \\ t_o &= \mu_o + \sigma\left(0 + 0 - 1\right) \end{aligned}$$

and the improvement in the overall time is the same $(-\sigma)$ in each case.

If the standard deviations in each event are not the same, it is not possible to distribute a common standard deviation in equation (3.2) to obtain equation (3.3). However, rearranging and multiplying the terms on the right of equation (3.1) by one gives

$$\begin{aligned} t_o &= \mu_o + \frac{\sigma_s}{\sigma_s}(t_s - \mu_s) + \frac{\sigma_b}{\sigma_b}(t_b - \mu_b) + \frac{\sigma_r}{\sigma_r}(t_r - \mu_r) \\ t_o &= \mu_o + \sigma_s\left(\frac{t_s - \mu_s}{\sigma_s}\right) + \sigma_b\left(\frac{t_b - \mu_b}{\sigma_b}\right) + \sigma_r\left(\frac{t_r - \mu_r}{\sigma_r}\right) \\ t_o &= \mu_o + \sigma_s(Z_s) + \sigma_b(Z_b) + \sigma_r(Z_r) \qquad (3.4) \end{aligned}$$

If the standard deviations are not the same in each triathlon component, then the components with the largest standard deviations have the largest influence on the overall triathlon time. For example, the data set described in Section 3.2 estimates the standard deviation in each of the race components of the an Ironman to be 8 minutes for the swim, 23 minutes for the bike, 23 minutes for the run. So, in a Half Ironman triathlon, if a triathlete finishes one standard deviation better than the mean in the swim, the overall time improves by only eight minutes. However, if the triathlete improves by one standard deviation in the bike or the run, the overall time improves by about 23 minutes.

To correct for the difference in the standard deviations of triathlon components, triathlon finishing times are modeled as a function of distance. This model is fit three different times—one for swim times, one for bike times, and one for run times. The three models are then used to find the distances of each event that equate the standard deviations.

3.2 Data Description

The data are a collection of finishing times from 55 different triathlons found on several different internet sites. The data set contains triathlons from 1996 to 2003. Table 3.1 contains information on the length of each triathlon component, the number of triathlons, and the number triathletes for each triathlon type in the data set. By far, the most numerous triathlons in the data set are the three standard distance triathlons—Olympic, Half Ironman, and Ironman. However, the shorter-distance sprint triathlons and the longer-distance double and triple Ironman triathlons allow prediction of standard deviations at shorter or longer distances when constructing the fair triathlon.

3.3 Model

This section describes the model used to fit the triathlon finishing times. Each event in a triathlon is modeled independently; therefore, the model described in this section is used to fit the data three different times—once to the swim times, once to the bike times, and once to the run times. The model can be broken down into several major parts—the sampling density, the prior distributions, posterior calculation, posterior summaries, and model checking.

3.3.1 Sampling Density Model

The sampling density in any model is the population density from which the observations in the data are drawn. Thus, the sampling density should “match”

Table 3.1: Types of triathlons in the dataset and their respective distances in miles for the swim, bike, and run. “Number” is the number of triathlons of that type in the data set. “Triathletes” is the number of triathletes in the data set who recorded a finishing time in that type of triathlon.

Type	Swim	Bike	Run	Number	Triathletes
Sprint 1	0.249	10.00	3.00	1	269
Sprint 2	0.310	11.80	3.10	2	1,246
Sprint 3	0.500	20.00	6.20	2	60
Sprint 4	0.466	12.40	3.10	1	50
Sprint 5	0.249	9.32	3.10	1	116
Olympic	0.930	24.86	6.21	13	5,686
Half Ironman	1.200	56.00	13.10	10	5,168
Ironman	2.400	112.00	26.20	15	13,427
Double Ironman	4.800	224.00	52.40	5	53
Triple Ironman	7.200	336.00	78.60	5	30

the data in terms of its general shape and its parameters. The shape of the distribution is discussed in section 3.3.1.1, and the parameters of the distribution—which must be modeled according to distance—are discussed in sections 3.3.1.2 and 3.3.1.3.

3.3.1.1 General Shape

The shape of the sampling density should approximately match the "shape" of the data. Figure 3.1 is a plot of finishing times for swimming, cycling, and running from three different triathlon types—the Olympic, Half Ironman, and Ironman. As Figure 3.1 shows, finishing times in all three events and across triathlons of different lengths are clearly nonnormal. In all cases, the data are right skewed and, in some cases, have fairly thick tails. Thus, the sampling density for the model must also be skewed and have flexibility in its tail thickness.

The three-parameter lognormal distribution satisfies both concerns. The lognormal distribution is right skewed, and the three-parameter lognormal distribution has an extra parameter α (sometimes called the "peakedness parameter," see Gajewski, Sedwick, and Antonelli (2004)) that allows for greater thickness in the tails of the distribution. This peakedness parameter makes the three-parameter lognormal distribution a superior choice to the two-parameter lognormal distribution. The two-parameter lognormal distribution lacks the flexibility to simultaneously increase the density in both tails while keeping the same general shape. Simply changing the parameter σ^2 in the two-parameter lognormal

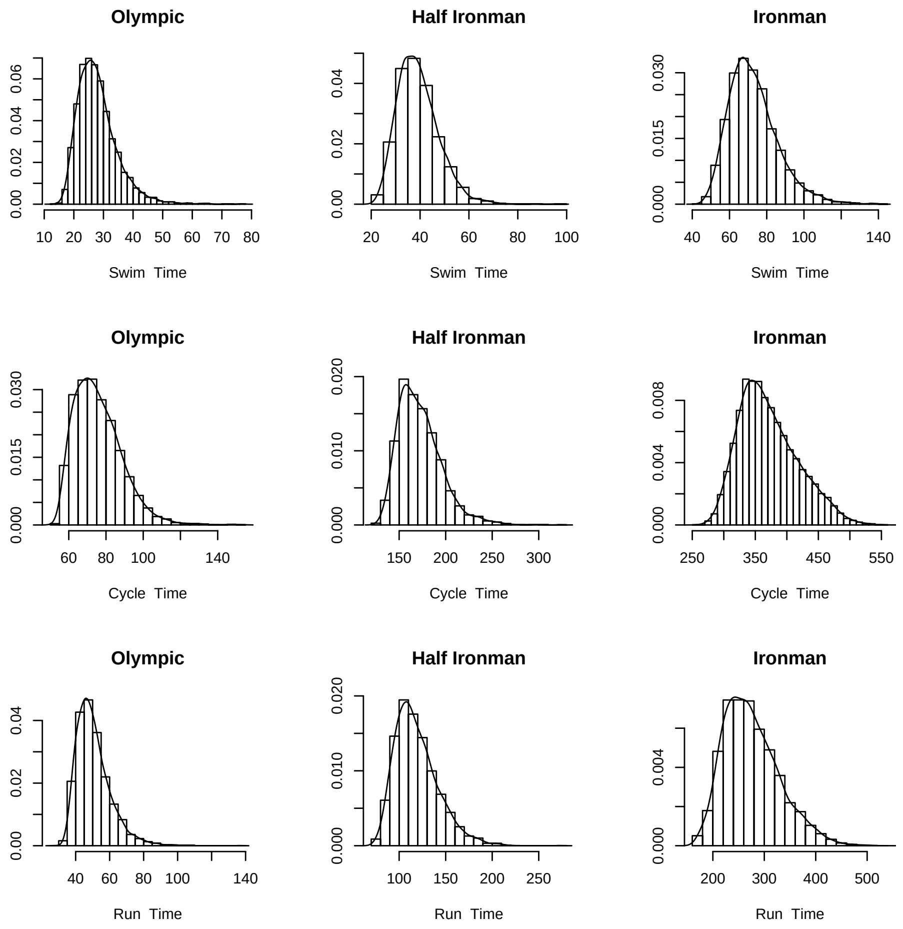

Figure 3.1: Histograms of the finishing times for each of the three events in the Olympic, Half Ironman, and Ironman triathlons in the data set. Each histogram is fit with a density smoother to highlight the shape of the data.

distribution results in an extremely skewed distribution that clearly does not fit the triathlon data.

Figure 4.2 is a plot of three lognormal distributions. Each distribution in the figure has $E[X] = 40$; but each distribution has a different value of the peakedness parameter α. The three-parameter lognormal distribution with peakedness of zero is simply the two-parameter lognormal distribution. The figure shows that, as the peakedness parameter increases, more of the density shifts into the tails of the distribution.

The density of the three-parameter lognormal is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}(x+\alpha)} e^{-\frac{1}{2\sigma^2}[\ln(x+\alpha)-\mu]^2}. \tag{3.5}$$

The parameter α in (3.5) is the peakedness parameter, μ is the mean of the distribution of $\ln(X+\alpha)$, and σ^2 is the variance of the distribution of $\ln(X+\alpha)$. The expected value and the variance of the three-parameter lognormal distribution are

$$E(X) = e^{\mu+\sigma^2/2} - \alpha \tag{3.6}$$

$$V(X) = e^{2\mu+\sigma^2}(e^{\sigma^2} - 1) \tag{3.7}$$

The square root of equation (3.7) is ultimately used for constructing the fair triathlon. With the model for μ discussed in Section 3.3.1.2, equation (3.7) becomes a function of distance and can then be used to find distances in each event that give the same standard deviation.

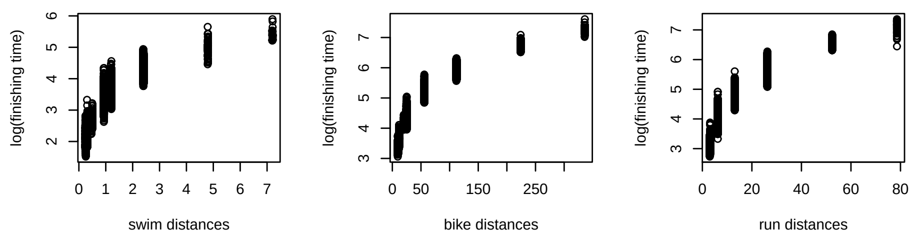

Figure 3.2: Plots of the logarithm of event times versus distance. Scales on the x and y axes are different for each plot.

3.3.1.2 Central Moment Formulation

If X follows a lognormal distribution, then the parameter μ in the lognormal density is the mean of $\ln(X)$. If X follows a three parameter lognormal distribution, then the parameter μ is the mean of $\ln(X + \alpha)$, where α is the peakedness parameter. Thus, any model for μ should closely fit the logarithm of the data. Figure 3.2 shows plots of the logarithm of finishing times for the three triathlon events versus distance. The plots show a definite nonlinear trend in the means for swimming, cycling, and running. Thus, any model for μ should reflect the nonlinear relationship between the log of finishing time and distance.

Schnute (1981) presents a general class of growth models that are appealing for nonlinear models. A special case of these growth models is used for modeling μ. This model is

$$\mu = \tau_1 + (\tau_2 - \tau_1)\frac{1 - e^{-\beta(d-\delta_1)}}{1 - e^{-\beta(\delta_2-\delta_1)}}, \tag{3.8}$$

where τ_1 is the log of the sum of the peakedness parameter α and the time it takes to complete an event (either swim, cycle, or run) of distance δ_1, τ_2 is the log of the sum of the peakedness parameter α and the time it takes to complete an event of distance δ_2, β is the growth rate of finishing times according to distance, and d is the actual distance of the event.

The model in equation (4.8) is particularly nice for Bayesian modeling because of the interpretability of the parameters τ_1 and τ_2. The parameters τ_1 and τ_2 are the log of the sum of $\alpha$and finishing times for distances of δ_1 and δ_2. This interpretability makes specifying prior distributions more manageable.

Although not as easily interpretable as τ_1 and τ_2, the parameter β has some nice properties that are useful in prior elicitation. Larger values of β give longer finishing times for shorter distances than smaller values of β. So, for example, the growth rate for swim times as a function of distance should be much larger than the growth rate for running times or cycling times.

Also, the value of β determines the shape of the growth curve. Positive values of β result in growth curves that are concave down. Negative values of β result in growth curves that are concave up. Therefore, beliefs about the shape of the growth in finishing times can be incorporated into the prior distributions for β.

3.3.1.3 Higher Order Moment Formulations

If X follows a three-parameter lognormal distribution, the parameter σ^2 is the variance of $\ln(X + \alpha)$. Thus, any model for σ^2 must approximate the spread of the log of the data. In Figure 3.2, the spread of the log finishing times at each distance is approximately equal. Therefore, σ^2 is modeled as constant for all distances within an event (swimming, cycling, or running).

3.3.1.4 The Likelihood

The likelihood is now determined by equations (3.5) and (3.8). The likelihood is

$$f(x|\mu, \alpha, \sigma^2) = \frac{1}{(x+\alpha)\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2\sigma^2}(\log(x+\alpha)-\mu)^2} \tag{3.9}$$

where, as mentioned previously, x is the finishing time, α is the peakedness parameter, σ^2 is the variance of $\ln(X+\alpha)$, and μ is a function of the distance d of the event according to the growth curve $\tau_1+(\tau_2-\tau_1)\frac{1-e^{-\beta(d-\delta_1)}}{1-e^{-\beta(\delta_2-\delta_1)}}$. The quantities δ_1 and δ_2 are not parameters but are constants that are chosen to be the distances that correspond respectively to parameters τ_1 and τ_2. In each triathlon component, δ_1 and δ_2 were chosen to be the distances corresponding to an Olympic triathlon and an Ironman triathlon, respectively. For example, in the swim component, $\delta_1 = 0.93$ and $\delta_2 = 2.4$.

3.3.2 Prior Distribution Model

The likelihood from equation (3.9) contains five parameters, thus a prior distributions must be specified for τ_1, τ_2, β, α, and $\sigma^2$for the three different events in a triathlon. The choices for the distributional form of each prior distribution are explained in this section. The choices for the parameter values for the prior distributions are also discussed.

The parameters τ_1 and τ_2 are defined as the natural log of the sum of α and finishing times for particular events of distances δ_1 and δ_2. Theoretically, τ_1 and τ_2 can be negative for certain values of finishing times and α (i.e. if the finishing time and α sum to a number between zero and one). However, in each of the triathlon components, the values of δ_1 and δ_2 correspond to the Olympic triathlon distances and the Ironman triathlon distances, respectively. Times for competitors at these distances always exceed one minute. Therefore, the quantities τ_1 and τ_2 are always positive quantities. The gamma distribution is defined for positive real numbers, and the gamma distribution is also flexible in its shape—it can be symmetric or skewed. Therefore, we use gamma prior distributions for τ_1 and τ_2 in each event.

The peakedness parameter α and σ^2 in the three-parameter lognormal distribution are both defined as a positive quantities. Again, the gamma distribution is defined for positive real numbers and offers flexibility in its shape. Thus, as with τ_1 and τ_2 we use gamma prior distributions for α and σ^2.

The parameter β represents the "growth rate" of finishing times. Theoretically, β can be either positive or negative, which suggests using a prior distribution defined on the entire real line. Also, a priori, there is no reason to believe that β is more likely take on larger values than smaller values (or smaller values than larger values), which suggests using a symmetric prior distribution. Because of these considerations, a normal prior distribution is used for β.

Table 3.2: Table of parameter values for prior distributions for all three portions of a triathlon.

Prior Parameter	Swim	Cycle	Run
a_{τ_1}	1156.0	529.0	430.0
b_{τ_1}	340.0	115.0	104.0
a_{τ_2}	900.0	1600.0	529.0
b_{τ_2}	200.0	267.0	92.0
a_α	4.0	4.0	4.0
b_α	0.2	0.2	0.2
a_σ	9.0	1.7	4.0
b_σ	300.0	130.0	200.0
m_β	0.6	0.1	0.05
s_β^2	0.04	0.25	2.25×10^{-4}

The prior distributions for each parameter are

$$\begin{aligned}
\pi(\tau_1) &\propto {\tau_1}^{a_{\tau_1}-1}e^{-\tau_1 b_{\tau_1}} \\
\pi(\tau_2) &\propto {\tau_2}^{a_{\tau_2}-1}e^{-\tau_2 b_{\tau_2}} \\
\pi(\beta) &\propto e^{-\frac{\left(\beta-m_\beta\right)^2}{2s_\beta^2}} \\
\pi(\sigma^2) &\propto (\sigma^2)^{a_\sigma-1}e^{-\sigma^2 b_\sigma} \\
\pi(\alpha) &\propto \alpha^{a_\alpha-1}e^{-\alpha b_\alpha}
\end{aligned}$$

where the values for the prior parameters are listed in Table 3.2 and the values for the expected values and standard deviations of the prior distributions are listed in Table 3.3.

The values in Table 3.2 were chosen using moment matching. In other words, values for the mean and standard deviation of the prior distribution were chosen first. Then the mean and standard deviation were used to solve for the actual parameters of the prior distributions.

Table 3.3: Table of means and standard deviations for prior distributions of each parameter in the model. Distances δ_1 and δ_2 (in miles) are also listed for the parameters τ_1 and τ_2.

	Swim		Bike		Run	
Prior Parameter	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
	δ_1= 0.93		δ_1= 24.86		δ_1= 6.21	
τ_1	3.4	0.1	4.6	0.2	4.15	0.2
	δ_2= 2.4		δ_2= 112.0		δ_2= 26.2	
τ_2	4.5	0.15	4.6	0.2	5.75	0.25
α	20.0	10.0	20.0	10.0	20.0	10.0
σ^2	0.03	0.01	0.013	0.01	0.02	0.01
β	0.60	0.20	0.02	0.01	0.03	0.015

Values for the prior distribution of τ_1 were chosen based on beliefs about an average triathlete. Because τ_1 is the log of the sum of α and the time it takes to finish a race of distance δ_1, we chose our prior for τ_1 based on the log of the time we believed it would take an average triathlete to swim, bike, or run a distance of δ_1, where values for δ_1 for each component are the Olympic triathlon distances—0.93 mile swim, 24.86 mile bike, and a 6.21 mile run. By taking the log of various plausible values for finishing times, we obtained an approximate mean and variance for the prior distributions of τ_1 in each of the triathlon components. For example, based on our prior knowledge, we felt an average triathlete might take as little as 20 minutes or as long as 40 minutes to swim 1500 meters in an Olympic triathlon. Taking the log of 20 and 40 minutes suggests a prior distribution for τ_1 with a mean of 3.4 and a standard deviation of 0.01. The mean and the variance for the distributions of τ_1 in the cycling and running events were obtained in a similar fashion, and parameters for the gamma prior distributions were solved using these means and standard deviations. Values for the parameters in the prior distributions for τ_2 were chosen similarly.

The parameter σ^2 is the variance of the distribution of $\ln(X+\alpha)$, which can be thought of more simply as the variance of $\ln(X)$. The parameter values for the prior distributions of σ^2 are based on beliefs about the maximum and minimum time it would take a triathlete to swim, bike, or run a specified distance. The distances used for this were 1 mile for the swim, 25 miles for the bike, and 6 miles for the run (which are the approximate distances of an Olympic triathlon). By using the range of the log of the proposed maximum finishing times and the log of the proposed minimum finishing times, we were able to obtain an estimates of the variance of $\ln(X)$ in each triathlon component by dividing the range by six and squaring the result. For example, in the swim event, an extremely fast swimmer might swim a mile in 15 minutes, and an extremely slow swimmer might take as long as 55 minutes. Or, alternatively, a fast swimmer might take 25 minutes to swim a mile, and a slow swimmer might take 45 minutes to swim a mile. Based on the difference between the log of 15 and the log of 55 divided by 6 and the difference between the log of 25 and the log of 35 divided by six, the prior distribution for σ^2 in the swim event was centered at 0.03 with a standard deviation of 0.01.

The parameter β is the "growth" rate of the finishing times. To choose a prior distribution for β in each of the three triathlon components, we calculated predicted finishing times for several different values of β at two different distances. We looked at the change in the finishing time between the two distances to determine what values of β gave reasonable changes in finishing times for the corresponding change in distance. For example, Table 3.4 contains predicted finishing times for a 1.0 mile and a 1.5 mile swim. These finishing times were computed using equation (3.8) with the means of the prior distributions for τ_1 and τ_2 and several values of β. For $\beta = 0.1$, the increase in time between a 1.0 mile swim and a 1.5 mile swim is 15 minutes. This implies that the average triathlete would finish the third half mile of a 1.5 mile swim at a pace slightly faster than the pace of the first half mile. For $\beta = 1.3$, the increase in time between

Table 3.4: Possible values for β and their corresponding finishing times for a swim of 1.0 mile and 1.5 miles. The means of the prior distributions for τ_1 and τ_2 were used in equation(3.8) along with the values of β in the table to calculate the predicted finishing times.

β	Finish Time for 1 mile	Finish Time for 1.5 miles	Difference
0.1	32	47	15
0.3	32	49	17
0.5	32	51	18
0.7	33	53	20
0.9	33	55	22
1.1	33	57	24
1.3	34	59	25

a 1.0 mile swim and 1.5 mile swim is 25 minutes. This implies that the average triathlete would finish the third half mile of a 1.5 mile race at a pace nearly twice as slow as the pace of the first half mile. Both of the above scenarios are unlikely; thus, a normal prior distribution with a mean of 0.6 and a standard deviation of 0.2 was chosen for β in the swim model. This prior distribution places most of the density over the more plausible intermediary values of β.

A priori, we knew very little about the parameter α. However, we believed the parameter to be nonzero in each event. We chose a value for α (based on several plots of the three-parameter lognormal distribution—see Figure 4.2) to be the mean of the prior distributions for α and chose a large variance for each distribution. A priori, we had no reason to believe that the peakedness parameter in each event should be different from the peakedness parameter in another event. Thus, the prior distribution for α in each event has a mean of 20 and a variance of 100.

3.3.3 Computation

The joint posterior distribution of τ_1, τ_2, β, α, and σ^2 has no closed form solution, so a simulation approach was used to approximate the posterior distribution. Gibbs sampling, as used by Gelfand and Smith (1990), is an algorithm that is especially suited to statistical problems that involve multiple parameters. The Gibbs sampling algorithm requires the use of complete conditional distributions for each parameter. As explained in chapter 2, the complete conditional distribution for a parameter θ_j is the distribution of $\theta_j|\, \theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_k$. For the model of the triathlon data, the complete conditionals are as follows, where the notation $[\theta]$ means the distribution of $\theta_j|\, \theta_1, \ldots, \theta_{j-1}, \theta_{j+1}, \ldots, \theta_k$:

$$\begin{aligned}
[\tau_1] &\propto \tau_1^{a_{\tau_1}-1} \exp\left\{-\tau_1 b_{\tau_1} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\ln(x_i+\alpha)-\mu\right]^2\right\} \\
[\tau_2] &\propto \tau_2^{a_{\tau_2}-1} \exp\left\{-\tau_2 b_{\tau_2} - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\ln(x_i+\alpha)-\mu\right]^2\right\} \\
[\beta] &\propto \exp\left\{-\frac{1}{2s^2}(\beta-m)^2 - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\ln(x_i+\alpha)-\mu\right]^2\right\} \\
\left[\sigma^2\right] &\propto (\sigma^2)^{a_\sigma-1} \exp\left\{-\sigma^2 b_\sigma - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\ln(x_i+\alpha)-\mu\right]^2\right\} \\
[\alpha] &\propto \alpha^{a_\alpha-1}\prod_{i=1}^{n}(x_i+\alpha)^{-1} \exp\left\{-\alpha b_\alpha - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left[\ln(x_i+\alpha)-\mu\right]^2\right\}
\end{aligned}$$

These complete conditional distributions were coded in C and used to obtain draws from the posterior distributions of each parameter in each event.

3.3.4 Posterior Summaries

The results of a Bayesian statistical model can be summarized by descriptive statistics on the posterior distributions of the parameters. Summary statistics for posterior distributions should include measures of central tendency and measures of spread. Thus, for the triathlon model, posterior means and standard deviations were calculated and reported for each parameter in the model in each of the three triathlon components. Credible intervals give a range of possible values for the "true" value of the parameter at a certain probability. Thus, credible intervals were also computed for each parameter in the model in each of the triathlon components.

3.3.5 Model Checking

For any statistical model, the model must be checked to see if it appropriately fits the data. Normal probability plots were used to check the fit of the model for triathlon finishing times. For a random variable X distributed as a three-parameter lognormal, the distribution of $\ln(X+\alpha)$ is normal. Thus normal probability plots were constructed for the residuals in each triathlon component. In the triathlon model, the residuals are defined as

$$\ln(X+\hat{\alpha})-\hat{\mu}, \tag{3.10}$$

where $\hat{\alpha}$ is the posterior mean for α, $\hat{\mu}=\tau_1+(\hat{\tau}_2-\hat{\tau}_1)\frac{1-e^{-\hat{\beta}(d-\delta_1)}}{1-e^{-\hat{\beta}(\delta_2-\delta_1)}}$, and $\hat{\tau}_1$, $\hat{\tau}_2$, and $\hat{\beta}$ are the posterior means for τ_1, τ_2, and β.

Chapter 4

Results

This chapter is a paper that was submitted to the Western North American Region of the International Biometric Society (WNAR) on May 25, 2004, for a student paper competition. The paper summarizes the major results of the model described in Chapter 3.

4.1 Introduction

According to popular myth[1], the first Ironman triathlon was conceived in a bar in Hawaii. Chatting over a few beers, Naval Commander John Collins debated with several others the following question: "Who are the toughest athletes—swimmers, cyclists, or runners." Collins suggested that to answer the question athletes from each discipline should compete in an event that consists of all three sports run back-to-back. Collins proposed a race comprised of the longest events in swimming, cycling, and running then contested in Hawaii—the 2.4 mile Waikiki Rough Water Swim, the 112 mile Around Oahu Bike Ride, and the 26.2 mile Honolulu Marathon. So, in 1978, Collins and a little over a dozen others participated in the first Ironman triathlon.

The popularity of the Ironman triathlon increased dramatically after the second Hawaii Ironman when *Sports Illustrated* ran a story on the 1979 winner Tom Warren and ABC began coverage of the Hawaii triathlon in 1980. Soon, other shorter-distance triathlons emerged. The Half-Ironman triathlon was formed by taking half the distances of each event in the Ironman triathlon (1.2 mile swim, 56 mile bike, and a 13.1 mile run). The International Triathlon Union (ITU) formed the Olympic triathlon by combining the longest event distances used in races sanctioned by each of the major federations for swimming (the International Federation for Swimming), cycling (International Cycling Union), and track (International Association of Athletics Federations).[2]The Olympic Triathlon consists of a 1.5 km (0.93 miles) swim, a 40 km (24.86 miles) bike, and a 10 km (6.21 miles) run.

Today, the three major triathlon types are the Olympic, Half-Ironman, and Ironman triathlons. However, triathletes can compete in a plethora of other triathlons of different distances. Sprint triathlons are composed of events of shorter distances than the Olympic triathlon. Triathletes also compete in Double and Triple Ironman triathlons, which—as their names imply—are twice and three times the distance of the Ironman triathlon.

As the history of the modern triathlon indicates, little consideration was

[1] For Collins' comments and clarifications on the genesis of the Ironman triathlon see `http://vnews.ironmanlive.com/vnews//1043361628/?keywords=john:collins`

[2] See `http://www.triathlon.org/tv/FAQs.htm`.

given to the relative importance of each event when forming the various triathlons. In fact, empirical evidence indicates that the swimming portion of the modern triathlon is extremely underweighted. Dengel et al. (1989), in a study of eleven male triathletes, report that, "swimming time was not significantly related to overall triathlon time."

In a fair triathlon, a one standard deviation improvement in time in any of the three triathlon components results in the same improvement in overall time. As described in Section 4.2, the standard deviations in each component of a fair trithlon are equal. Because finishing times in each triathlon component increase with distance, a fair tritathlon can be constructed by finding distances for each event such that event standard deviations are the same. Thus, a statistical model for triathlon finishing times can be used to find the appropriate distances that equate the standard deviations in each triathlon component. In this paper we present a Bayesian model for finishing times in each of the components of the modern triathlon. We use this model to construct several "fair" triathlons of different distances and provide a general ratio of event distances for constructing fair triathlons.

4.2 The Fair Triathlon Defined

The fair triathlon is defined as a triathlon in which a one standard deviation improvement in time in any of the individual triathlon components results in the same improvement in overall triathlon time. This "fair" result can happen if the standard deviations in each of the triathlon components are the same—that is, if the standard deviations of finishing times in swimming, cycling, and running are equal. For example, the overall finishing time t_o for an arbitrary triathlete in an arbitrary triathlon can be written

$$t_o = t_s + t_b + t_r$$

where t_o is the triathlete's finishing time for the entire triathlon and t_s, t_b, and t_r are the triathlete's finishing times for swimming, cycling, and running respectively. Adding and subtracting the mean overall finishing time μ_o for all triathletes in the triathlon gives

$$t_o = \mu_o + t_s + t_b + t_r - \mu_s - \mu_b - \mu_r \tag{4.1}$$

where $\mu_o = \mu_s + \mu_b + \mu_r$ and μ_s, μ_b, and μ_r are the mean finishing times in swimming, cycling, and running for all triathletes the triathlon. In the fair triathlon, the standard deviations in each of the three events are equal. We represent this standard deviation by σ. Then rearranging the terms and multiplying by one ($\frac{\sigma}{\sigma}$) gives

$$\begin{aligned} t_o &= \mu_o + \frac{\sigma}{\sigma}\left(t_s - \mu_s + t_b - \mu_b + t_r - \mu_r\right) && (4.2) \\ t_o &= \mu_o + \sigma\left(\frac{t_s - \mu_s}{\sigma} + \frac{t_b - \mu_b}{\sigma} + \frac{t_r - \mu_r}{\sigma}\right) \\ t_o &= \mu_o + \sigma\left(Z_s + Z_b + Z_r\right). && (4.3) \end{aligned}$$

Thus, if a triathlete finishes one standard deviation faster than the mean time in any of the triathlon components, equation 4.3 becomes one of

$$\begin{aligned} t_o &= \mu_o + \sigma\left(-1+0+0\right) \\ t_o &= \mu_o + \sigma\left(0-1+0\right) \\ t_o &= \mu_o + \sigma\left(0+0-1\right) \end{aligned}$$

and the improvement in the overall time is the same ($-\sigma$) in each case.

If the standard deviations in each event are not the same, it is not possible to distribute a common standard deviation in equation (4.2) to obtain equation (4.3). However, rearranging and multiplying the terms on the right of equation (4.1) by one gives

$$\begin{aligned} t_o &= \mu_o + \frac{\sigma_s}{\sigma_s}\left(t_s-\mu_s\right) + \frac{\sigma_b}{\sigma_b}\left(t_b-\mu_b\right) + \frac{\sigma_r}{\sigma_r}\left(t_r-\mu_r\right) \\ t_o &= \mu_o + \sigma_s\left(\frac{t_s-\mu_s}{\sigma_s}\right) + \sigma_b\left(\frac{t_b-\mu_b}{\sigma_b}\right) + \sigma_r\left(\frac{t_r-\mu_r}{\sigma_r}\right) \\ t_o &= \mu_o + \sigma_s\left(Z_s\right) + \sigma_b\left(Z_b\right) + \sigma_r\left(Z_r\right) \qquad (4.4) \end{aligned}$$

If the standard deviations are not the same in each triathlon component, then the components with the largest standard deviations have the largest influence on the overall triathlon time. For example, the data set described in Section 4.3 estimates the standard deviation in each of the race components of the an Ironman to be 8 minutes for the swim, 23 minutes for the bike, 23 minutes for the run. So, in a Half Ironman triathlon, if a triathlete finishes one standard deviation better than the mean in the swim, the overall time improves by only eight minutes. However, if the triathlete improves by one standard deviation in the bike or the run, the overall time improves by about 23 minutes.

To correct for the difference in the standard deviations of triathlon components, triathlon finishing times are modeled as a function of distance. This model is fit three different times—one for swim times, one for bike times, and one for run times. The three models are then used to find the distances of each event that equate the standard deviations.

4.3 Data Description

The data are a collection of finishing times from 55 different triathlons found on several different internet sites. The data set contains triathlons from 1996 to 2003. Table 4.1 contains information on the length of each triathlon component, the number of triathlons, and the number triathletes for each triathlon type in the data set. By far, the most numerous triathlons in the data set are the three standard distance triathlons—Olympic, Half Ironman, and Ironman. However, the shorter-distance sprint triathlons and the longer-distance double and triple Ironman triathlons allow prediction of standard deviations at shorter or longer distances when constructing the fair triathlon.

4.4 The Model

This section describes the model used to fit the triathlon finishing times. Each event in a triathlon is modeled independently; therefore, the model described

Table 4.1: Types of triathlons in the dataset and their respective distances in miles for the swim, bike, and run. "Number" is the number of triathlons of that type in the data set. "Triathletes" is the number of triathletes in the data set who recorded a finishing time in that type of triathlon.

Type	Swim	Bike	Run	Number	Triathletes
Sprint 1	0.249	10.00	3.00	1	269
Sprint 2	0.310	11.80	3.10	2	1,246
Sprint 3	0.500	20.00	6.20	2	60
Sprint 4	0.466	12.40	3.10	1	50
Sprint 5	0.249	9.32	3.10	1	116
Olympic	0.930	24.86	6.21	13	5,686
Half Ironman	1.200	56.00	13.10	10	5,168
Ironman	2.400	112.00	26.20	15	13,427
Double Ironman	4.800	224.00	52.40	5	53
Triple Ironman	7.200	336.00	78.60	5	30

in this section is used to fit the data three different times—once to the swim times, once to the bike times, and once to the run times.

4.4.1 Sampling Density Model

The sampling density in any model is the population density from which the observations in the data are drawn. Thus, the sampling density should "match" the data in terms of its general shape and its parameters. The shape of the distribution is discussed in section 4.4.1.1, and the parameters of the distribution—which must be modeled according to distance—are discussed in sections 4.4.1.2 and 4.4.1.3.

4.4.1.1 General Shape

The shape of the sampling density should approximately match the shape of the data. Figure 4.1 is a plot of finishing times for swimming, cycling, and running from three different triathlon types—the Olympic, Half Ironman, and Ironman. As Figure 4.1 shows, finishing times in all three events and across triathlons of different lengths are clearly nonnormal. In all cases, the data are right skewed and, in some cases, have fairly thick tails. Thus, the sampling density for the model must also be skewed and have flexibility in its tail thickness.

The three-parameter lognormal distribution satisfies both concerns. The lognormal distribution is right skewed, and the three-parameter lognormal distribution has an extra parameter α (sometimes called the "peakedness parameter,"

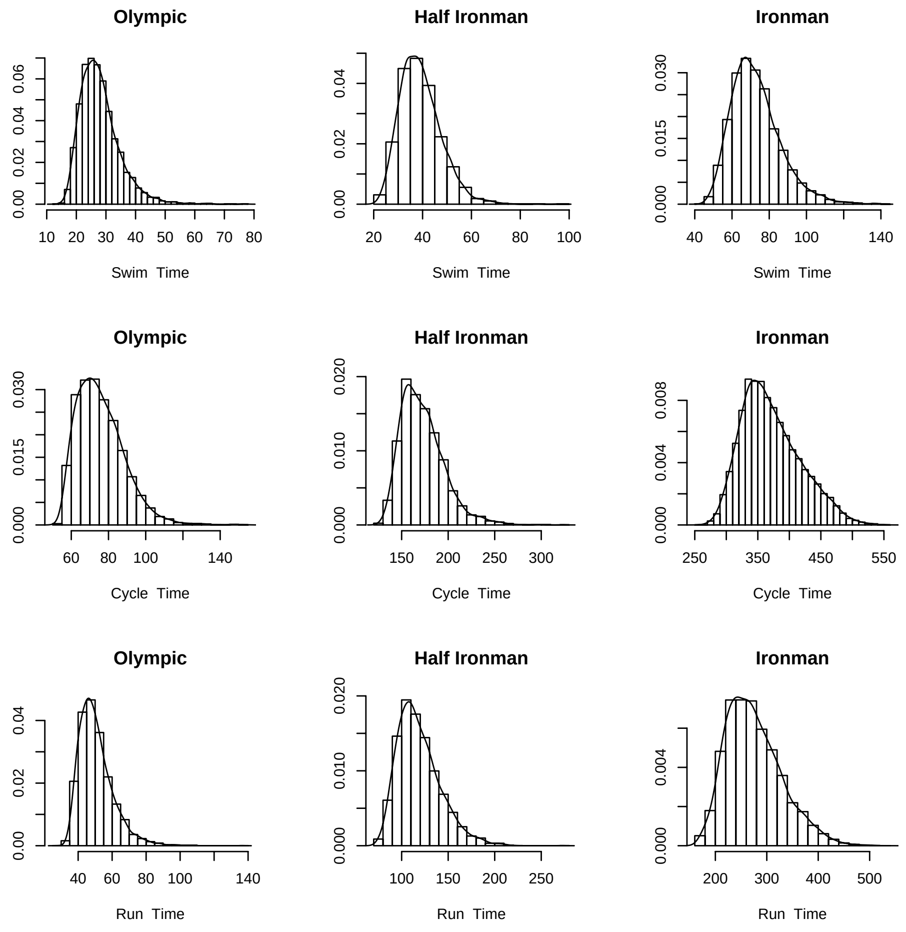

Figure 4.1: Histograms of the finishing times for each of the three events in the Olympic, Half Ironman, and Ironman triathlons in the data set. Each histogram is fit with a density smoother to highlight the shape of the data.

see Gajewski, Sedwick, and Antonelli (2004)) that allows for greater thickness in the tails of the distribution. This peakedness parameter makes the three-parameter lognormal distribution a superior choice to the two-parameter lognormal distribution. The two-parameter lognormal distribution lacks the flexibility to simultaneously increase the density in both tails while keeping the same general shape. Simply changing the parameter σ^2 in the two-parameter lognormal distribution results in an extremely skewed distribution that clearly does not fit the triathlon data.

Figure 4.2 is a plot of three lognormal distributions. Each distribution in the figure has $E[X] = 40$; but each distribution has a different value of the peakedness parameter α. The three-parameter lognormal distribution with peakedness of zero is simply the two-parameter lognormal distribution. The figure shows that, as the peakedness parameter increases, more of the density shifts into the tails of the distribution.

The density of the three-parameter lognormal is

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}(x+\alpha)} e^{-\frac{1}{2\sigma^2}[\ln(x+\alpha)-\mu]^2} \tag{4.5}$$

where parameter α in (4.5) is the peakedness parameter, μ is the mean of the distribution of $\ln(X+\alpha)$, and σ^2 is the variance of the distribution of $\ln(X+\alpha)$. The expected value and the variance of the three-parameter lognormal distribution

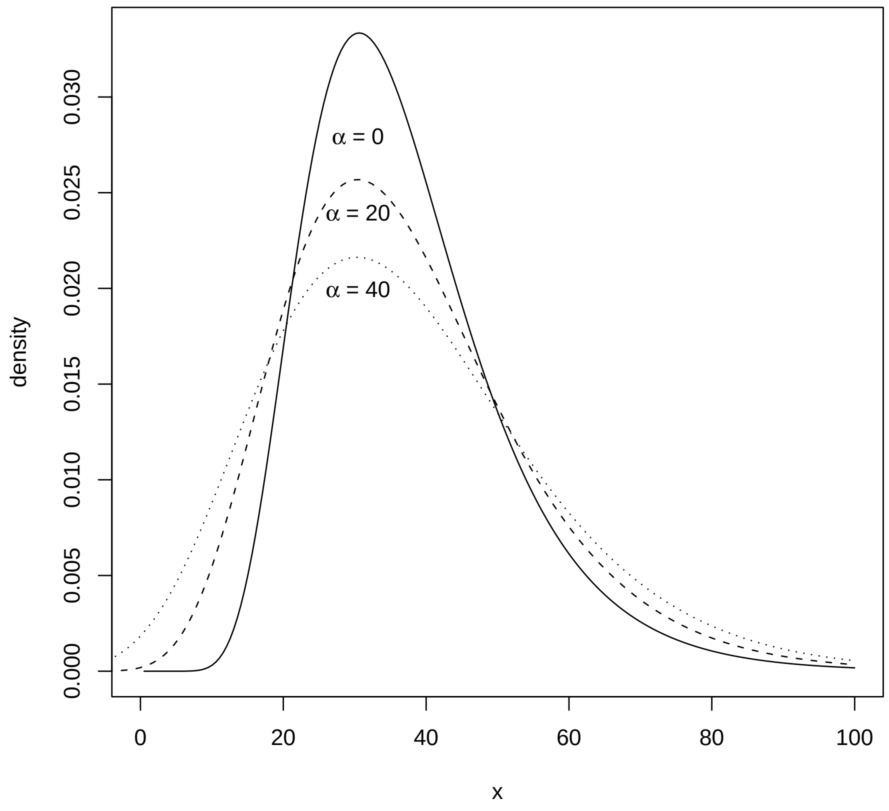

Figure 4.2: The three parameter lognormal distribution with three different values for the peakedness parameter α and $E[X] = 40$.

are

$$\begin{aligned} E(X) &= e^{\mu+\sigma^2/2} - \alpha & (4.6) \\ V(X) &= e^{2\mu+\sigma^2}(e^{\sigma^2} - 1) & (4.7) \end{aligned}$$

The square root of equation (4.7) is ultimately used for constructing the fair triathlon. With the model for μ discussed in Section 4.4.1.2, equation (4.7) becomes a function of distance and can then be used to find distances in each event that give the same standard deviation.

4.4.1.2 Central Moment Formulation

If X follows a lognormal distribution, then the parameter μ in the lognormal density is the mean of $\ln(X)$. If X follows a three-parameter lognormal distribution, then the parameter μ is the mean of $\ln(X + \alpha)$, where α is the peakedness parameter. Thus, any model for μ should closely fit the logarithm of the data. Figure 4.3 shows plots of the log of finishing times for the three triathlon events versus distance. The plots show a definite nonlinear trend in the means for swimming, cycling, and running. Thus, any model for μ should reflect the nonlinear relationship between the log of finishing time and distance.

Schnute (1981) presents a general class of growth models that are appealing for nonlinear models. A special case of these growth models is used for modeling μ. This model is

$$\mu = \tau_1 + (\tau_2 - \tau_1)\frac{1 - e^{-\beta(d-\delta_1)}}{1 - e^{-\beta(\delta_2-\delta_1)}}, \tag{4.8}$$

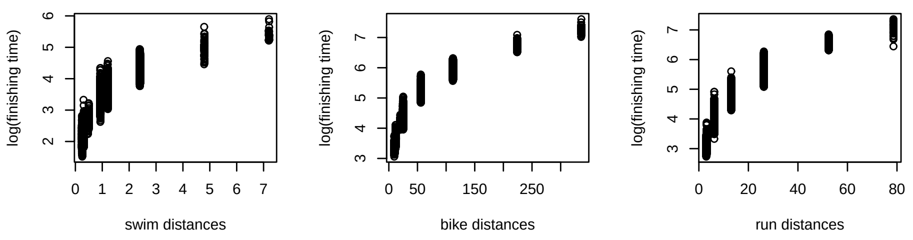

Figure 4.3: Plots of the logarithm of event times versus distance. Scales on the x and y axes are different for each plot.

where τ_1 is the log of the sum of the peakedness parameter α and the time it takes to complete an event (either swim, cycle, or run) of distance δ_1, τ_2 is the log of the sum of the peakedness parameter α and the time it takes to complete an event of distance δ_2, β is the growth rate of finishing times according to distance, and d is the actual distance of the event.

The model in equation (4.8) is particularly nice for Bayesian modeling because of the interpretability of the parameters—τ_1 and τ_2. The parameters τ_1 and τ_2 are the log of the sum of $\alpha$and finishing times for distances of δ_1 and δ_2. This interpretability makes specifying prior distributions more manageable.

Although not as easily interpretable as τ_1 and τ_2, the parameter β has some nice properties that useful in prior elicitation. Larger values of β give longer finishing times for shorter distances than smaller values of β. So, for example, the growth rate for swim times as a function of distance should be much larger than the growth rate for cycling times.

Also, the value of β determines the shape of the growth curve. Positive values of β result in growth curves that are concave down. Negative values of β result in growth curves that are concave up. Therefore, beliefs about the shape of the growth in finishing times can be incorporated into the prior distributions for β.

4.4.1.3 Higher Order Moment Formulations

If X follows a three-parameter lognormal distribution, the parameter σ^2 is the variance of $\ln(X + \alpha)$. Thus, any model for σ^2 must approximate the spread of the log of the data. In Figure 4.3, the spread of the log finishing times at each distance is approximately equal. Therefore, σ^2 is modeled as constant for all distances within an event (swimming, cycling, or running).

4.4.1.4 The Likelihood

The likelihood is now determined by equations (4.5) and (4.8). The likelihood is

$$f(x|\mu, \alpha, \sigma^2) = \frac{1}{(x+\alpha)\sqrt{2\pi\sigma^2}} \; e^{-\frac{1}{2\sigma^2}(\log(x+\alpha)-\mu)^2} \tag{4.9}$$

where, as mentioned previously, x is the finishing time, α is the peakedness parameter, σ^2 is the variance of $\ln(X + \alpha)$, and μ is a function of the distance d of the event according to the growth curve $\tau_1 + (\tau_2 - \tau_1)\frac{1-e^{-\beta(d-\delta_1)}}{1-e^{-\beta(\delta_2-\delta_1)}}$. The quantities δ_1 and δ_2 are not parameters but are constants that are chosen to be the distances that correspond respectively to parameters τ_1 and τ_2. In each triathlon component δ_1 and δ_2 are the distances corresponding to an Olympic triathlon and an

Ironman triathlon, respectively. For example, in the swim component, $\delta_1 = 0.93$ and $\delta_2 = 2.4$.

4.4.2 Prior Distribution Model

The likelihood from equation (4.9) contains five parameters, thus a prior distributions must be specified for τ_1, τ_2, β, α, and $\sigma^2$for the three different events in a triathlon. The choices for the distributional form of each prior distribution are explained in this section. The choices for the parameter values for the prior distributions are also discussed.

The parameters τ_1 and τ_2 are defined as the natural log of the sum of α and finishing times for particular events of distances δ_1 and δ_2. Theoretically, τ_1 and τ_2 can be negative for certain values of finishing times and α (i.e. if the finishing time and α sum to a number between zero and one). However, in each of the triathlon components, the values of δ_1 and δ_2 correspond to the Olympic triathlon distances and the Ironman triathlon distances, respectively. Times for competitors at these distances always exceed one minute. Therefore, the quantities τ_1 and τ_2 are always positive quantities. The gamma distribution is defined for positive real numbers, and the gamma distribution is also flexible in its shape—it can be symmetric or skewed. Therefore, we use gamma prior distributions for τ_1 and τ_2 in each event.

The peakedness parameter α and σ^2 in the three-parameter lognormal distribution are both defined as a positive quantities. Again, the gamma distribution is defined for positive real numbers and offers flexibility in its shape. Thus, as with τ_1 and τ_2 we use gamma prior distributions for α and σ^2.

The parameter β represents the "growth rate" of finishing times. Theoretically, β can be either positive or negative, which suggests using a prior distribution defined on the entire real line. Also, a priori, there is no reason to believe that β is more likely take on larger values than smaller values (or smaller values than larger values), which suggests using a symmetric prior distribution. Because of these considerations, a normal prior distribution is used for β.

The prior distributions for each parameter are

$$\begin{aligned}
\pi(\tau_1) &\propto {\tau_1}^{a_{\tau_1}-1}e^{-\tau_1 b_{\tau_1}} \\
\pi(\tau_2) &\propto {\tau_2}^{a_{\tau_2}-1}e^{-\tau_2 b_{\tau_2}} \\
\pi(\beta) &\propto e^{-\frac{(\beta-m)^2}{2s^2}} \\
\pi(\sigma^2) &\propto (\sigma^2)^{a_\sigma-1}e^{-\sigma^2 b_\sigma} \\
\pi(\alpha) &\propto \alpha^{a_\alpha-1}e^{-\alpha b_\alpha}
\end{aligned}$$

where the values for the prior parameters are listed in Table 4.2.

The values in Table 4.2 were chosen using moment matching. In other words, values for the mean and standard deviation of the prior distribution were chosen first. Then the mean and standard deviation were used to solve for the actual parameters of the prior distributions.

Values for the prior distribution of τ_1 were chosen based on beliefs about an average triathlete. Because τ_1 is the log of the sum of α and the time it takes to finish a race of distance δ_1, we chose our prior for τ_1 based on the log of

Table 4.2: Table of parameter values for prior distributions for all three portions of a triathlon.

Prior Parameter	Swim	Cycle	Run
a_{τ_1}	1156.0	529.0	430.0
b_{τ_1}	340.0	115.0	104.0
a_{τ_2}	900.0	1600.0	529.0
b_{τ_2}	200.0	267.0	92.0
a_α	4.0	4.0	4.0
b_α	0.2	0.2	0.2
a_σ	9.0	1.7	4.0
b_σ	300.0	130.0	200.0
m_β	0.6	0.1	0.05
s^2_β	0.04	0.25	2.25×10^{-4}

Table 4.3: Table of means and standard deviations for prior distributions of each parameter in the model. Distances δ_1 and δ_2 (in miles) are also listed for the parameters τ_1 and τ_2.

	Swim		Bike		Run	
Prior Parameter	Mean	Std Dev	Mean	Std Dev	Mean	Std Dev
	δ_1= 0.93		δ_1= 24.86		δ_1= 6.21	
τ_1	3.4	0.1	4.6	0.2	4.15	0.2
	δ_2= 2.4		δ_2= 112.0		δ_2= 26.2	
τ_2	4.5	0.15	4.6	0.2	5.75	0.25
α	20.0	10.0	20.0	10.0	20.0	10.0
σ^2	0.03	0.01	0.013	0.01	0.02	0.01
β	0.60	0.20	0.02	0.01	0.03	0.015

the time we believed it would take an average triathlete to swim, bike, or run a distance of δ_1, where values for δ_1 for each component are the Olympic triathlon distances—0.93 mile swim, 24.86 mile bike, and a 6.21 mile run. By taking the log of various plausible values for finishing times, we obtained an approximate mean and variance for the prior distributions of τ_1 in each of the triathlon components. For example, based on our prior knowledge, we felt an average triathlete might take as little as 20 minutes or as long as 40 minutes to swim 1500 meters in an Olympic triathlon. Taking the log of 20 and 40 minutes suggests a prior distribution for τ_1 with a mean of 3.4 and a standard deviation of 0.01. The mean and the variance for the distributions of τ_1 in the cycling and running events were obtained in a similar fashion, and parameters for the gamma prior distributions were solved using these means and standard deviations. Values for the parameters in the prior distributions for τ_2 were chosen similarly.

The parameter σ^2 is the variance of the distribution of $\ln(X+\alpha)$, which can be thought of more simply as the variance of $\ln(X)$. The parameter values for the prior distributions of σ^2 are based on beliefs about the maximum and minimum time it would take a triathlete to swim, bike, or run a specified distance. The distances used for this were 1 mile for the swim, 25 miles for the bike, and 6 miles for the run (which are the approximate distances of an Olympic triathlon). By using the range of the log of the proposed maximum finishing times and the log of the proposed minimum finishing times, we were able to obtain an estimates of the variance of $\ln(X)$ in each triathlon component by dividing the range by six and squaring the result. For example, in the swim event, an extremely fast swimmer might swim a mile in 15 minutes, and an extremely slow swimmer might take as long as 55 minutes. Or, alternatively, a fast swimmer might take 25 minutes to swim a mile, and a slow swimmer might take 45 minutes to swim a mile. Based on the difference between the log of 15 and the log of 55 divided by 6 and the difference between the log of 25 and the log of 35 divided by six, the prior distribution for σ^2 in the swim event was centered at 0.03 with a standard deviation of 0.01.

The parameter β is the "growth" rate of the finishing times. To choose a prior distribution for β in each of the three triathlon components, we calculated predicted finishing times for several different values of β at two different distances. We looked at the change in the finishing time between the two distances to determine what values of β gave reasonable changes in finishing times for the corresponding change in distance. For example, Table 4.4 contains predicted finishing times for a 1.0 mile and a 1.5 mile swim. These finishing times were computed using equation (4.8) with the means of the prior distributions for τ_1 and τ_2 and several values of β. For $\beta = 0.1$, the increase in time between a 1.0 mile swim and a 1.5 mile swim is 15 minutes. This implies that the average triathlete would finish the third half mile of a 1.5 mile swim at a pace slightly faster than the pace of the first half mile. For $\beta = 1.3$, the increase in time between a 1.0 mile swim and 1.5 mile swim is 25 minutes. This implies that the average triathlete would finish the third half mile of a 1.5 mile race at a pace nearly twice as slow as the pace of the first half mile. Both of the above scenarios are unlikely;

Table 4.4: Possible values for β and their corresponding finishing times for a swim of 1.0 mile and 1.5 miles. The means of the prior distributions for τ_1 and τ_2 were used in equation(4.4.1.3) along with the values of β in the table to calculate the predicted finishing times.

β	Finish Time for 1 mile	Finish Time for 1.5 miles	Difference
0.1	32	47	15
0.3	32	49	17
0.5	32	51	18
0.7	33	53	20
0.9	33	55	22
1.1	33	57	24
1.3	34	59	25

thus, a normal prior distribution with a mean of 0.6 and a standard deviation of 0.2 was chosen for β in the swim model. This prior distribution places most of the density over the more plausible intermediary values of β.

A priori, we knew very little about the parameter α. However, we believed the parameter to be nonzero in each event. We chose a value for α (based on several plots of the three-parameter lognormal distribution—see Figure 4.2) to be the mean of the prior distributions for α and chose a large variance for each distribution. A priori, we had no reason to believe that the peakedness parameter in each event should be different from the peakedness parameter in another event. Thus, the prior distribution for α in each event has a mean of 20 and a variance of 100.

4.5 Results

The joint posterior distribution for all parameters in the model is intractable. Therefore, a MCMC approach was used to draw samples from the posterior distribution. Specifically, Gibbs sampling (Geman and Geman (1984)) was used to with nested Metropolis-Hastings algorithms(Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller (1953) and Hastings (1970)) to generate draws from the posterior distributions of each parameter. We used the `gibbsit` software (Raferty and Lewis (1996)) to check for convergence. After thinning the chains for each parameter, the `gibbsit` output indicated that our chains had converged.

Table 4.5 contains a summary of the posterior distributions for each parameter in each model. As expected, the posterior mean for β in the swim is greater than the posterior mean for β in the run, and the posterior mean for β in the run is greater than the posterior mean for β in the bike. Also, the posterior distributions for the other parameters yielded sensible results according to their parameter interpretations. For instance, the posterior mean for τ_1 in the swim component is 3.381. Exponentiating this result and subtracting the posterior mean for α gives a finishing time of 26 minutes. This finishing time is a reasonable finishing time for a swim distance of 30 minutes.

Figure 4.4 is a plot that checks the fit of the model in each triathlon component. As mentioned in Section 4.4, in a three-parameter lognormal distribution, $\ln(X + \alpha)$ is distributed normally with mean μ and variance σ^2. The model for triathlon finishing times models μ with equation (4.8). Figure 4.4, then, is a plot of the log of the sum of the posterior mean of α and the finishing times in each component. The line on the plot is equation (4.8) with the posterior means for each of the parameters—τ_1, τ_2, β. The figure shows that our model for the mean

Table 4.5: Posterior means, posterior standard deviations, and credible intervals for each parameter in the model for each triathlon component.

	Swim			
Parameter	Mean	Std. Dev.	2.5% LCL	97.5% UCL
τ_1	3.381	0.029	3.325	3.440
τ_2	4.463	0.019	4.424	4.501
β	0.561	0.028	0.508	0.616
σ^2	0.039	0.004	0.032	0.047
α	3.204	0.758	1.802	4.792
	Cycle			
τ_1	4.731	0.051	4.624	4.830
τ_2	6.083	0.013	6.056	6.108
β	0.008	0.0004	0.007	0.009
σ^2	0.011	0.001	0.009	0.014
α	46.231	5.451	35.322	57.077
	Run			
τ_1	3.833	0.035	3.772	3.906
τ_2	5.778	0.024	5.731	5.825
β	0.048	0.002	0.044	0.052
σ^2	0.046	0.004	0.038	0.054
α	3.256	1.375	1.030	6.327

seems reasonable in all three triathlon events.

However, Figure 4.5 provides another graphical model check. As mentioned in Section 4.4.1.2, the distribution of $\ln(X + \alpha)$ is normal in a three-parameter lognormal distribution. Thus Figure 4.5 contains normal probability plots for the residuals in our model. Because the $\ln(X+\alpha)$ is follows a normal distribution when X is distributed as a three-parameter lognormal random variable, the residuals are calculated as

$$\ln(X + \hat{\alpha}) - \hat{\mu}, \tag{4.10}$$

where $\hat{\alpha}$ is the posterior mean for α, $\hat{\mu} = \tau_1 + (\hat{\tau}_2 - \hat{\tau}_1)\frac{1-e^{-\hat{\beta}(d-\delta_1)}}{1-e^{-\hat{\beta}(\delta_2-\delta_1)}}$, and $\hat{\tau}_1$, $\hat{\tau}_2$, and $\hat{\beta}$ are the posterior means for τ_1, τ_2, and β. With the exception of a few outliers, the plots in Figure 4.5 show no major departures from normality.

4.6 The Fair Triathlon

The standard deviation of the three-parameter lognormal distribution is

$$\sqrt{e^{2\mu+\sigma^2}(e^{\sigma^2} - 1)}. \tag{4.11}$$

Notice that equation (4.11) is a function of the parameter μ. In the model for each triathlon component, the parameter μ is modeled as a function of distance. Thus, in the triathlon model, the standard deviation is a function of the distance. Substituting equation (4.8) for μ into equation (4.11) gives

$$\sqrt{e^{2\left[\tau_1+(\tau_2-\tau_1)\frac{1-e^{-\beta(d-\delta_1)}}{1-e^{-\beta(\delta_2-\delta_1)}}\right]+\sigma^2}(e^{\sigma^2} - 1)}. \tag{4.12}$$

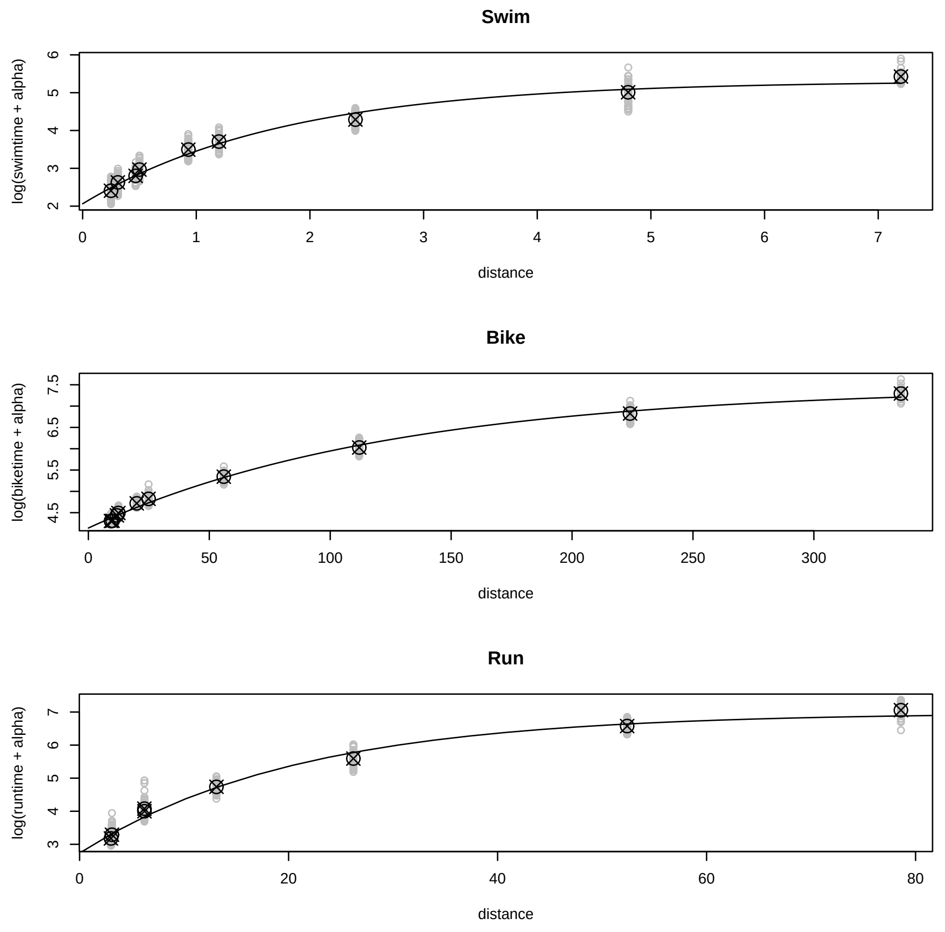

Figure 4.4: Plots of the log of the sum of the posterior mean of αplus the finishing time for each triathlon component. The line on the plot is equation (4.8) with the posterior means of τ_1, τ_2, and β.

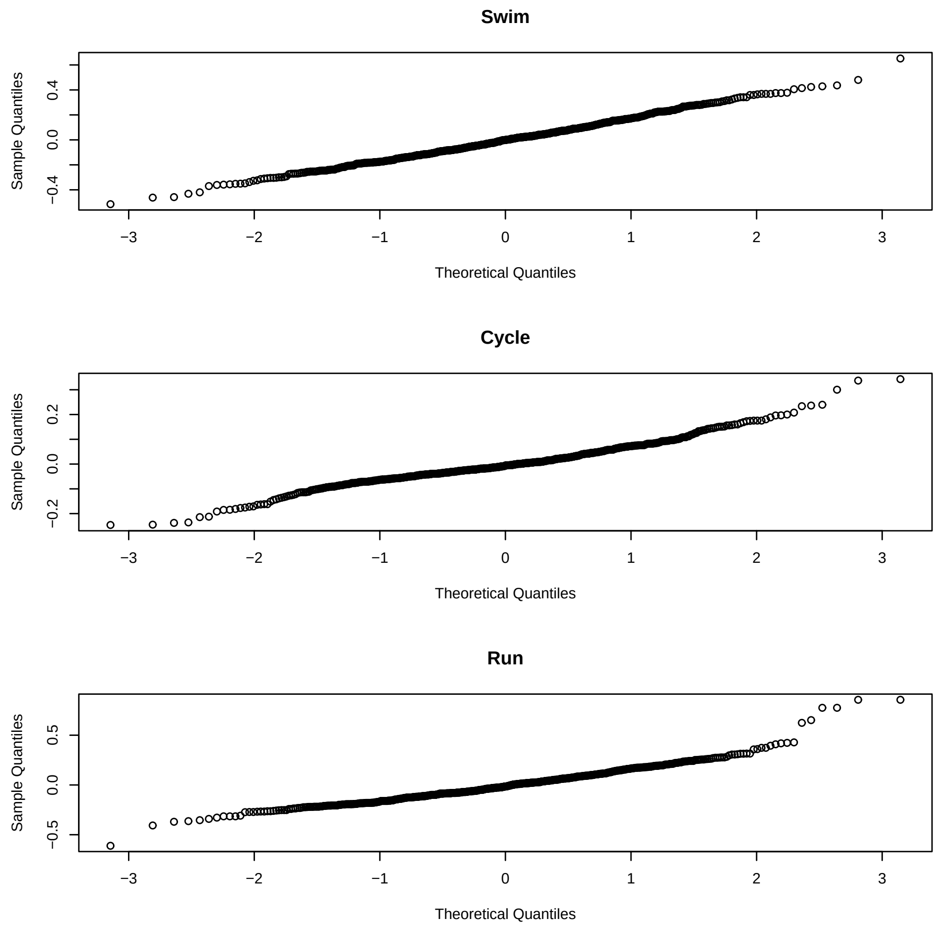

Figure 4.5: Normal probability plots of the residuals for each event. The residuals were calculated as the difference between the log of the sum of finishing time and the posterior mean for alpha and the predicted value for μ at a given distance.

Table 4.6: Table of fair triathlon distances (in miles). Each triathlon is based on a run distance of 10, 15, 20, or 25 kilometers. Also listed are the ratios of the bike distances to the swim distances and the run distances to the swim distances.

Base Run Distance	Swim	Bike	Run	Bike/Swim	Run/Swim
10k	1.5	18.0	6.2	11.8	4.1
15k	2.2	38.9	9.3	17.6	4.2
20k	2.9	46.6	12.4	19.3	4.0
25k	4.1	61.5	15.5	17.8	3.4

Thus equation (4.12) can be used along with parameter estimates (posterior means) to find the distances in each triathlon component that give the same standard deviation.

Table 4.6 contains several fair triathlons of different lengths. Each triathlon uses the run triathlon component as the base distance. In other words, the triathlons were computed by using a run distance to solve for the standard deviation. Then, the distances for the swim and cycle components were solved for that particular standard deviation. The run distances in Table 4.6 correspond to distances of 10, 15, 20, 25, and 30 kilometers.

The first triathlon in Table 4.6 uses the same running distance as the current Olympic Triathlon and, therefore, can be thought of as the fair Olympic triathlon. The swim distance is approximately 60% longer and the bike distance is about 28% shorter in the fair version of the Olympic triathlon than in the current Olympic triathlon.

If a distance of 13.1 miles (the current run distance for the Half Ironman triathlon)is used as the base run distance, then the swim and bike distances associated with a run distance of 13.1 miles are 3.4 and 64.8 miles respectively. As with the Olympic triathlon, the fair distance for the swim in a Half Ironman is 183% longer and the bike distance is about 16% longer than the current distances for the swim and the bike in the Half Ironman.

If a distance of 26.2 miles (the current run distance for an Ironman triathlon) is used as the base run distance, then the model calculates a swim distance much longer than the longest swim distance in the data set. This distance is extremely unreliable because of the lack of data at the longer distances. However, Table 4.6 also lists the ratios of the bike distances to the swim distances and the run distances to swim distances for the triathlons in the table. These ratios suggest an approximate rule of thumb for constructing the fair triathlon. This rule of thumb is the ratio 1:17:4 of swim to bike to run. If the rule of thumb is used, then a fair Ironman triathlon is a 6.6 mile swim, a 111.4 mile bike, and a 26.2 mile run. Once again, the swim distance is 175% longer than the swim distance in the current Ironman. However, the bike distance is the same.

The difficulty in calculating the fair Ironman underscores possible unreliability in constructing a fair triathlon with the current data. The only data on swimming distances longer than the 2.4 mile swim in the Ironman come from Double and Triple Ironman triathlons. For obvious reasons, not many triathletes compete in Double and Triple Ironman triathlons, which makes obtaining large amounts of data at these distances impossible. And those triathletes who do participate in Double and Triple triathlons cannot be considered "typical" triathletes by any means. Thus predictions for fair triathlons that require swim distances longer than the current distances in the data set are unreliable, and predictions for swim distances longer than 2.4 miles may be influenced by the types of triathletes that choose to compete in Double and Triple Ironman triathlons.

However, despite the lack of data at longer swim distances, the data clearly show that the swimming portion of the triathlon is underweighted. Our model gives a longer distance for the swim portion of the triathlon, which will clearly improve the fairness of the triathlon regardless of the problems with the data.

In summary, the major triathlons of today are severely imbalanced in the relative importance they place on each event. Strong swimmers are at a disadvantage, not because of any lack of athleticism, but because of the improper relative event distances of the major triathlons. We have presented a model for the finishing times in each event of a triathlon, and we have used this model to create the fair triathlon. Based on the results of our model, we suggest a simple ratio for constructing fair triathlons—1:17:4 for swim to bike to run. Only by using fair triathlons can we ever hope to answer John Collins' question, "Who are the better athletes?"

Chapter 5

Further Research and Contributions to Statistical Practice

This chapter discusses possible problems in the current model for triathlon finishing times and suggests possible areas for further research. The first section begins with a discussion of a hierarchical model for triathlon finishing times. The next section discusses possible research in the covariance structure of triathlon finishing times, and the last section discusses deficiencies in the data.

5.1 A Hierarchical Model

The model in Chapter 3 makes the implicit assumption of no differences among different triathlons of the same distance (other than chance variation). The parameter μ is modeled as a function of distance but is not allowed to vary from one triathlon to another. The parameters σ^2 and α are also not allowed to vary from one triathlon to another. Given that each triathlon is competed in a different part of the world and under different weather, water, and road conditions, the assumption that there are no differences among triathlons of the same distance seems tenuous.

A hierarchical model can easily model differences between triathlons. This model is listed as follows. Let y_{ij} be the finishing time in a triathlon component for the j^{th} triathlete in the i^{th} triathlon. Now let y_{ij} follow a three-parameter lognormal distribution with parameters θ_i, σ_i^2, and α_i. Thus, each of the parameters in the three-parameter lognormal distribution are allowed to vary from triathlon to triathlon. Now let σ_i^2 follow a gamma distribution with parameters γ_σ and ξ_σ and α_i follow a gamma distribution with parameters γ_α and ξ_α, where γ_{σ^2}, ξ_{σ^2}, γ_α, and ξ_α all follow independent gamma distributions. Let θ_i follow a normal distribution with mean μ and variance σ_μ^2, where $\mu = \tau_1 + (\tau_2 - \tau_1)\frac{1-e^{-\beta(d-\delta_1)}}{1-e^{-\beta(\delta_2-\delta_1)}}$. Finally, let τ_1, τ_2, and β follow the distributions specified in Chapter 3.

With this hierarchical model, each triathlon is allowed to have its own parameters θ and σ^2, and its own peakedness parameter α. This additional flexibility allows the model to account for differences in triathlons that are due to the different locations and/or different weather conditions, etc., of each triathlon.

5.2 Covariance Structure

Another implicit assumption in the model described in Chapter 3 is independence of triathlon components. Modeling each component separately implies that knowing the finishing time of a triathlete in one component of a triathlon gives no information about where the triathlete finished in another component of the event. This is not a very realistic assumption. A more realistic model should take into account that triathletes who finish with good times in one of the components are likely the finish with good times in the other components, and triathletes who finish with poor times are likely to finish with poor times in the other triathlon components.

5.3 Sparsity of Data

Another complication in modeling triathlon finishing times was already mentioned in Chapter 4—the lack of data at swimming distances longer than 2.4 miles. All triathlons in competition today have a swim distance that is too short for the triathlon to be fair. The only data available on triathlons with swim distances longer than 2.4 miles is from Double and Triple Ironman triathlons. Not surprisingly, very few triathletes participate in these events. This lack of data could make inference at longer swim distances unreliable.

However, if the triathlon community embraces the idea of a fair triathlon, more triathlons will be organized with relatively longer swim distances. When this happens, more data will be available at longer swim distances, which will make the estimates for longer fair triathlons (i.e. a triathlon with a swim distance longer than 2.4 miles) more reliable.

5.4 Contributions to Statistical Practice

One of the main focuses of statistical research is to produce novel ways to model data that are appropriate for a given data set. The model for triathlon finishing times uses two statistical techniques that are unique in the statistical literature. The first is the use of a nonlinear function to model the parameter μ in a lognormal distribution. This model was fit using a Bayesian paradigm that incorporated prior information about model parameters. The fair triathlon could be constructed because the nonlinear function for μ accurately captured trends over different distances seen in the data and because the standard deviation in a lognormal distribution is a function of μ.

The second technique is the idea of balancing the relative importance of related physical phenomena by equating their uncertainty. In the triathlon model, the physical phenomena are the individual events in a triathlon and the uncertainty is represented by the standard deviations in each event. The triathlon model provides a way to create the fair triathlon by equating the standard deviations in each triathlon component.

Appendix A

Data Sources

The internet sources for all triathlons in the data set are listed in Table A.1.

Table A.1: Internet sources for the triathlons in the dataset.

Triathlon	Internet Site
2001 Ironman World Championship	www.ironmanlive.com
1999 Blackwater Triathlon	www.coolrunning.com
1997 Boulderpeak Triathlon	www.boulderpeak.com
1998 Boulderpeak Triathlon	www.boulderpeak.com
1999 Boulderpeak Triathlon	www.boulderpeak.com
2001 Ironman Japan	www.ironmanlive.com
2002 Ironman Japan	www.ironmanlive.com
2003 Ironman Brazil	www.ironmanlive.com
2001 Ironman California	www.ironmanlive.com
2002 Half Ironman California	www.ironmanlive.com
1996 Ironman Hawaii	www.bigbowls.com
2001 Ironman Florida	www.ironmanflorida.com
1997 Keauhou Kona Half Ironman	www.keauhoutriathlon.com
1998 Keauhou Kona Half Ironman	www.keauhoutriathlon.com
1997 Keauhou Kona Olympic Triathlon	www.keauhoutriathlon.com
1998 Keauhou Kona Olympic Triathlon	www.keauhoutriathlon.com
1999 Keauhou Kona Olympic Triathlon	www.keauhoutriathlon.com
1999 Age Groups Triathlon	www.tricalifornia.com
1999 Ironman Part	www.geocities.com
2002 Ironman Lanzarote	www.ironmalive.com
1999 College Olympic Triathlon	www.tricalifornia.com
1999 High School Olympic Triathlon	www.tricalifornia.com
2002 Ironman New Zealand	www.ironmanlive.com
2001 Lake Placid Ironman	www.ironmanusa.com
1997 Memphis in May Triathlon	www.mimtri.org
1998 Memphis in May Triathlon	www.mimtri.org
1998 Sri Chinmoy Sprint Triathlon	www.sunsite.anu.edu.au
1999 Sri Chinmoy Sprint Triathlon	www.sunsite.anu.edu.au
2001 Sri Chinmoy Sprint Triathlon	www.sunsite.anu.edu.au
2003 Mansfield Short Course Triathlon	www.onestepbeyond.org.uk
1999 Tupper Lake Sprint Triathlon	www.tupperlakeinfo.com
2000 Tupper Lake Sprint Triathlon	www.tupperlakeinfo.com
2003 Troon Sprint Triathlon	www.ayrodynamic.uk.co
2002 Wed Dog Sprint Triathlon	www.usat-se.org
1999 Odyssey Double Iron Triathlon	www.angelfire.com
2000 Odyssey Double Iron Triathlon	www.angelfire.com
2001 Odyssey Double Iron Triathlon	www.angelfire.com
2002 Odyssey Double Iron Triathlon	www.angelfire.com
2003 Odyssey Double Iron Triathlon	www.angelfire.com
2000 Odyssey Triple Iron Triathlon	www.angelfire.com
2001 Odyssey Triple Iron Triathlon	www.angelfire.com
2002 Odyssey Triple Iron Triathlon	www.angelfire.com
2003 Odyssey Triple Iron Triathlon	www.angelfire.com
2001 Le Defi Mondial de l'Endurance	www.angelfire.com

Appendix B

Further Graphical Model Checks

This section contains several plots of the residuals that demonstrate the fit of the model. Figure B.1 is a plot of the histograms of the residuals for the swim, cycle, and run. The thick solid line on each plot is a normal density with mean equal to the mean of the residuals and variance equal to the variance of the residuals. The thin line on each plot is a density smooth of the residuals The plots show that for the swim and the run the normal distribution is a particularly good fit. Although the plot for the bike residuals does not fit a normal distribution as well as the plots for the swim and the run residuals, the density smooth for the bike is approximately symmetric and single peaked.

Figures B.2, B.3, B.4, B.5, B.6, and B.7 check the normality of the residuals at each of the unique distances in the data set for each of the triathlon components. Figures B.2 and B.3 show that the residuals are normally distributed for most distances in the swim event. However, there are some outliers at the 7.2 mile distance that might indicate a lack of normality at this distance. Figures B.4 and B.5 show no major departures from normality at any of the individual bike

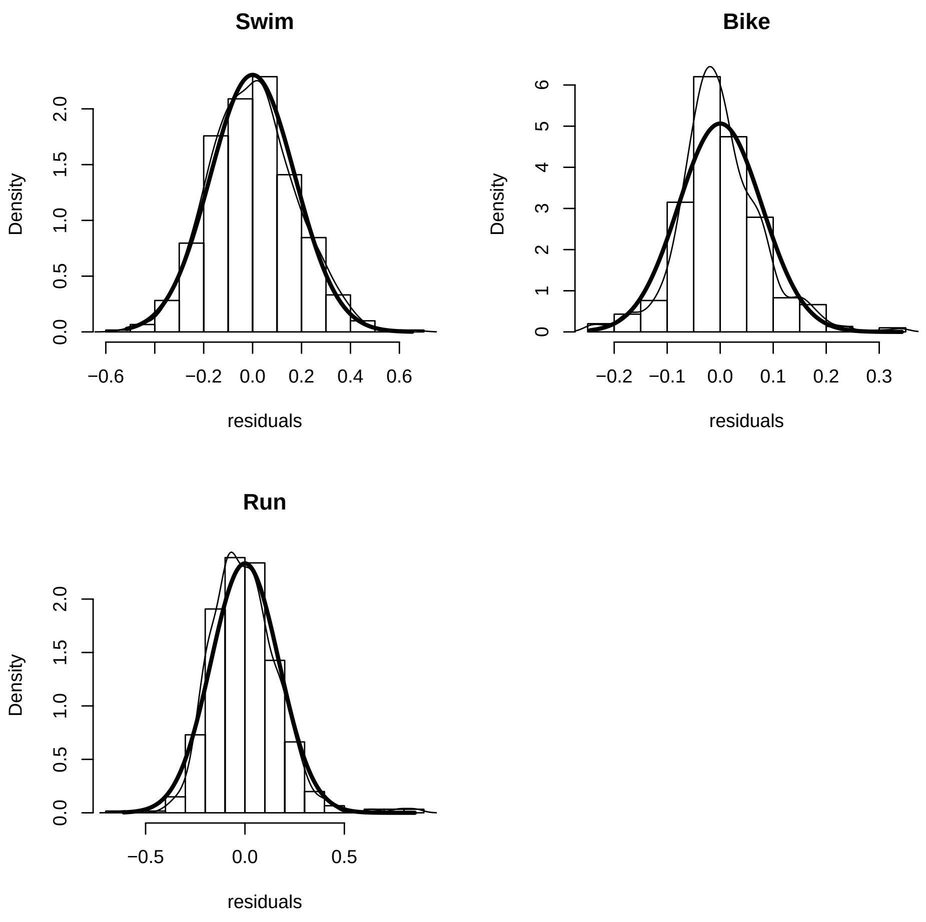

Figure B.1: Histograms of the residuals for the swim, cycle, and run. The thick solid line on each plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals. The thin line on each plot is a density smooth of the residuals.

distances. Figures B.6 and B.7 show that, at most distances, the run residuals are roughly normal. However, the plot for a distance of 6.2 miles contains four outliers which could indicate a lack of normality. The plot at a run distance of 13.1 miles shows a bimodal pattern, which could also indicate nonnormality.

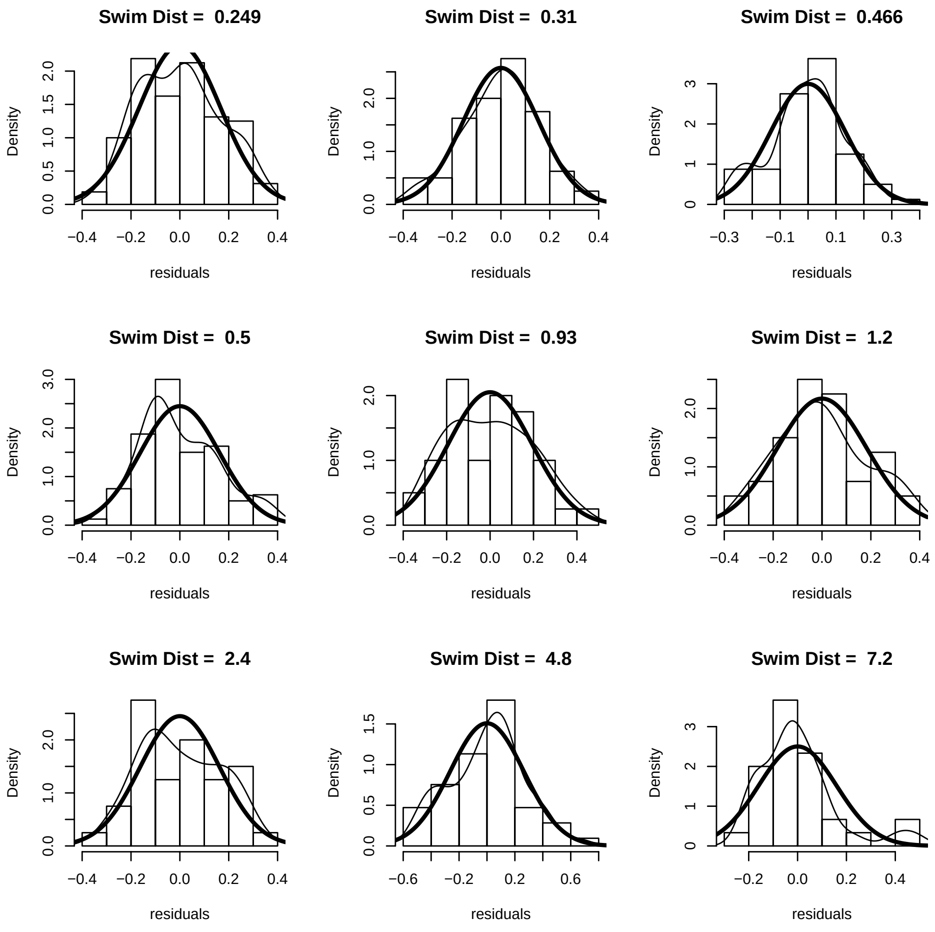

Figure B.2: Histograms of the residuals at each unique swim distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals at that distance. The thin line is a density smooth of the data.

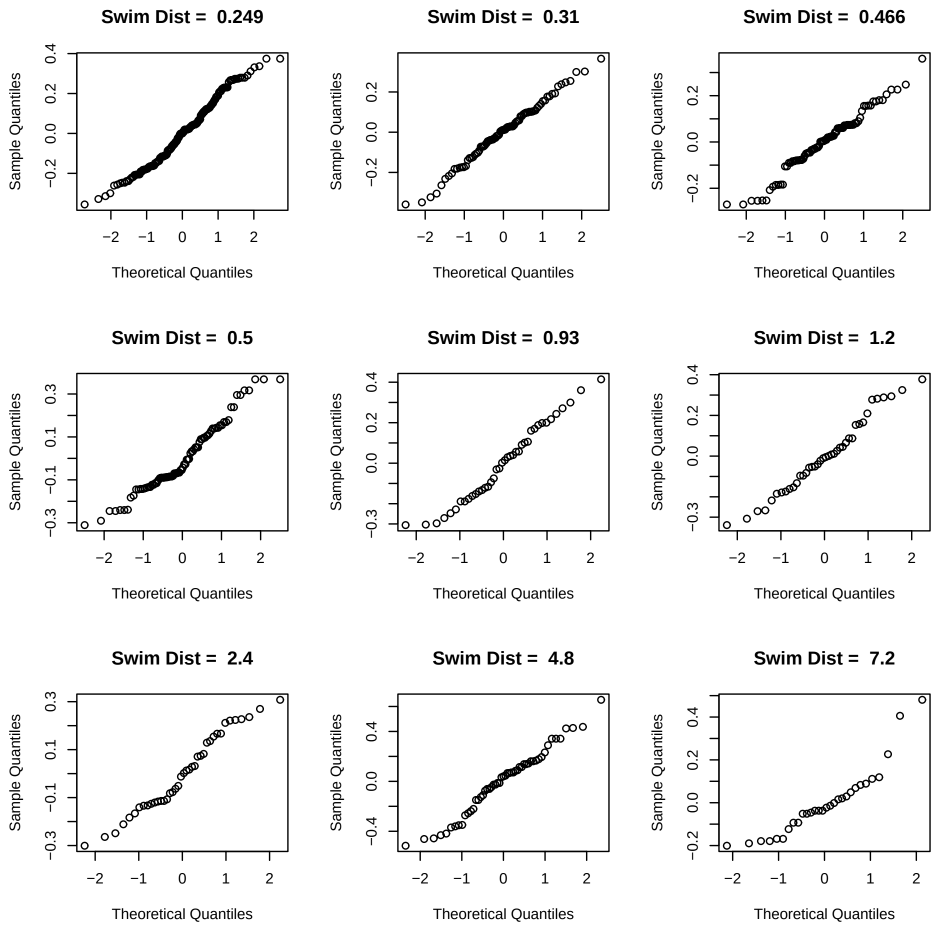

Figure B.3: Normal probability plots of the residuals at each unique swim distance in the data set.

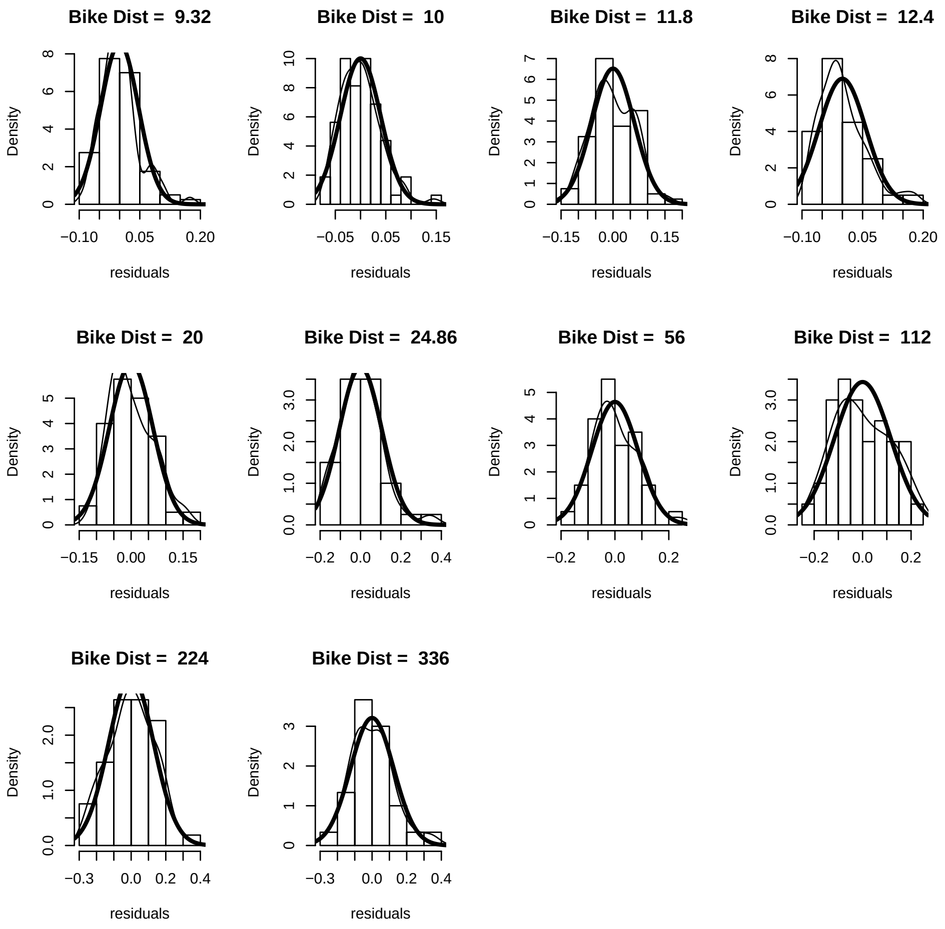

Figure B.4: Histograms of the residuals at each unique bike distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance of the residuals at that distance. The thin line is a density smooth of the data.

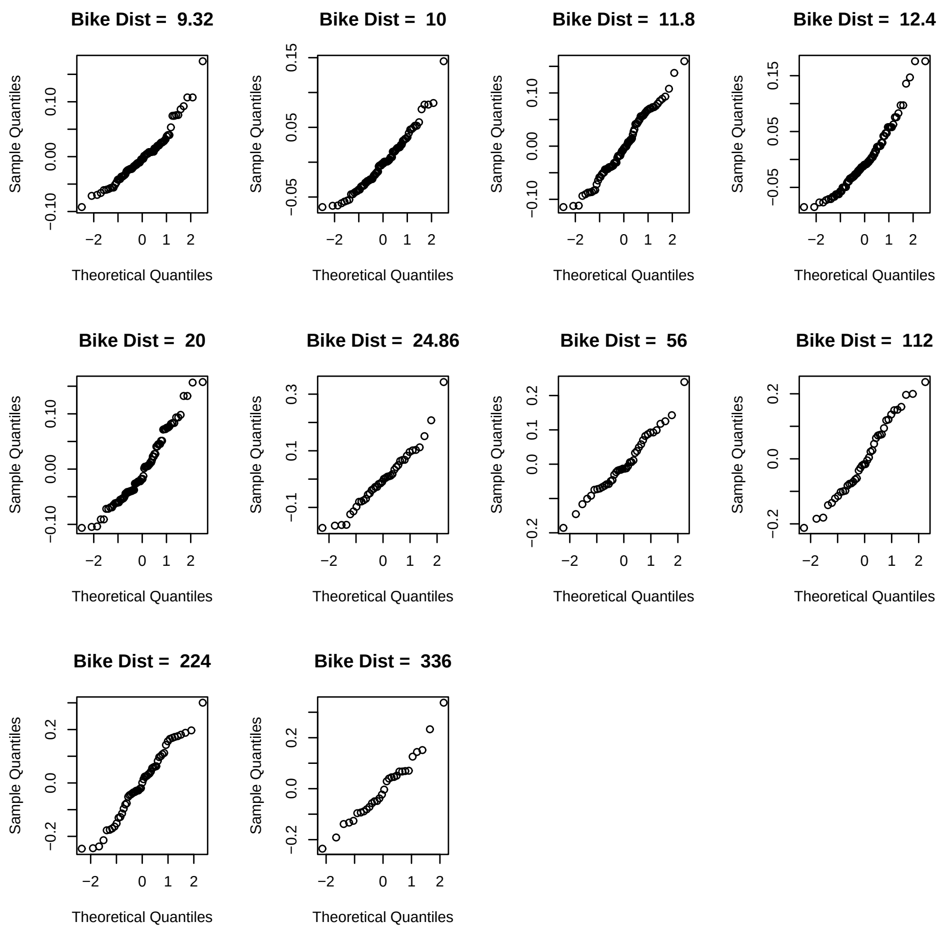

Figure B.5: Normal probability plots of the residuals at each unique bike distance in the data set.

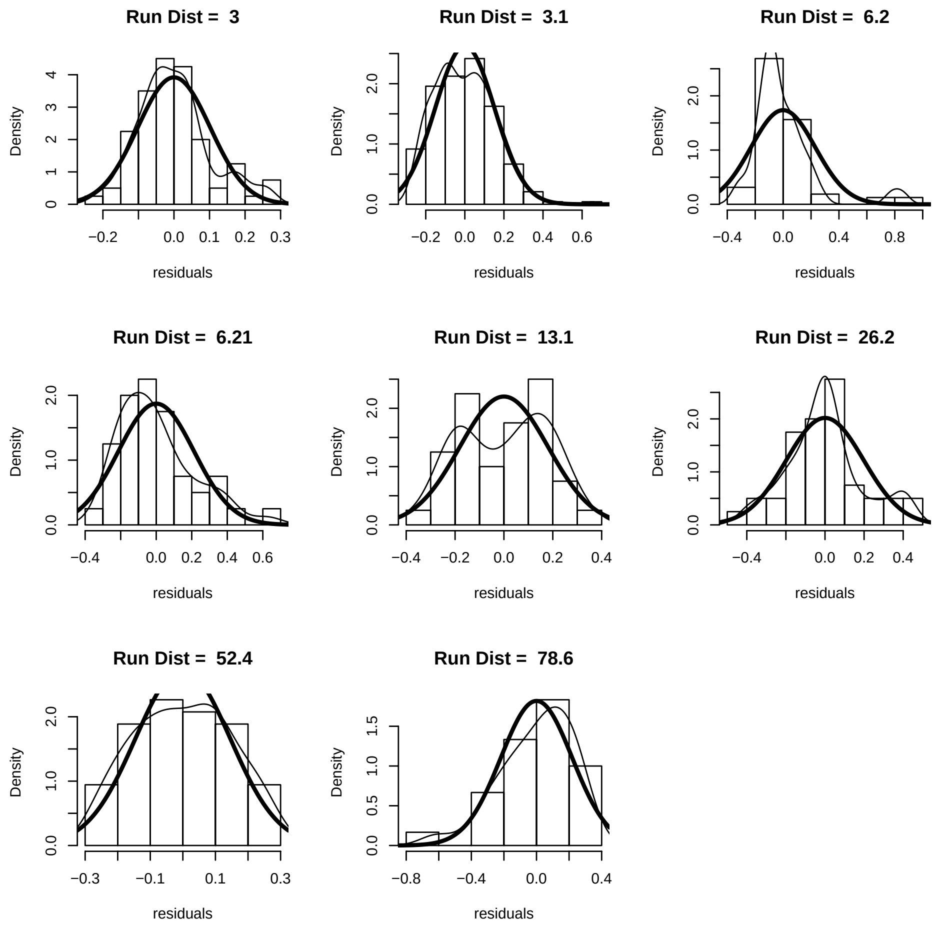

Figure B.6: Histograms of the residuals at each unique run distance in the data set. The thick line on the plot is a normal density with mean equal to the sample mean of the residuals and variance equal to the sample variance at that distance. The thin line is a density smooth of the data.

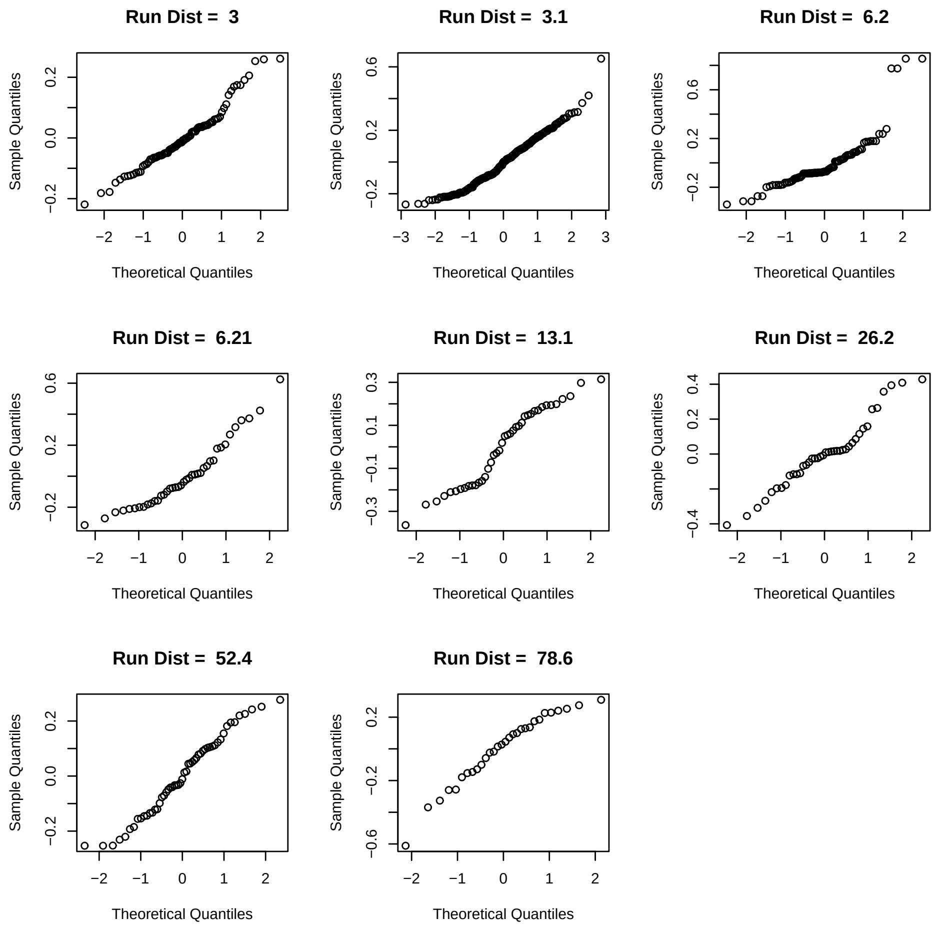

Figure B.7: Normal probability plots of the residuals at each unique run distance in the data set.

Appendix C

Code: Posterior Simulation

This section contains the Gibbs sampling C-code used to generate simulations from the posterior distributions of the parameters.

C.1 Swim Code

```
#include <stdio.h>
// For all standard print functions, file i/o, etc...
#include <string.h>
// For tokenizing and otherwise manipulating strings
#include <stdlib.h>
// For converting a string into a double precision number,
// and allocating memory
#include <fstream.h>
// For easier file i/o
#include <math.h>
// pow and exp functions
#include <gsl/gsl_rng.h>
// For random number generators
#include <gsl/gsl_randist.h>
// Random Gaussian Numbers
#include <time.h>


typedef struct data {
double swimtime;
double biketime;
double runtime;
double bdist;
double rdist;
double sdist;
} Data;

// Parameter values for priors
#define A_TAU1 1156.0
#define B_TAU1 340.0
#define A_TAU2 900.0
#define B_TAU2 200.0
#define A_ALPHA 4.0
#define B_ALPHA 0.2
#define A_SIG_SQ 9.0
#define B_SIG_SQ 300.0
#define M_BETA 0.06
#define S2_BETA 0.04

#define DELTA1 0.93
#define DELTA2 2.40

unsigned long REPS = 0;

double sumit(Data*,double,double,double,double);
double lccTau1(Data*,double,double,double,double,double);
double lccTau2(Data*,double,double,double,double,double);
double lccBeta(Data*,double,double,double,double,double);
double lccSigmaSq(Data*,double,double,double,double,double);
double lccAlpha(Data*,double,double,double,double,double);

double sigCandTau1 = 1.0,
sigCandTau2 = 1.0,
sigCandBeta = 1.0,
sigCandSigSq = 1.0,
sigCandAlpha = 1.0;

// Counts how many rows have been read in
int numData = 0;

int main(int argc, char **argv)
{
// Used for reading one token at a time.
char *token;

// Allocate enough memory to hold all the data
```

```
Data *triData = (Data*)malloc(sizeof(Data)*26106);

// Get filename from argument list given on commandline.
char *dataFilePath = argv[1];

// The second parameter will be the output file name.
char *outputFilename = argv[2];

// Create a buffer to read file
char buffer[1024];
char* filebuffer;

// Create an object that READS a file
ifstream fileD;
fileD.open(dataFilePath);
// fileD.open("f:/master's project/deleteme2.csv");
if(!fileD)
printf("Could not open file.");

//read in entire csv file:
// get length of file:
fileD.seekg (0, ios::end);
int length = fileD.tellg();
//set length greater by one to make last bytes NULL
length++;
fileD.seekg (0, ios::beg);

// allocate memory:
filebuffer = new char [length];

// make the whole filebuffer NULLs
memset(filebuffer, '\0', length);

// read data as a block:
fileD.read (filebuffer,length);

fileD.close();


/*********************************************/
/****** Read in the Candidate Sigma File ******/
/*********************************************/
```

```
// Read in candidate sigma file
ifstream candidateFile;
candidateFile.open("candidate.txt");

while(!candidateFile.eof())
{
candidateFile.getline(buffer, 1024, '\n');
token = strtok(buffer, "=");

if( (token != NULL) && (strcmp(token, "sigCandTau1") == 0) )
{
sigCandTau1 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandTau2") == 0) )
{
sigCandTau2 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandBeta") == 0) )
{
sigCandBeta = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandSigSq") == 0) )
{
sigCandSigSq = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandAlpha") == 0) )
{
sigCandAlpha = strtod( strtok(NULL, "="), NULL );
}


if( (token != NULL) && (strcmp(token, "REPS") == 0) )
{
REPS = strtol( strtok(NULL, "="), NULL, 10);
}
}

printf("Using the following values for candidate sigmas:\n");
printf("sigCandTau1: %f\n
sigCandTau2: %f\n
sigCandBeta: %f\n
sigCandAlpha: %f\n
sigCandSigSq: %f\nREPS: %ld\n\n",
sigCandTau1,
sigCandTau2,
sigCandBeta,
sigCandSigSq,
sigCandAlpha,
REPS);


/**********************************/
/****** Read in the Data File ******/
/**********************************/

//Print Header (7 tokens long)
token = strtok(filebuffer, ",");
printf("Stripping headers -----\n");
for(int cnt = 0; cnt < 6; cnt++)
{
token = strtok(NULL, ", \n");
printf("%s  ", token);
}

printf("\n");

token = strtok(NULL, ", \n");

while(token != NULL)
{
// swimtime
triData[numData].swimtime = strtod(token, NULL);

// biketime
token = strtok(NULL, ", \n");
triData[numData].biketime = strtod(token, NULL);

// runtime
token = strtok(NULL, ", \n");
triData[numData].runtime = strtod(token, NULL);

// sdist
```

```
token = strtok(NULL, ", \n");
triData[numData].sdist = strtod(token, NULL);

// bdist
token = strtok(NULL, ", \n");
triData[numData].bdist = strtod(token, NULL);

// rdist
token = strtok(NULL, ", \n");
triData[numData].rdist = strtod(token, NULL);

// throw away describe (ignore it)
token = strtok(NULL, ", \n");

// Increment the number of datalines read.
numData++;

//Get next token on next line
token = strtok(NULL, ", \n");

}

printf("Data read in: %ld\n",numData);

unsigned long i; // Loop counter

// Lookin’ at the data
printf("First ten observations:\n");
for(i = 0; i < 10; i++)
{
printf("%f , %f ,%f\n",
triData[i].swimtime,
triData[i].biketime,
triData[i].runtime);
}
int c = getchar();

printf("\nLast ten observations:\n");
int j;
for(j = (numData-10);j < numData; j++)
{
printf("%f , %f , %f\n",
triData[j].swimtime,
triData[j].biketime,
triData[j].runtime);
}
c = getchar();
// Done lookin' at the data

unsigned long startTime = time(NULL);

double *tau1;
tau1 = new double[REPS];
double *tau2;
tau2 = new double[REPS];
double *beta;
    beta = new double[REPS];
double *sigmaSq;
sigmaSq = new double[REPS];
double *alpha;
alpha = new double[REPS];

// Starting Values
tau1[0] = 3.524;
tau2[0] = 4.353;
beta[0] = 0.296;
sigmaSq[0] = 0.027;
alpha[0] = 10.429;

double candidate = 0.0;

// Initialize Random Number Generator
gsl_rng *runif = gsl_rng_alloc(gsl_rng_mt19937);

for(i = 1; i < REPS; i++)
{
if(!(i%250))
{ printf("[%ld of %ld] %f%% done. Time: %ld secs.\n",
i,
REPS,
(double)i/(double)REPS*100.0,
time(NULL) - startTime);
}

// Update TAU1
tau1[i] = tau1[i-1];
```

```
candidate = tau1[i-1] + gsl_ran_gaussian(runif,sigCandTau1);
if( candidate > 0.0 )
{
double temp =
lccTau1(  triData,
candidate,
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]) -
lccTau1(  triData,
tau1[i-1],
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
tau1[i] = candidate;
}
}

// Update TAU2
tau2[i] = tau2[i-1];
candidate = tau2[i-1] + gsl_ran_gaussian(runif,sigCandTau2);
if( candidate > 0.0 )
{
double temp =
lccTau2(  triData,
tau1[i],
candidate,
beta[i-1],
sigmaSq[i-1],
alpha[i-1]) -
lccTau2(  triData,
tau1[i],
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
```

```
{
tau2[i] = candidate;
}

}

// Update BETA
beta[i] = beta[i-1];
candidate = beta[i-1] + gsl_ran_gaussian(runif,sigCandBeta);
double temp =
lccBeta(  triData,
tau1[i],
tau2[i],
candidate,
sigmaSq[i-1],
alpha[i-1]) -
lccBeta(  triData,
tau1[i],
tau2[i],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
beta[i] = candidate;
}

// Update SIGMASQ
sigmaSq[i] = sigmaSq[i-1];
candidate = sigmaSq[i-1] + gsl_ran_gaussian(runif,sigCandSigSq);
if( candidate > 0.0 )
{
double temp =
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
candidate,
alpha[i-1]) -
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
sigmaSq[i] = candidate;
}
}

// Update ALPHA
alpha[i] = alpha[i-1];
candidate = alpha[i-1] + gsl_ran_gaussian(runif,sigCandAlpha);
if( candidate > 0.0 )
{
double temp =
lccAlpha( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i],
candidate) -
  lccAlpha( triData,
   tau1[i],
   tau2[i],
   beta[i],
   sigmaSq[i],
   alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
alpha[i] = candidate;
}
}
}

unsigned long endTime = time(NULL);

printf("** Elapsed time: %ld\n", endTime - startTime);

// Write out the parameters to a CSV (comma sep) file.  4 columns.
ofstream params;
params.open(outputFilename);
```

```
params << "Tau1,Tau2,Beta,SigmaSq,Alpha" << endl;

for(i = 0; i < REPS; i++)
{
params  << tau1[i] << ","
<< tau2[i] << ","
<< beta[i] << ","
<< sigmaSq[i] <<","
<< alpha[i] << endl;
}

params.close();
return 0;
}


double sumit(Data *list,
double tau1,
double tau2,
double beta,
double alpha)
{
double sum   = 0.0;
double inner = 0.0;
double multiplier, eBetaDelta1, eMBeta;
multiplier =  (tau2 - tau1) /
(1 - exp( -beta*(DELTA2 - DELTA1) ) );
eBetaDelta1 = exp( beta * DELTA1 );
eMBeta = exp(-beta);

for(int i = 0; i < numData; i++)
{
inner =
log(list[i].swimtime + alpha) -
(tau1 +
multiplier * ( 1.0 - eBetaDelta1 * pow(eMBeta, list[i].sdist) ) );
inner = inner*inner;

sum += inner;
}
return sum;
}
```

```
double lccTau1(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_TAU1 - 1) * log(tau1) -
tau1 * B_TAU1 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccTau2(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_TAU2 - 1) * log(tau2) -
tau2 * B_TAU2 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccBeta(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = -(beta - M_BETA) * (beta - M_BETA) / (2*S2_BETA) -
sumit(list, tau1, tau2, beta, alpha) /(2 * sigmaSq);

return result;
}

double lccSigmaSq(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_SIG_SQ - numData/2 - 1) * log(sigmaSq) -
sigmaSq * B_SIG_SQ -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccAlpha(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;
double sum = 0.0;
double temp = 0.0;
for(int i = 0; i < numData; i++)
{
temp = log( list[i].swimtime + alpha );
sum += temp;
}
result = -sum -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq) -
alpha * B_ALPHA + (A_ALPHA - 1) * log(alpha) ;
return result;
}
```

C.2 Bike Code

```
#include <stdio.h>
// For all standard print functions, file i/o, etc...
#include <string.h>
// For tokenizing and otherwise manipulating strings
#include <stdlib.h>
// For converting a string into a double precision number,
// and allocating memory
#include <fstream.h>
// For easier file i/o
#include <math.h>
// pow and exp functions
#include <gsl/gsl_rng.h>
// For random number generators
#include <gsl/gsl_randist.h>
// Random Gaussian Numbers
#include <time.h>


typedef struct data {
double swimtime;
double biketime;
double runtime;
double bdist;
double rdist;
double sdist;
} Data;

// Parameter values for priors
#define A_TAU1 529.0
#define B_TAU1 115.0
#define A_TAU2 1600.0
#define B_TAU2 266.7
#define A_ALPHA 4.0
#define B_ALPHA 0.2
#define A_SIG_SQ 1.69
#define B_SIG_SQ 130.0
#define M_BETA 0.02
#define S2_BETA 0.0001

#define DELTA1 24.86
#define DELTA2 112.0
```

```
unsigned long REPS = 0;

double sumit(Data*,double,double,double,double);
double lccTau1(Data*,double,double,double,double,double);
double lccTau2(Data*,double,double,double,double,double);
double lccBeta(Data*,double,double,double,double,double);
double lccSigmaSq(Data*,double,double,double,double,double);
double lccAlpha(Data*,double,double,double,double,double);

double sigCandTau1 = 1.0,
sigCandTau2 = 1.0,
sigCandBeta = 1.0,
sigCandSigSq = 1.0,
sigCandAlpha = 1.0;

// Counts how many rows have been read in
int numData = 0;

int main(int argc, char **argv)
{
// Used for reading one token at a time.
char *token;

// Allocate enough memory to hold all the data
Data *triData = (Data*)malloc(sizeof(Data)*26106);

// Get filename from argument list given on commandline.
char *dataFilePath = argv[1];

// The second parameter will be the output file name.
char *outputFilename = argv[2];

// Create a buffer for entire file.
char buffer[1024];
char* filebuffer;

// Create an object that READS a file
ifstream fileD;
fileD.open(dataFilePath);
if(!fileD)
printf("Could not open file.");

//read in entire csv file:
```

```
// get length of file:
fileD.seekg (0, ios::end);
int length = fileD.tellg();
//set length greater by one to make last bytes NULL
length++;
fileD.seekg (0, ios::beg);

// allocate memory:
filebuffer = new char [length];

// make the whole filebuffer NULLs
memset(filebuffer, '\0', length);

// read data as a block:
fileD.read (filebuffer,length);

fileD.close();


/*******************************************/
/****** Read in the Candidate Sigma File ******/
/*******************************************/

// Read in candidate sigma file
ifstream candidateFile;
candidateFile.open("candidate.txt");

while(!candidateFile.eof())
{
candidateFile.getline(buffer, 1024, '\n');
token = strtok(buffer, "=");

if( (token != NULL) && (strcmp(token, "sigCandTau1") == 0) )
{
sigCandTau1 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandTau2") == 0) )
{
sigCandTau2 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandBeta") == 0) )
{
sigCandBeta = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandSigSq") == 0) )
{
sigCandSigSq = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandAlpha") == 0) )
{
sigCandAlpha = strtod( strtok(NULL, "="), NULL );
}


if( (token != NULL) && (strcmp(token, "REPS") == 0) )
{
REPS = strtol( strtok(NULL, "="), NULL, 10);
}
}

printf("Using the following values for candidate sigmas:\n");
printf("sigCandTau1: %f\n
sigCandTau2: %f\n
sigCandBeta: %f\n
sigCandAlpha: %f\n
sigCandSigSq: %f\n
REPS: %ld\n\n",
sigCandTau1,
sigCandTau2,
sigCandBeta,
sigCandSigSq,
sigCandAlpha,
REPS);


/**********************************/
/****** Read in the Data File ******/
/**********************************/

//Print Header (7 tokens long)
token = strtok(filebuffer, ",");
printf("Stripping headers -----\n");
```

```
for(int cnt = 0; cnt < 6; cnt++)
{
token = strtok(NULL, ", \n");
printf("%s  ", token);
}

printf("\n");

token = strtok(NULL, ", \n");

while(token != NULL)
{
// swimtime
triData[numData].swimtime = strtod(token, NULL);

// biketime
token = strtok(NULL, ", \n");
triData[numData].biketime = strtod(token, NULL);

// runtime
token = strtok(NULL, ", \n");
triData[numData].runtime = strtod(token, NULL);

// sdist
token = strtok(NULL, ", \n");
triData[numData].sdist = strtod(token, NULL);

// bdist
token = strtok(NULL, ", \n");
triData[numData].bdist = strtod(token, NULL);

// rdist
token = strtok(NULL, ", \n");
triData[numData].rdist = strtod(token, NULL);

// throw away describe (ignore it)
token = strtok(NULL, ", \n");

// Increment the number of datalines read.
numData++;

//Get next token on next line
token = strtok(NULL, ", \n");

}

printf("Data read in: %ld\n",numData);

unsigned long i; // Loop counter

// Lookin' at the data
printf("First ten observations:\n");
for(i = 0; i < 10; i++)
{
printf("%f , %f , %f\n",
triData[i].swimtime,
triData[i].biketime,
triData[i].runtime);
}
int c = getchar();

printf("\nLast ten observations:\n");
int j;
for(j = (numData-10);j < numData; j++)
{
printf("%f , %f , %f\n",
triData[j].swimtime,
triData[j].biketime,
triData[j].runtime);
}
c = getchar();
// Done lookin' at the data

unsigned long startTime = time(NULL);

double *tau1;
tau1 = new double[REPS];
double *tau2;
tau2 = new double[REPS];
double *beta;
    beta = new double[REPS];
double *sigmaSq;
sigmaSq = new double[REPS];
double *alpha;
alpha = new double[REPS];
```

```
// Starting Values
tau1[0] = 4.472;
tau2[0] = 6.069;
beta[0] = 0.00979;
sigmaSq[0] = 0.0208;
alpha[0] = 22.105;

double candidate = 0.0;

// Initialize Random Number Generator
gsl_rng *runif = gsl_rng_alloc(gsl_rng_mt19937);


for(i = 1; i < REPS; i++)
{
if(!(i%250))
{ printf("[%ld of %ld] %f%% done. Time: %ld secs.\n",
i,
REPS,
(double)i/(double)REPS*100.0,
time(NULL) - startTime); }

// Update TAU1

tau1[i] = tau1[i-1];
candidate = tau1[i-1] + gsl_ran_gaussian(runif,sigCandTau1);
if( candidate > 0.0 )
{
double temp =
lccTau1(triData,
candidate,
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1])-
lccTau1(triData,
tau1[i-1],
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
```

```
if( log(u) < temp )
{
tau1[i] = candidate;
}
}

// Update TAU2
tau2[i] = tau2[i-1];
candidate = tau2[i-1] + gsl_ran_gaussian(runif,sigCandTau2);
if( candidate > 0.0 )
{
double temp =
lccTau2(triData,
tau1[i],
candidate,
beta[i-1],
sigmaSq[i-1],
alpha[i-1]) -
lccTau2(triData,
tau1[i],
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
tau2[i] = candidate;
}

}


// Update BETA

beta[i] = beta[i-1];
candidate = beta[i-1] + gsl_ran_gaussian(runif,sigCandBeta);

double temp =
lccBeta(triData,
tau1[i],
tau2[i],
candidate,
sigmaSq[i-1],
alpha[i-1]) -
lccBeta(triData,
tau1[i],
tau2[i],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
beta[i] = candidate;
}

// Update SIGMASQ
sigmaSq[i] = sigmaSq[i-1];
candidate = sigmaSq[i-1] + gsl_ran_gaussian(runif,sigCandSigSq);
if( candidate > 0.0 )
{
double temp =
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
candidate,
alpha[i-1]) -
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
sigmaSq[i] = candidate;
}
}

// Update ALPHA
alpha[i] = alpha[i-1];
candidate = alpha[i-1] + gsl_ran_gaussian(runif,sigCandAlpha);
if( candidate > 0.0 )
```

```
{
double temp =
lccAlpha( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i],
candidate) -
lccAlpha( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i],
alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
alpha[i] = candidate;
}
}
}

unsigned long endTime = time(NULL);

printf("** Elapsed time: %ld\n", endTime - startTime);

// Write out the parameters to a CSV (comma sep) file.  4 columns.
ofstream params;
params.open(outputFilename);

params << "Tau1,Tau2,Beta,SigmaSq,Alpha" << endl;

for(i = 0; i < REPS; i++)
{
params  << tau1[i] << ","
<< tau2[i] << ","
<< beta[i] << ","
<< sigmaSq[i] <<","
<< alpha[i] << endl;
}

params.close();
return 0;
```

```
}


double sumit( Data *list,
double tau1,
double tau2,
double beta,
double alpha)
{
double sum   = 0.0;
double inner = 0.0;
double multiplier, eBetaDelta1, eMBeta;
multiplier = (tau2 - tau1) / (1 - exp( -beta*(DELTA2 - DELTA1) ) );
eBetaDelta1 = exp( beta * DELTA1 );
eMBeta = exp(-beta);

for(int i = 0; i < numData; i++)
{

inner = log(list[i].biketime + alpha) -
(tau1 + multiplier * ( 1.0 -
eBetaDelta1 * pow(eMBeta, list[i].bdist) ) );
inner = inner*inner;

sum += inner;
}
return sum;
}

double lccTau1( Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result =
(A_TAU1 - 1) * log(tau1) - tau1 * B_TAU1 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
```

```
}

double lccTau2( Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_TAU2 - 1) * log(tau2) - tau2 * B_TAU2 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccBeta( Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = -(beta - M_BETA) * (beta - M_BETA) / (2*S2_BETA) -
sumit(list, tau1, tau2, beta, alpha) /(2 * sigmaSq);

return result;
}

double lccSigmaSq(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result =
(A_SIG_SQ - numData/2 - 1) * log(sigmaSq) - sigmaSq * B_SIG_SQ -
```

```
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccAlpha(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;
double sum = 0.0;
double temp = 0.0;
for(int i = 0; i < numData; i++)
{
temp = log( list[i].biketime + alpha );
sum += temp;
}
result =
-sum - sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq) -
alpha * B_ALPHA + (A_ALPHA - 1) * log(alpha) ;
return result;
}
```

C.3 Run Code

```
#include <stdio.h>
// For all standard print functions, file i/o, etc...
#include <string.h>
// For tokenizing and otherwise manipulating strings
#include <stdlib.h>
// For converting a string into a double precision number,
// and allocating memory
#include <fstream.h>
// For easier file i/o
#include <math.h>
// pow and exp functions
#include <gsl/gsl_rng.h>
// For random number generators
#include <gsl/gsl_randist.h>
// Random Gaussian Numbers
```

```
#include <time.h>

typedef struct data {
double swimtime;
double biketime;
double runtime;
double bdist;
double rdist;
double sdist;
} Data;


#define A_TAU1 1156.0
#define B_TAU1 340.0
#define A_TAU2 430.5625
#define B_TAU2 103.75
#define A_ALPHA 4.0
#define B_ALPHA 0.2
#define A_SIG_SQ 4.0
#define B_SIG_SQ 200.0
#define M_BETA 0.03
#define S2_BETA 0.000225

#define DELTA1 6.21
#define DELTA2 26.2

unsigned long REPS = 0;

double sumit(Data*,double,double,double,double);
double lccTau1(Data*,double,double,double,double,double);
double lccTau2(Data*,double,double,double,double,double);
double lccBeta(Data*,double,double,double,double,double);
double lccSigmaSq(Data*,double,double,double,double,double);
double lccAlpha(Data*,double,double,double,double,double);

double sigCandTau1 = 1.0, sigCandTau2 = 1.0,
sigCandBeta = 1.0, sigCandSigSq = 1.0, sigCandAlpha = 1.0;

// Counts how many rows have been read in
int numData = 0;

int main(int argc, char **argv)
{
```

```
// Used for reading one token at a time.
char *token;

// Allocate enough memory to hold all the data
Data *triData = (Data*)malloc(sizeof(Data)*26106);

// Get filename from argument list given on commandline.
char *dataFilePath = argv[1];

// The second parameter will be the output file name.
char *outputFilename = argv[2];

// Create a buffer to read entire file
char buffer[1024];
char* filebuffer;

// Create an object that READS a file
ifstream fileD;
fileD.open(dataFilePath);
if(!fileD)
printf("Could not open file.");

//read in entire csv file:
// get length of file:
fileD.seekg (0, ios::end);
int length = fileD.tellg();
//set length greater by one to make last bytes NULL
length++;
fileD.seekg (0, ios::beg);

// allocate memory:
filebuffer = new char [length];
// make the whole filebuffer NULLs
memset(filebuffer, '\0', length);

// read data as a block:
fileD.read (filebuffer,length);

fileD.close();

// Read in candidate sigma file
ifstream candidateFile;
candidateFile.open("candidate.txt");
```

```
while(!candidateFile.eof())
{
candidateFile.getline(buffer, 1024, '\n');
token = strtok(buffer, "=");

if( (token != NULL) && (strcmp(token, "sigCandTau1") == 0) )
{
sigCandTau1 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandTau2") == 0) )
{
sigCandTau2 = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandBeta") == 0) )
{
sigCandBeta = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandSigSq") == 0) )
{
sigCandSigSq = strtod( strtok(NULL, "="), NULL );
}

if( (token != NULL) && (strcmp(token, "sigCandAlpha") == 0) )
{
sigCandAlpha = strtod( strtok(NULL, "="), NULL );
}


if( (token != NULL) && (strcmp(token, "REPS") == 0) )
{
REPS = strtol( strtok(NULL, "="), NULL, 10);
}
}

printf("Using the following values for candidate sigmas:\n");
printf("sigCandTau1: %f\n
sigCandTau2: %f\n
sigCandBeta: %f\n
sigCandAlpha: %f\n
```

```
sigCandSigSq: %f\n
REPS: %ld\n\n",
sigCandTau1,
sigCandTau2,
sigCandBeta,
sigCandSigSq,
sigCandAlpha,
REPS);

//Print Header (7 tokens long)
token = strtok(filebuffer, ",");
printf("Stripping headers -----\n");

for(int cnt = 0; cnt < 6; cnt++)
{
token = strtok(NULL, ", \n");
printf("%s  ", token);
}

printf("\n");

token = strtok(NULL, ", \n");

while(token != NULL)
{

// swimtime
triData[numData].swimtime = strtod(token, NULL);

// biketime
token = strtok(NULL, ", \n");
triData[numData].biketime = strtod(token, NULL);

// runtime
token = strtok(NULL, ", \n");
triData[numData].runtime = strtod(token, NULL);

// sdist
token = strtok(NULL, ", \n");
triData[numData].sdist = strtod(token, NULL);

// bdist
token = strtok(NULL, ", \n");
```

```
triData[numData].bdist = strtod(token, NULL);

// rdist
token = strtok(NULL, ", \n");
triData[numData].rdist = strtod(token, NULL);

// throw away describe (ignore it)
token = strtok(NULL, ", \n");

// Increment the number of datalines read.
numData++;

//Get next token on next line
token = strtok(NULL, ", \n");
}

printf("Data read in: %ld\n",numData);

unsigned long i; // Loop counter

// Lookin' at the data
printf("First ten observations:\n");
for(i = 0; i < 10; i++)
{
printf("%f , %f , %f\n",
triData[i].swimtime,
triData[i].biketime,
triData[i].runtime);
}
int c = getchar();

printf("\nLast ten observations:\n");
int j;
for(j = (numData-10);j < numData; j++)
{
printf("%f , %f , %f\n",
triData[j].swimtime,
triData[j].biketime,
triData[j].runtime);
}
c = getchar();
// Done lookin' at the data
```

```
unsigned long startTime = time(NULL);

double *tau1;
tau1 = new double[REPS];
double *tau2;
tau2 = new double[REPS];
double *beta;
  beta = new double[REPS];
double *sigmaSq;
sigmaSq = new double[REPS];
double *alpha;
alpha = new double[REPS];

tau1[0] = 4.0;
tau2[0] = 5.0;
beta[0] = 0.068;
sigmaSq[0] = 0.036;
alpha[0] = 3.0;

double candidate = 0.0;

// Initialize Random Number Generator
gsl_rng *runif = gsl_rng_alloc(gsl_rng_mt19937);

for(i = 1; i < REPS; i++)
{
if(!(i%250))
{ printf("[%ld of %ld] %f%% done. Time: %ld secs.\n",
i,
REPS,
(double)i/(double)REPS*100.0,
time(NULL) - startTime); }


// Update TAU1

tau1[i] = tau1[i-1];
candidate = tau1[i-1] + gsl_ran_gaussian(runif,sigCandTau1);
if( candidate > 0.0 )
{
double temp =
lccTau1( triData,
candidate,
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]) -
lccTau1( triData,
tau1[i-1],
tau2[i-1],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
tau1[i] = candidate;
}
}


// Update TAU2

tau2[i] = tau2[i-1];
candidate = tau2[i-1] + gsl_ran_gaussian(runif,sigCandTau2);
if( candidate > 0.0 )
{
double temp =
lccTau2( triData,
tau1[i],
candidate,
beta[i-1],
sigmaSq[i-1],
alpha[i-1]) -
 lccTau2( triData,
  tau1[i],
  tau2[i-1],
  beta[i-1],
  sigmaSq[i-1],
  alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
tau2[i] = candidate;
}
}
```

```
// Update BETA

beta[i] = beta[i-1];
candidate = beta[i-1] + gsl_ran_gaussian(runif,sigCandBeta);

double temp =
lccBeta( triData,
tau1[i],
tau2[i],
candidate,
sigmaSq[i-1],
alpha[i-1]) -
lccBeta( triData,
tau1[i],
tau2[i],
beta[i-1],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);
if( log(u) < temp )
{
beta[i] = candidate;
}

// Update SIGMASQ

sigmaSq[i] = sigmaSq[i-1];
candidate = sigmaSq[i-1] + gsl_ran_gaussian(runif,sigCandSigSq);
if( candidate > 0.0 )
{
double temp =
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
candidate,
alpha[i-1]) -
lccSigmaSq( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i-1],
alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
sigmaSq[i] = candidate;
}
}

// Update ALPHA

alpha[i] = alpha[i-1];
candidate = alpha[i-1] + gsl_ran_gaussian(runif,sigCandAlpha);
if( candidate > 0.0 )
{
double temp =
lccAlpha( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i],
candidate) -
lccAlpha( triData,
tau1[i],
tau2[i],
beta[i],
sigmaSq[i],
alpha[i-1]);
double u = gsl_rng_uniform(runif);// = runiform
if( log(u) < temp )
{
alpha[i] = candidate;
}
}
}

unsigned long endTime = time(NULL);

printf("** Elapsed time: %ld\n", endTime - startTime);

// Write out the parameters to a CSV (comma sep) file.  4 columns.
ofstream params;
params.open(outputFilename);
```

```
params << "Tau1,Tau2,Beta,SigmaSq,Alpha" << endl;

for(i = 0; i < REPS; i++)
{
params  << tau1[i] << ","
<< tau2[i] << ","
<< beta[i] << ","
<< sigmaSq[i] <<","
<< alpha[i] << endl;
}
params.close();
return 0;
}


double sumit(Data *list,
double tau1,
double tau2,
double beta,
double alpha)
{
double sum   = 0.0;
double inner = 0.0;
double multiplier, eBetaDelta1, eMBeta;
multiplier = (tau2 - tau1) /
(1 - exp( -beta*(DELTA2 - DELTA1) ) );
eBetaDelta1 = exp( beta * DELTA1 );
eMBeta = exp(-beta);

for(int i = 0; i < numData; i++)
{

inner = log(list[i].runtime + alpha) -
(tau1 + multiplier * ( 1.0 -
eBetaDelta1 * pow(eMBeta, list[i].rdist) ) );
inner = inner*inner;

sum += inner;
}
return sum;
}

double lccTau1(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_TAU1 - 1) * log(tau1) - tau1 * B_TAU1 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccTau2(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = (A_TAU2 - 1) * log(tau2) - tau2 * B_TAU2 -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccBeta(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result = -(beta - M_BETA) * (beta - M_BETA) / (2*S2_BETA) -
sumit(list, tau1, tau2, beta, alpha) /(2 * sigmaSq);

return result;
}
```

```
double lccSigmaSq(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;

result =
(A_SIG_SQ - numData/2 - 1) * log(sigmaSq) - sigmaSq * B_SIG_SQ -
sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq);

return result;
}

double lccAlpha(Data *list,
double tau1,
double tau2,
double beta,
double sigmaSq,
double alpha)
{
double result = 0.0;
double sum = 0.0;
double temp = 0.0;
for(int i = 0; i < numData; i++)
{
temp = log( list[i].runtime + alpha );
sum += temp;
}
result =
-sum - sumit(list, tau1, tau2, beta, alpha) / (2.0 * sigmaSq) -
alpha * B_ALPHA + (A_ALPHA - 1) * log(alpha) ;
return result;
}
```

Appendix D

Code: Model Summaries

This section contains the R Code used to make many of the tables and graphs in this document.

D.1 Swim Code

This section contains R Code for mixing plots, posterior distributions, `gibbsit` analysis, and model checking for the swim triathlon component.

```
rm(list=ls())
nonlin <-
function(x,tau1,tau2,beta,delta1=0.93,delta2=2.4)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

##########################
#### Read in Raw Data ####
##########################
male.data <-
read.table("f:/master’s project/bigmale1.csv", sep=",", header=T)
```

```
#male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)
n <- nrow(male.data)

#######################################
#### Read in Posterior Simulations ####
#######################################
swim.output <-
read.table(file="c:/temp2/sout.txt",sep=",",header=T)
attach(swim.output)

##########################################
####  Read in Candidate Sigma Values  ####
##########################################
cand.sigmas <- read.table(file="f:/ppswim/candidate.txt",sep="=")
total <- cand.sigmas[nrow(cand.sigmas),ncol(cand.sigmas)]
cand.sigmas <- cand.sigmas[1:ncol(swim.output),2]
start <- 10000
rubble <- (start+1):total


####################################
#### Compute "Stay" Percentages ####
####################################
count <- numeric(ncol(swim.output))

for(j in 1:ncol(swim.output))
{
temp1 <- swim.output[(start+1):total,j]
temp2 <- swim.output[start:(total-1),j]
stay <- temp1-temp2
stay <- as.numeric(stay != 0)
count[j] <- sum(stay)
}
pct.stay <- round(count/(total-start),d=3)*100


########################
####  Mixing Plots  ####
########################
windows()
par(mfrow=c(3,2))
for(i in 1:ncol(swim.output))
```

```
{
plot(swim.output[rubble,i],type="l",xlab="")
title(main=paste(names(swim.output)[i]," (Swim)"),
sub=
paste("Stay%= ",pct.stay[i]," | Cand.Sig = ",cand.sigmas[i]))
}


#############################
#### Just the Posteriors ####
#############################
tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
sig2.ave  <- mean(SigmaSq)
post.means <- c(tau1.ave,tau2.ave,beta.ave,sig2.ave,alpha.ave)
post.sd <- c(sd(Tau1),sd(Tau2),sd(Beta),sd(SigmaSq),sd(Alpha))
windows()
par(mfrow=c(3,2))
for ( i in 1:length(post.means) )
{
plot(density(swim.output[rubble,i]),
type="l",
lwd=3,
main="",
ylab="density",
xlab="")
title( main=paste(names(swim.output)[i]," (Swim)"),
sub=paste("Mean= ", round(post.means[i],d=3),
" | Std. Dev= ", round(post.sd[i],d=3)))
}


#############################################
## Line Plot Using Means of the Parameters ##
#############################################

logstimea <- log(male.data$swimtime+alpha.ave)
swim.dists <- sort(unique(male.data$sdist))
mean.logstimesa <-
tapply(logstimea,as.factor(male.data$sdist),mean)
```

```
windows()
par(mfrow=c(1,1))
xx <- seq(0,max(male.data$sdist),len=100)
plot(male.data$sdist, logstimea,
col="gray",
ylab="log(swimtime + alpha)",
xlab="distance")
points(swim.dists, mean.logstimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Swim")


#######################
####  Model Check  ####
#######################

fits <- mean.logstimesa[match(male.data$sdist,swim.dists)]
resids <- logstimea - fits
par(mfrow=c(3,3))
for( i in 1:length(swim.dists) )
{
qqnorm(resids[male.data$sdist==swim.dists[i]],
main=paste("Distance = ",swim.dists[i]))
}


#######################
#### Gibbsit Stuff ####
#######################

for ( i in c(2,5,10,20,30) )
{

thin <- seq(60000,140000, by=i)
print(gibbsit(swim.output[thin,]))
}
```

D.2 Bike Code

This section contains R Code for mixing plots, posterior distributions, `gibbsit` analysis, and model checking for the bike triathlon component.

```
rm(list=ls())
nonlin <-
function(x,tau1,tau2,beta,delta1=24.86,delta2=112.0)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

##########################
#### Read in Raw Data ####
##########################
#male.data <-
read.table("f:/master's project/bigmale1.csv", sep=",", header=T)
male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)
n <- nrow(male.data)

#######################################
#### Read in Posterior Simulations ####
#######################################
bike.output <-
read.table(file="c:/temp2/bout.txt",sep=",",header=T)
attach(bike.output)

##########################################
####  Read in Candidate Sigma Values  ####
##########################################
cand.sigmas <-
read.table(file="f:/ppbike/candidate.txt",sep="=")
total <- cand.sigmas[nrow(cand.sigmas),ncol(cand.sigmas)]
cand.sigmas <- cand.sigmas[1:ncol(bike.output),2]
start <- 50000
rubble <- (start+1):total


####################################
#### Compute "Stay" Percentages ####
####################################
```

```
count <- numeric(ncol(bike.output))

for(j in 1:ncol(bike.output))
{
temp1 <- bike.output[(start+1):total,j]
temp2 <- bike.output[start:(total-1),j]
stay <- temp1-temp2
stay <- as.numeric(stay == 0)
count[j] <- sum(stay)
}
pct.stay <- round(count/(total-start),d=3)*100


########################
####  Mixing Plots  ####
########################
windows()
par(mfrow=c(3,2))
for(i in 1:ncol(bike.output))
{
plot(bike.output[rubble,i],type="l",xlab="")
title(main=paste(names(bike.output)[i], " (Bike)"),
sub=paste("Stay%= ",pct.stay[i]," | Cand.Sig = ",cand.sigmas[i]))
}


#############################
#### Just the Posteriors ####
#############################
tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
sig2.ave  <- mean(SigmaSq)
post.means <- c(tau1.ave,tau2.ave,beta.ave,sig2.ave,alpha.ave)
post.sd <- c(sd(Tau1),sd(Tau2),sd(Beta),sd(SigmaSq),sd(Alpha))
windows()
par(mfrow=c(3,2))
for ( i in 1:length(post.means) )
{
plot(density(bike.output[rubble,i]),
type="l",
lwd=3,
main="",
ylab="density",
xlab="")
title( main=paste(names(bike.output)[i], " (Bike)"),
sub=paste("Mean= ", round(post.means[i],d=3),
" | Std. Dev= ", round(post.sd[i],d=3)))
}


#############################################
## Line Plot Using Means of the Parameters ##
#############################################

logbtimea <- log(male.data$biketime+alpha.ave)
bike.dists <- sort(unique(male.data$bdist))
mean.logbtimesa <-
tapply(logbtimea,as.factor(male.data$bdist),mean)

windows()
par(mfrow=c(1,1))
xx <- seq(0,max(male.data$bdist),len=100)
plot(male.data$bdist, logbtimea,
col="gray",
ylab="log(biketime + alpha)",
xlab="distance")
points(bike.dists, mean.logbtimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Bike")


#######################
####  Model Check  ####
#######################

fits <- mean.logbtimesa[match(male.data$bdist,bike.dists)]
resids <- logbtimea - fits
par(mfrow=c(3,4))
for( i in 1:length(bike.dists) )
{
qqnorm(resids[male.data$bdist==bike.dists[i]],
main=paste("Distance = ",bike.dists[i]))
}
```

```
#######################
#### Gibbsit Stuff ####
#######################

for ( i in c(2,5,10,20,30) )
{

thin <- seq(60000,140000, by=i)
print(gibbsit(bike.output[thin,]))
}
```

D.3 Run Code

This section contains R Code for mixing plots, posterior distributions, `gibbsit` analysis, and model checking for the run triathlon component.

```
rm(list=ls())
nonlin <-
function(x,tau1,tau2,beta,delta1=6.21,delta2=26.2)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

##########################
#### Read in Raw Data ####
##########################
#male.data <-
read.table("f:/master's project/bigmale1.csv", sep=",", header=T)
male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)
n <- nrow(male.data)

#######################################
#### Read in Posterior Simulations ####
```

```
#######################################
run.output <-
read.table(file="c:/temp2/rout.txt",sep=",",header=T)
attach(run.output)

##########################################
####  Read in Candidate Sigma Values  ####
##########################################
cand.sigmas <- read.table(file="f:/pprun/candidate.txt",sep="=")
total <- cand.sigmas[nrow(cand.sigmas),ncol(cand.sigmas)]
cand.sigmas <- cand.sigmas[1:ncol(run.output),2]
start <- 10000
rubble <- (start+1):total


####################################
#### Compute "Stay" Percentages ####
####################################
count <- numeric(ncol(run.output))

for(j in 1:ncol(run.output))
{
temp1 <- run.output[(start+1):total,j]
temp2 <- run.output[start:(total-1),j]
stay <- temp1-temp2
stay <- as.numeric(stay != 0)
count[j] <- sum(stay)
}
pct.stay <- round(count/(total-start),d=3)*100


########################
####  Mixing Plots  ####
########################
windows()
par(mfrow=c(3,2))
for(i in 1:ncol(run.output))
{
plot(run.output[rubble,i],type="l",xlab="")
title(main=paste(main=names(run.output)[i]," (Run)"),
sub=paste("Stay%= ",pct.stay[i]," | Cand.Sig = ",cand.sigmas[i]))
}
```

```
#############################
#### Just the Posteriors ####
#############################
tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
sig2.ave  <- mean(SigmaSq)
post.means <- c(tau1.ave,tau2.ave,beta.ave,sig2.ave,alpha.ave)
post.sd <- c(sd(Tau1),sd(Tau2),sd(Beta),sd(SigmaSq),sd(Alpha))
windows()
par(mfrow=c(3,2))
for ( i in 1:length(post.means) )
{
plot(density(run.output[rubble,i]),
type="l",
lwd=3,
main="",
ylab="density",
xlab="")
title( paste(main=names(run.output)[i]," (Run)"),
sub=paste("Mean= ", round(post.means[i],d=3),
" | Std. Dev= ", round(post.sd[i],d=3)))
}


##############################################
## Line Plot Using Means of the Parameters ##
##############################################

logrtimea <- log(male.data$runtime+alpha.ave)
run.dists <- sort(unique(male.data$rdist))
mean.logrtimesa <-
tapply(logrtimea,as.factor(male.data$rdist),mean)

windows()
par(mfrow=c(1,1))
xx <- seq(0,max(male.data$rdist))
plot(male.data$rdist, logrtimea,
col="gray",
ylab="log(runtime + alpha)",
xlab="distance")
```

```
points(run.dists, mean.logrtimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Run")


#######################
####  Model Check  ####
#######################

fits <- mean.logrtimesa[match(male.data$rdist,run.dists)]
resids <- logrtimea - fits
par(mfrow=c(3,3))
for( i in 1:length(run.dists) )
{
qqnorm(resids[male.data$rdist==run.dists[i]],
main=paste("Distance = ",run.dists[i]))
}



####################################
#### Plot of Variance(distance) ####
####################################

xx <- seq(0,3*26.2,len=100)
vars <-
pplnvar(nonlin(xx, tau1.ave, tau2.ave, beta.ave), sig2.ave)
sds <- sqrt(vars)
windows()
plot(xx,sds,type="l",lwd=3)
title(main="Run")



#######################
#### Gibbsit Stuff ####
#######################

for ( i in c(2,5,10,20,30) )
{
thin <- seq(60000,140000, by=i)
print(gibbsit(run.output[thin,]))
}
```

D.4 Posterior Summaries

This section contains R code that generated the posterior summaries and plots found in Chapter 4.

```
######################
##### Swim Stuff #####
######################

nonlin <- function(x,tau1,tau2,beta,delta1=0.93,delta2=2.4)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

## Read in Raw Data ##
male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)

## Read in Simulations ##
output <-
read.table("c:/temp2/sout.txt", sep=",", header=T)
attach(output)
num.sum <- 2
rubble <- 10000:nrow(output)
swim.stats <-
matrix(numeric(ncol(output)*num.sum), ncol=num.sum)
for( i in 1:ncol(output) )
{
swim.stats[i,1]<- mean(output[rubble,i])
swim.stats[i,2]<- sd(output[rubble,i])
}
swim.stats <-
data.frame(mean=swim.stats[,1], std.dev=swim.stats[,2])

swim.cr <-  t(apply(output,2,quantile,prob=c(.025,.975)))
```

```
tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
swim.dists <- sort(unique(male.data$sdist))
logtimea <- log(male.data$swimtime+alpha.ave)
mean.logtimesa <- numeric(length(swim.dists))
for(i in 1:length(swim.dists))
{
mean.logtimesa[i] <-
mean(logtimea[male.data$sdist==swim.dists[i]], na.rm=TRUE)
}
windows()
par(mfrow=c(3,1))
xx <- seq(0,max(male.data$sdist),len=100)
plot(male.data$sdist, logtimea,
col="gray",
ylab="log(swimtime + alpha)",
xlab="distance")
points(swim.dists, mean.logtimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Swim")


######################
##### Bike Stuff #####
######################

nonlin <- function(x,tau1,tau2,beta,delta1=24.86,delta2=112.0)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

## Read in the Raw Data ##
male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)
```

```
## Read in the Simulations ##
output <- read.table("c:/temp2/bout.txt", sep=",", header=T)
attach(output)
num.sum <- 2
rubble <- 10000:nrow(output)
bike.stats <- matrix(numeric(ncol(output)*num.sum), ncol=num.sum)
for( i in 1:ncol(output) ){
bike.stats[i,1]<- mean(output[rubble,i])
bike.stats[i,2]<- sd(output[rubble,i])
}
bike.stats <-
data.frame(mean=bike.stats[,1], std.dev=bike.stats[,2])

bike.cr <-  t(apply(output,2,quantile,prob=c(.025,.975)))


tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
bike.dists <- sort(unique(male.data$bdist))
logtimea <- log(male.data$biketime+alpha.ave)
mean.logtimesa <- numeric(length(bike.dists))
for(i in 1:length(bike.dists))
{
mean.logtimesa[i] <-
mean(logtimea[male.data$bdist==bike.dists[i]], na.rm=TRUE)
}
xx <- seq(0,max(male.data$bdist),len=100)
plot(male.data$bdist, logtimea,
col="gray",
ylab="log(biketime + alpha)",
xlab="distance")
points(bike.dists, mean.logtimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Bike")


#####################
##### Run Stuff #####
#####################

nonlin <- function(x,tau1,tau2,beta,delta1=6.21,delta2=26.2)
```

```
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}

## Read in the Raw Data ##
male.data <-
read.table("f:/master's project/rmale.csv", sep=",", header=T)

## Read in the Simulations ##
output <- read.table("c:/temp2/rout.txt", sep=",", header=T)
attach(output)

num.sum <- 2
rubble <- 10000:nrow(output)
run.stats <-
matrix(numeric(ncol(output)*num.sum), ncol=num.sum)
for( i in 1:ncol(output) )
{
run.stats[i,1]<- mean(output[rubble,i])
run.stats[i,2]<- sd(output[rubble,i])
}
run.stats <-
data.frame(mean=run.stats[,1], std.dev=run.stats[,2])

run.cr <-  t(apply(output,2,quantile,prob=c(.025,.975)))


tau1.ave <- mean(Tau1)
tau2.ave <- mean(Tau2)
beta.ave <- mean(Beta)
alpha.ave <- mean(Alpha)
run.dists <- sort(unique(male.data$rdist))
logtimea <- log(male.data$runtime+alpha.ave)
mean.logtimesa <- numeric(length(run.dists))
for(i in 1:length(run.dists))
{
mean.logtimesa[i] <-
mean(logtimea[male.data$rdist==run.dists[i]], na.rm=TRUE)
}
xx <- seq(0,max(male.data$bdist),len=100)
plot(male.data$rdist, logtimea,
col="gray",
ylab="log(runtime + alpha)",
xlab="distance")
points(run.dists, mean.logtimesa, pch=13, cex=2)
lines(xx, nonlin(xx,tau1.ave, tau2.ave, beta.ave))
title(main="Run")
dev.copy2eps(file="f:/master’s project/project/mdlchk.eps")
dev.copy2eps(file="f:/master’s project/paper/mdlchk.eps")
summary.stats <- data.frame(swim.stats, bike.stats, run.stats)
summary.stats

cred.int <- data.frame(swim.cr,bike.cr,run.cr)

dev.copy2eps(file="f:/master’s project/project/mdlchk.eps")
dev.copy2eps(file="f:/master’s project/paper/mdlchk.eps")
```

D.5 The Fair Triathlon Code

This section contains the code that was used to compute the fair triathlons found in Chapter 4.

```
rm(list=ls())

## Converts km to miles ##
km2mi <- function(k)
{
0.621371192 * k
}

## Converts miles to km ##
mi2km <- function(m)
{
1/0.621371192 * m
}

nonlin.swim <-
function(x,tau1,tau2,beta,delta1=0.93,delta2=2.4)
```

```
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}
nonlin.bike <-
function(x,tau1,tau2,beta,delta1=24.86,delta2=112.0)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}
nonlin.run <-
function(x,tau1,tau2,beta,delta1=6.21,delta2=26.2)
{
tau1 + ( tau2 - tau1 ) * (1-exp(-beta*(x - delta1))) /
(1-exp(-beta*(delta2-delta1)))
}

pplnvar <- function(mu,sig2)
{
exp(2*mu)*( exp(sig2)*(exp(sig2)-1) )
}


swim.output <-
read.table("c:/temp2/sout.txt", sep=",", header=T)
bike.output <-
read.table("c:/temp2/bout.txt", sep=",", header=T)
run.output <-
read.table("c:/temp2/rout.txt", sep=",", header=T)

rubble.swim <- 5000:nrow(swim.output)
rubble.bike <- 5000:nrow(bike.output)
rubble.run <- 5000:nrow(run.output)

tau1a.swim <- mean(swim.output$Tau1[rubble.swim])
tau2a.swim <- mean(swim.output$Tau2[rubble.swim])
betaa.swim <- mean(swim.output$Beta[rubble.swim])
sig2a.swim <- mean(swim.output$SigmaSq[rubble.swim])
alphaa.swim <- mean(swim.output$Alpha[rubble.swim])

tau1a.bike <- mean(bike.output$Tau1[rubble.bike])
tau2a.bike <- mean(bike.output$Tau2[rubble.bike])
betaa.bike <- mean(bike.output$Beta[rubble.bike])
```

```
sig2a.bike <- mean(bike.output$SigmaSq[rubble.bike])
alphaa.bike <- mean(bike.output$Alpha[rubble.bike])

tau1a.run <- mean(run.output$Tau1[rubble.run])
tau2a.run <- mean(run.output$Tau2[rubble.run])
betaa.run <- mean(run.output$Beta[rubble.run])
sig2a.run <- mean(run.output$SigmaSq[rubble.run])
alphaa.run <- mean(run.output$Alpha[rubble.run])

windows()
xx <- seq(0.249,3*2.4,len=100)
vars.swim <-
pplnvar(nonlin.swim(xx, tau1a.swim, tau2a.swim, betaa.swim),
sig2a.swim)
sds.swim <- sqrt(vars.swim)
plot(sds.swim,xx,type="l",lwd=2,
ylim=c(0,90),
xlim=c(10,max(sds.swim)),
xlab="standard deviation",
ylab="distance" )

xx <- seq(9.32,3*112,len=100)
vars.bike <-
pplnvar(nonlin.bike(xx, tau1a.bike, tau2a.bike, betaa.bike),
sig2a.bike)
sds.bike <- sqrt(vars.bike)
lines(sds.bike,xx,type="l",lwd=2)


xx <- seq(3,3*26.2,len=100)
vars.run <-
pplnvar(nonlin.run(xx, tau1a.run, tau2a.run, betaa.run),
sig2a.run)
sds.run <- sqrt(vars.run)
lines(sds.run,xx,type="l",lwd=2)


## Function to find the swim distance given the Std.Dev.
get.swim.dist <-
function(s,tau1,tau2,beta,sig2,d1=0.93,d2=2.4,total.iter=1000)
{
cnt <- 1
upp <- 7.8
```

```
low <- 0
span <- upp-low
delta <- 1
out <- NULL
trail <- numeric(total.iter)
while(delta>0.000001 & cnt<total.iter)
{
cand.dists <- seq(low, upp, len=1000)
std.devs <-
sqrt(pplnvar(
nonlin.swim(cand.dists, tau1, tau2, beta), sig2))
errors <- abs(std.devs-s)
span <- 0.9*span
best <- cand.dists[errors==min(errors)]
trail[cnt] <- best
upp <- best + span/2
low <- max(best - span/2,0)
cnt <- cnt + 1
delta <- min(errors)
}
out$std.dev <- std.devs[errors==min(errors)]
out$n.iter <- cnt-1
out$dist <- best
out$error <- min(errors)
out$trail <- trail[1:length(trail)-1]
out
}

## Function to find the bike distance given the Std Dev
get.bike.dist <-
function(s,tau1,tau2,beta,sig2,d1=24.86,d2=112,total.iter=1000)
{
cnt <- 1
upp <- 112
low <- 0
span <- upp-low
delta <- 1
out <- NULL
trail <- numeric(total.iter)
while(delta>0.000001 & cnt<total.iter)
{
cand.dists <- seq(low, upp, len=1000)
std.devs <-
sqrt(pplnvar(nonlin.bike(cand.dists, tau1, tau2, beta),
sig2))
errors <- abs(std.devs-s)
span <- 0.9*span
best <- cand.dists[errors==min(errors)]
trail[cnt] <- best
upp <- best + span/2
low <- max(best - span/2,0)
cnt <- cnt + 1
delta <- min(errors)
}
out$std.dev <- std.devs[errors==min(errors)]
out$n.iter <- cnt-1
out$dist <- best
out$error <- min(errors)
out$trail <- trail[1:length(trail)-1]
out
}

get.run.dist <-
function(s,tau1,tau2,beta,sig2,d1=6.21,d2=26.2,total.iter=1000)
{
cnt <- 1
upp <- 26.2
low <- 0
span <- upp-low
delta <- 1
out <- NULL
trail <- numeric(total.iter)
while(delta>0.000001 & cnt<total.iter)
{
cand.dists <- seq(low, upp, len=1000)
std.devs <-
sqrt(pplnvar(nonlin.run(cand.dists, tau1, tau2, beta), sig2))
errors <- abs(std.devs-s)
span <- 0.9*span
best <- cand.dists[errors==min(errors)]
trail[cnt] <- best
upp <- best + span/2
low <- max(best - span/2,0)
cnt <- cnt + 1
delta <- min(errors)
}
```

```
out$std.dev <- std.devs[errors==min(errors)]
out$n.iter <- cnt-1
out$dist <- best
out$error <- min(errors)
out$trail <- trail[1:length(trail)-1]
out
}

#### A Sampling of Fair Triathlons ####
kdist <- c(10,15,20,25)
new.dists <- km2mi(kdist)
fair.ones <- matrix(numeric(length(new.dists)*3), ncol=3)
for( i in 1:length(new.dists) )
{
fair.ones[i,3] <- new.dists[i]
std.dev <-
sqrt(pplnvar(
nonlin.run(new.dists[i], tau1a.run, tau2a.run, betaa.run),
sig2a.run))
swim.junk <-
get.swim.dist(std.dev,tau1a.swim,tau2a.swim,betaa.swim,sig2a.swim)
fair.ones[i,1] <- swim.junk$dist
bike.junk <-
get.bike.dist(std.dev,tau1a.bike,tau2a.bike,betaa.bike,sig2a.bike)
fair.ones[i,2] <- bike.junk$dist
}
ratios <-
matrix(numeric((ncol(fair.ones)-1)*nrow(fair.ones)),
nrow=nrow(fair.ones))
for(i in 1:nrow(ratios))
{
ratios[i,1]<-fair.ones[i,2]/fair.ones[i,1]
ratios[i,2]<-fair.ones[i,3]/fair.ones[i,1]
}
round(fair.ones, d=1)
round(ratios, d=1)

#### The Longest Fair Triathlon ####
max.s <-
sqrt(pplnvar(nonlin.swim(7.8,tau1a.swim,tau2a.swim,betaa.swim),
sig2a.swim))
get.bike.dist(max.s,
tau1a.bike,
tau2a.bike,
betaa.bike,
sig2a.bike)$dist
get.run.dist(max.s,
tau1a.run,
tau2a.run,
betaa.run,
sig2a.run)$dist

#### The Fair Half Ironman ####
std.dev <-
sqrt(pplnvar(nonlin.run(13.1, tau1a.run, tau2a.run, betaa.run),
sig2a.run))
get.swim.dist(std.dev,
tau1a.swim,
tau2a.swim,
betaa.swim,
sig2a.swim)$dist
get.bike.dist(std.dev,
tau1a.bike,
tau2a.bike,
betaa.bike,
sig2a.bike)$dist

#### The Fair Full Ironman ####
std.dev <-
sqrt(pplnvar(nonlin.swim(2.4,tau1a.swim,tau2a.swim,betaa.swim),
sig2a.swim))
get.bike.dist(std.dev,
tau1a.bike,
tau2a.bike,
betaa.bike,
sig2a.bike)$dist
get.run.dist(std.dev,
tau1a.run,
tau2a.run,
betaa.run,
sig2a.run)$dist

#### The Fair Full Ironman 2 ####
std.dev <-
sqrt(pplnvar(nonlin.run(26.2,tau1a.run,tau2a.run,betaa.run),
sig2a.run))
```

```
get.swim.dist(std.dev,
tau1a.swim,
tau2a.swim,
betaa.swim,
sig2a.swim)$dist
get.bike.dist(std.dev,
tau1a.bike,
tau2a.bike,
betaa.bike,
sig2a.bike)$dist
```

Bibliography

Bayes, T. (1764), "An Essay Towards Solving a Problem in the Doctrine of Chances," *Philosophical Transactions of the Royal Society of London*, 53, 370–418.

Casella, G. and George, E. I. (1992), "Explaining the Gibbs Sampler," *American Statistician*, 46, 167–174.

Chambers, J. M. (1973), "Fitting Nonlinear Models: Numerical Techniques," *Biometrika*, 60, 1–13.

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

Cook, R. D. and Tsai, C. (1985), "Residuals in Nonlinear Regression," *Biometrika*, 72, 23–29.

Dengel, D. R., Flynn, M. G., Costill, D. L., and Kirwan, J. P. (1989), "Determinants of Success During Triathlon Competition," *Research Quarterly for Exercise and Sport*, 60, 234–238.

Eaves, D. M. (1983), "On Bayesian Nonlinear Regression with an Enzyme Example," *Biometrika*, 70, 373–379.

Gajewski, B. J., Sedwick, J. D., and Antonelli, P. J. (2004), "A Log-normal Distribution Model of the Effect of Bacteria and Ear Fenestration on Hearing Loss: a Bayesian Approach," *Statistics in Medicine*, 23, 493–508.

Gallant, A. R. (1975), "Nonlinear Regression," *The American Statistician*, 29, 73–81.

Gelfand, A. E. and Smith, A. F. M. (1990), "Sampling-based Approaches to Calculating Marginal Densities," *Journal of the American Statistical Association*, 85, 398–409.

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2004), *Bayesian Data Analysis*, New York: Chapman and Hall/CRC.

Gelman, A. and Rubin, D. (1992), "Inference from Iterative Simulation Using Multiple Sequences," *Statistical Science*, 7, 457–511.

Geman, S. and Geman, D. (1984), "Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6, 721–741.

Graves, T., Reese, C. S., and Fitzgerald, M. (2003), "Hierarchical Models for Permutations: Analysis of Auto Racing Results," *Journal of the American Statitistical Association*, 98, 282–291.

Hartley, D. (1749), *Observations on Man, his Frame, his Duty, and his Expectations*, London: Richardson.

Hastings, W. K. (1970), "Monte Carlo Methods Using Markov Chains and their Applications," *Biometrika*, 57, 97–109.

Hills, S. E. (1987), "Reference Priors and Identifiability Problems in Non-Linear Models," *The Statistician*, 36, 235–240.

Kass, R. E., Carlin, B. P., Gelman, A., and Neal, R. M. (1998), "Markov Chain Monte Carlo in Practice: A Roundtable Discussion," *American Statistician*, 52, 93–100.

Laplace, P. S. (1774, reprinted in 1986), "Memoir on the Probability of the Causes of Events," *Statistical Science*, 1, 364–378.

Laursen, P. B. and Rhodes, E. C. (2001), "Factors Affecting Performance in an Ultraendurance Triathlon," *Sports Medicine*, 31, 195–209.

Lindley, D. V. and Smith, A. F. M. (1972), "Bayesian Estimates for the Linear Model," *Journal of the Royal Statistical Society. Series B (Methodological)*, 34, 1–41.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953), "Equations of State Calculations by Fast Computing Machines," *Journal of Chemical Physics*, 21, 1087–1091.

Millet, G. P., Chollet, D., Chalies, S., and Chatard, J. C. (2002), "Coordination in Front Crawl in Elite Triathletes and Elite Swimmers," *International Journal of Sports Medicine*, 23, 99–104.

Raferty, A. E. and Lewis, S. M. (1996), "Implementing MCMC," in *Markov Chain Monte Carlo in Practice*, London: Chapman and Hall, pp. 115–130.

Reese, C. S., Calvin, J. A., George, J. C., and Tarpley, R. J. (2001), "Estimation of Fetal Growth and Gestation in Bowhead Whales," *Journal of the American Statistical Association*, 96, 915–938.

Sandland, R. L. and McGilchrist, C. A. (1979), "Stochastic Growth Curve Analysis," *Biometrics*, 35, 255–271.

Schnute, J. (1981), "A Versatile Growth Model with Statistically Stable Parameters," *Canadian Journal of Fisheries and Aquatic Sciences*, 38, 1128–1140.

Stigler, S. M. (1978), "Laplace's Early Work: Chronology and Citations," *Isis*, 69, 234–254.

— (1982), "Thomas Bayes's Bayesian Inference," *Journal of the Royal Statistical Society. Series A*, 145, 250–258.

— (1983), "Who Discovered Bayes's Theorem?" *The American Statistician*, 37, 290–296.

— (1986), "Laplace's 1774 Memoir on Inverse Probability," *Statistical Science*, 1, 359–363.

White, H. (1981), "Consequences and Detection of Misspecified Nonlinear Regression Models," *Journal of the American Statistical Association*, 76, 419–433.