THE UNIVERSITY OF WESTERN AUSTRALIA

Methodology for Bayesian monotonic polynomials

Author:

Andrew Anthony Manderson

Department of Mathematics

and Statistics

Supervisor:

Berwin Turlach

Department of Mathematics

and Statistics

Supervisor:

Kevin Murray

School of Population and Global

Health

This thesis is presented for the degree of

Master of Philosophy

of the University of Western Australia

June 13, 2018

Thesis Declaration

I, Andrew Manderson, certify that:

This thesis has been substantially accomplished during enrolment in the degree.

This thesis does not contain material which has been accepted for the award of any other degree or diploma in my name, in any university or other tertiary institution.

No part of this work will, in the future, be used in a submission in my name, for any other degree or diploma in any university or other tertiary institution without the prior approval of The University of Western Australia and where applicable, any partner institution responsible for the joint-award of this degree.

This thesis does not contain any material previously published or written by another person, except where due reference has been made in the text.

The work(s) are not in any way a violation or infringement of any copyright, trademark, patent, or other rights whatsoever of any person.

This thesis contains published work and/or work prepared for publication, some of which has been co-authored.

Signature: 

Date: 30/01/2018

Abstract

Many physical processes are postulated to obey a monotonic relationship, whereby the entity of interest must strictly increase or decrease as a function of a covariate. Monotonic polynomials are a popular tool for incorporating such *a priori* knowledge into statistical models, particularly in settings where irreducible noise induces unconstrained model fits which violate the presupposed monotonic relationship. This thesis develops novel Bayesian methodologies for fitting monotonic polynomials and selecting the appropriate polynomial model, in a variety of scenarios.

Our methodology for fitting basic monotonic relationships in a Bayesian framework is implemented in `BUGS` and `Stan`, two popular probabilistic programming languages, for the ease of use of practitioners. The model for basic relationships is then extended to handle longitudinal, repeated measures settings, where multiple realisations from the process of interest are recorded. To do so, random effects are incorporated into our methodology, resulting in a Bayesian hierarchical model. The nature and structure of our random effect model easily adjusts for missing data, and makes use of the information from recorded observations, whilst appropriately propagating uncertainty in the regions of missing data. Thus the model can serve as both an interpolation and predictive tool and is again implemented in `Stan` to facilitate practioner use, whilst also taking advantage of the powerful numerical algorthim underpinning `Stan` to appropriately sample our high dimensional parameter space. We provide methodology for model selection in the Bayesian monotonic polynomial framework, and in doing so implement a fast reversible jump Markov chain Monte Carlo algorithm to facilitate inference about the polynomial degree posterior distribution. Our implementation undergoes a number of validation exercises, including comparisons to other model selection methods for monotonic polynomials, and is made available in an `R` package entitled `rjmonopoly`.

Acknowledgements

Firstly, to my supervisors Berwin and Kevin. You have both been instrumental to this project, and more broadly to my general passion for statistics. Berwin, the impact you have had cannot be understated, and you have still (seemingly) yet to tire of me knocking on the door at any point during the day with a barrage of questions. Kevin, the past two years of work would have been no where near as effective without your input, occasional ribbings included.

I'd also like to thank Edward Cripps, not only for your input into the published portion of this project, but also for being amenable to conversations about all the other topics.

To my fellow research students, specifically Alex Khor, Calum Braham, Nick Gale, Ben Luo, Lachy Astfalck and Mike Bertolacci, for ensuring I didn't stare at diagnostic plots for days on end, and proof reading extensive sections of this Thesis.

To Josh Bon, for endless statistical discussion amongst many other things.

I would like to thank my parents and sister for their support, encouragement and advice, particularly over the past two years.

This research was supported by an Australian Government Research Training Program (RTP) Scholarship

AUTHORSHIP DECLARATION: CO-AUTHORED PUBLICATIONS

This thesis contains work that has been published.

Details of the work:

Manderson, A.A., Cripps, E., Murray, K. and Turlach, B.A. (2017).
Monotone polynomials using BUGS and Stan,
Australian & New Zealand Journal of Statistics **59**(4): 353–370.
Doi: 10.1111/anzs.12207.

Location in thesis:

Chapter 2: Bayesian Monotonc Polynomials using BUGS and Stan

Student contribution to work:

Developed Stan code; authored Stan sections, as well as the section pertaining to the calibration of the credible intervals and corresponding coverage (Prose that corresponds to Figure 2.5); and critically reviewed & edited manuscript.

Co-author signatures and dates:

Berwin Turlac [redacted] Date: 31/01/2018

Kevin Murray: Date: 31/01/18

Edward Cripp Date: 31/01/18.

Student signature: [redacted]

Date: 30/01/18

I, Berwin Turlach, certify that the student statements
regarding their contribution to each of the works listed above are correct.
Coordinating supervisor signature: [redacted]
Date: 31/01/2018

Contents

List of Figures

List of Tables

Introduction

Many physical processes are postulated to obey a monotonic relationship, whereby the entity of interest must strictly increase or decrease as a function of a covariate. Examples of such relationships include the height of human adolescents as they age (Cole et al.; 1998), the size of untreated malignant tumours over time (Barbara et al.; 1992), and the capacity of modern lithium ion batteries as they undergo multiple recharging cycles (Millner; 2010). Monotonic polynomials are useful tool for incorporating such *a priori* knowledge into statistical models (Bon et al.; 2017; Murray et al.; 2013, 2016), particularly in settings where irreducible noise induces unconstrained model fits which violate the presupposed monotonic relationship.

Given the ease with which monotonicity can be stipulated, we simply require the derivative to be strictly positive or negative, it can be a deceptively challenging constraint. It cannot be enforced via linear inequality constraints on the parameters, and as such it truncates the parameter space in a highly non-trivial way. An example of this is presented in in Figure 1.1, which was generated by fitting a monotonic polynomial of degree 5, and plotting the posterior samples of the regression coefficients in a pair wise manner. A hard, monotonicity induced boundary is visible in the upper left of the scatter plot of the linear and cubic terms (row 4, column 2), and strong correlation is present in a number of the other plots. This is despite the use of a number of numerical methods that aim to remove all possible correlation in the plots of Figure 1.1. Such technical details and reparameterisation techniques will be explained in detail in the forthcoming chapters, for now it suffices to say that this constraint is often more challenging than it appears to be.

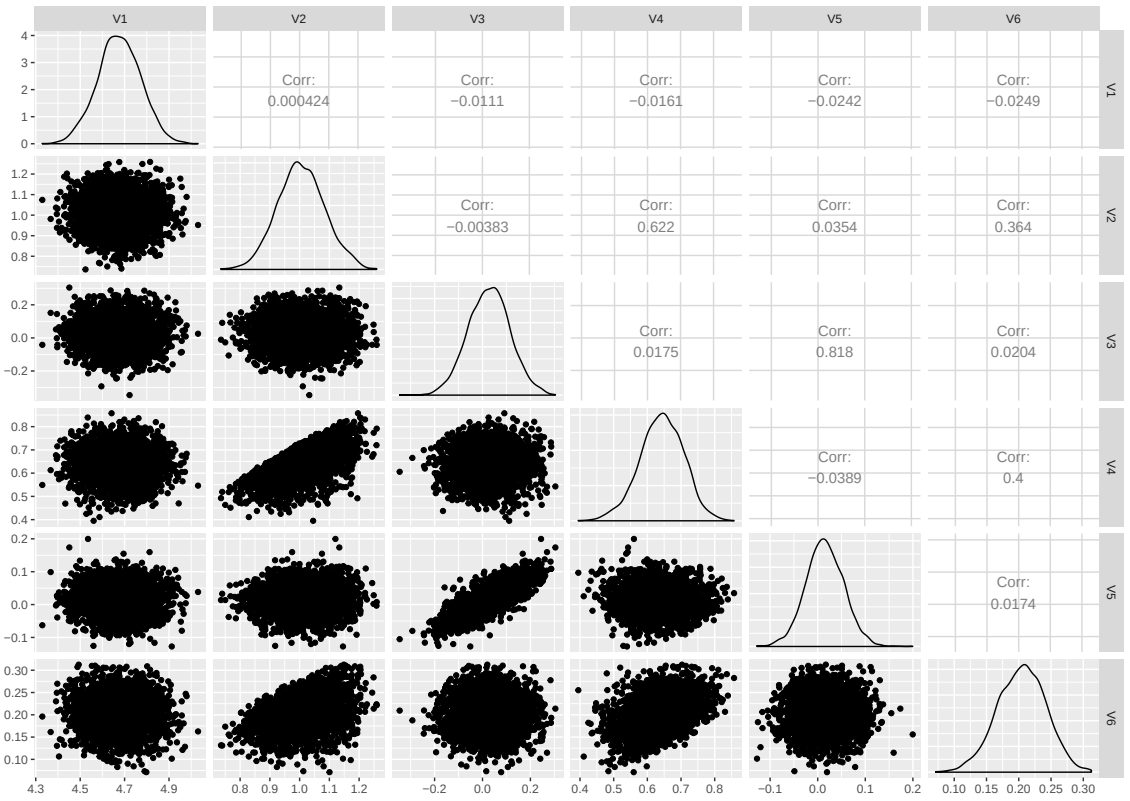


Figure 1.1: A pair wise plot of samples of the distribution of regression coefficients, for a degree 5 monotonic polynomial fit.

1.1 Monotonic methodology

The regularity with which scientists have encountered monotonic relationships has spurred substantial methodological research including: isotonic regression for ordinal data (Barlow et al.; 1972; Barlow and Brunk; 1972) and it's applications in smoothing (Friedman and Tibshirani; 1984); monotonic and general shape-constrained spline fitting (Dierckx; 1980; Utreras; 1985); as well as the monotonic polynomial, first considered by Elphinstone (1983), with later refinements by Hawkins (1994), Murray et al. (2013, 2016) and Bon et al. (2017). Monotonic polynomial methodologies such as Murray et al. (2013, 2016) build on the idea of Elphinstone (1983), whereby monotonicity naturally results from the chosen parameterisation, as opposed to it being enforced via constraints.

There has also been considerable work on curve fitting and shape constrained regression in a Bayesian framework, which was first considered by Geweke (1986). Such methodologies include, but are not limited to, Gibbs sampling to address constrained parameter and truncated data problems (Gelfand et al.; 1992), isotonic regression (Holmes and Heard; 2003; Neelon and Dunson; 2004), monotonic splines (Brezger and Steiner; 2008; Hazelton and Turlach; 2011), and hierarchical combinations there of (Shively et al.; 2009). More recent Bayesian methodological developments include monotonic multiple regression (Saarela and Arjas; 2011), monotone Gaussian processes (Lin and Dunson; 2013; Wang and Berger; 2016) and the more general technique of sequentially constrained Monte Carlo (Golchi and Campbell; 2014). Applications of such Bayesian methodology include crack growth prediction (Hermann et al.; 2016), dose-response modelling (Ohlssen and Racine; 2015), analysis of economic elasticity (Terrell; 1996; O'Donnell and Coelli; 2005) and economic efficiency (Henningsen and Henning; 2009), and variable selection under monotonicity constraints (Curtis; 2008).

This thesis adapts the monotonic parameterisation described by Murray et al. (2016) to a Bayesian framework, the combination of which we do not believe to have been previously considered. In doing so we take advantage of this new, faster, more flexible parameterisation, as well as popular probabilistic programming languages for Bayesian inference, to develop a methodology that has immediate practitioner value. We extend this methodology by including subject-specific terms, also known as random effects. Such models have been shown to be useful (O'Donnell and Coelli; 2005), and we again make use of popular probabilistic programming languages. Bon et al. (2017) developed a similar idea in a frequentist framework, but our focus differs, in that we are primarily concerned with the borrowing of information across individuals, particularly in the presence of missing data.

1.2 Constrained model selection

An important step in the application of any of the previously mentioned methodologies, is that of selecting the appropriate model. Specifying the ideal number of knots for a spline, or the optimal covariance and smoothness of a Gaussian process, is vitally important for model performance. The same issue is faced when fitting polynomial models, as the degree of the polynomial must be chosen with great care in order to ensure ideal model behaviour. For unconstrained Bayesian models, this issue has been extensively explored. Perhaps the most notable model selection heuristics are that of the information criteria, of which the most popular seem to be the Akaike information criterion (Akaike; 1973), the Bayesian information criterion (Schwarz; 1978), and the deviance information criterion (Spiegelhalter et al.; 2002). Other model selection methods include Bayes factors (Kass and Raftery; 1995; Raftery; 1996), and the encompassing concept of Bayesian model averaging (Hoeting et al.; 1999). The advantage of these latter methods is that they naturally incorporate the residual uncertainty about the value of the parameters; if discrete alternative model structures are proposed for the data, they are able to coherently incorporate this additional model uncertainty.

However, the topic of model selection in the presence of complex shape constraints, such as monotonicity, has received comparatively less attention. Whilst the previously mentioned model selection methods can still be used, some either implicitly assume that the parameter space is unconstrained, or have poor empirical performance in the presence of complex constraints. Current Bayesian methodology for constrained model selection includes, but is not limited to, the selection criteria of Abramovich and Grinshtein (2013) for structurally constrained regression, and the Bayes factor methodology of Kato and Hoijtink (2006) for mixed linear models subject to linear constraints. In the specific context of monotonic polynomial regression Hawkins (1994) recommended an F-test based methodology for model selection, which has good empirical performance despite violating some of the assumptions of the classical F-test, while Murray et al. (2016) made use of their

efficient parameterisation by applying an M-out-of-N bootstrap methodology, in order to select the polynomial degree with minimum prediction error.

In this thesis we implement a Reversible jump Markov chain Mote Carlo algorithm (Green; 1995) in order to simultaneously explore the model space, induced by allowing the polynomial degree to vary, and the parameter space. In doing so we are able to appropriately incorporate our monotonicity constraint via the prior distribution. Our implementation is faster than the M-out-of-N bootstrap of Murray et al. (2016), and is available as an `R` package for ease of use.

1.3 Thesis structure

The outline of this thesis is as follows: in Chapter 2 we present our Bayesian implementation of the monotonic polynomial parameterisation of Murray et al. (2016) in the popular probabilistic programming languages `BUGS` and `Stan`. We then extend this implementation to incorporate random effects in Chapter 3, which enable us to fit subject-specific curves in the presence of missing data. Details of our reversible jump implementation and validation exercises are contained in Chapter 4, and some conclusions and discussions of possible future work are in Chapter 5.

Bayesian Monotonic Polynomials using BUGS and Stan

In some research problems it might be necessary to impose monotonicity constraints on the regression curve while the main interest is actually some derived quantity of the regression curve, such as the location of its inflection points (Firmin et al.; 2011, 2012). As monotone polynomials have many useful properties, including that they are naturally strictly monotone and have easily identifiable derivatives, Murray et al. (2013) revisited the idea of using monotone polynomials and provided algorithms for fitting them to data based on the isotonic parameterisations proposed by Elphinstone (1983) and Hawkins' (1994) semi-definite programming algorithm. Murray et al. (2016) used a different isotonic parameterisation of monotone polynomials which leads to more efficient algorithms for fitting these to data and, more crucially, allows for fitting such polynomials to data when the function is only constrained to be monotone over a compact or semi-compact region.

All of these approaches for fitting monotone polynomials to data were developed in a frequentist framework, and to our knowledge, no work has been done in a Bayesian framework beyond the linear case. While it is possible to implement monotone polynomials in a Bayesian framework using tailor-made Markov chain Monte Carlo (MCMC) algorithms, we discuss here how the isotonic parameterisations studied in Murray et al. (2016) can be used to implement monotone polynomials in probabilistic programming languages such as BUGS (Lunn et al.; 2000, 2009, 2012) and Stan (Carpenter et al.; 2017).

Specifically, we consider inference about the regression parameters β_j, and the regression function, in the linear polynomial regression model:

$$Y = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_q x^q + \varepsilon, \tag{2.1}$$

where q is the degree of the polynomial, assumed to be monotone, and ε is an

error term that is assumed to follow a Gaussian distribution with mean 0 and variance σ_ε^2. Denoting the vector of regression parameters by $\boldsymbol{\beta} = (\beta_0, \beta_1, \ldots, \beta_q)^\top$, the usual parameterisation for the polynomial regression function is:

$$p(x) = p(x; \boldsymbol{\beta}) = \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_q x^q. \tag{2.2}$$

However, this parameterisation is not convenient to use if there are monotonicity constraints on the polynomial over a set $\mathcal{R} \subseteq \mathbb{R}$. Previously published work (Elphinstone; 1983; Hawkins; 1994; Heinzmann; 2008; Murray et al.; 2013) on monotone polynomials only considered the case $\mathcal{R} = (-\infty, \infty)$. Murray et al. (2016) proposed to use another isotonic parameterisation which allows, in addition to $(-\infty, \infty)$, the specification of more general regions, namely semi-compact intervals $[a, \infty)$ and compact intervals $[a, b]$ for finite $a, b \in \mathbb{R}$, on which the fitted polynomial satisfies a monotonicity constraint.

The structure of this chapter is as follows: in Section 2.1 we specify the mathematical framework used to ensure monotonicity. The specific implementation in BUGS and Stan is presented in Section 2.3, with the complete code provided in an online repository, available at either `https://github.com/hhau/BayesianMonPol`, or in the supporting information section of the online version of our corresponding publication (Manderson et al.; 2017). Numerical experiments were carried out on simulated and real data sets and a summary of the results is given in Section 4, along with a discussion of the diagnostic outputs. Concluding remarks and discussion of future work are provided in Section 5.

2.1 Isotonic parameterisation

An alternative parameterisation to (2.2), which is suitable for monotonic polynomials, is obtained by writing the polynomial in the form

$$p(x) = \beta_0 + \alpha \int_0^x \check{p}(u)\, du, \tag{2.3}$$

where $\check{p}(u)$ is required to be non-negative on $\mathcal{R}$. Non-negative polynomials are well studied in the literature (see for example Marshall; 2008, and references therein). From (2.3) it follows immediately that the first derivative of the polynomial $p(x)$ is $p'(x) = \alpha\check{p}(x)$, which ensures the monotonicity of the polynomial in either an increasing or decreasing manner, depending on whether α is positive or negative.

The isotonic parameterisations considered by Murray et al. (2013), based on and extending work by Elphinstone (1983), wrote $\check{p}(x)$ as a product of quadratic polynomials, where each of these quadratics had either conjugate complex roots or a real root of multiplicity 2. These parameterisations of non-negative polynomials are not suitable for computations in a Bayesian framework, as it is difficult to specify suitable priors for the location of the roots of the quadratics terms, and to design efficient MCMC schemes. Furthermore, these parameterisations cannot be readily extended to more general forms of $\mathcal{R}$.

Based on Murray et al. (2016), we investigate another isotonic parameterisation for $\check{p}(u)$ which is based on the following proposition.

Proposition 2.1.1. A polynomial $\check{p}(x)$ of degree $q-1 \geq 0$ is non-negative

1. on $\mathcal{R} = (-\infty, \infty)$ if and only if $q = 2K+1$ and it can be written as the sum of two squared polynomials

$$\check{p}(x) = p_1(x)^2 + p_2(x)^2, \quad \text{for all } x \in \mathbb{R} \tag{2.4}$$

   where $p_1(x)$ and $p_2(x)$ are polynomials whose degrees are at most K, where $K \in \mathbb{N}$;

2. on $\mathcal{R} = [a, \infty)$ if and only if it can be written as

$$\check{p}(x) = p_1(x)^2 + (x-a)p_2(x)^2, \quad \text{for all } x \in \mathbb{R} \tag{2.5}$$

   where,

(a) if $q = 2K + 1$, $p_1(x)$ and $p_2(x)$ are polynomials whose degrees are at most K and $K - 1$, respectively, and,

(b) if $q = 2K$, both degrees are at most $K - 1$;

3. on $\mathcal{R} = [a, b]$ if and only if it can be written as

(a) if $q = 2K + 1$:

$$\check{p}(x) = p_1(x)^2 + (x - a)(b - x)p_2(x)^2, \quad \text{for all } x \in \mathbb{R} \tag{2.6}$$

where $p_1(x)$ and $p_2(x)$ are polynomials whose degrees are at most K and $K - 1$, respectively,

(b) if $q = 2K$:

$$\check{p}(x) = (b - x)p_1(x)^2 + (x - a)p_2(x)^2, \quad \text{for all } x \in \mathbb{R} \tag{2.7}$$

where $p_1(x)$ and $p_2(x)$ are polynomials with their degree at most $K - 1$.

For the purpose of Proposition 2.1.1, $p_2(x) = 0$ if $p_2(x)$ has a negative degree (cases 2(a) and 3(a) when $q = 1$ and $K = 0$). This proposition can be proved using the theory of Tchebycheff systems (Karlin and Studden; 1966) or the theory of canonical moments (Dette and Studden; 1997). A proof that does not utilise such deep mathematical theories can be found in Brickman and Steinberg (1962). The latter authors also point out that these parameterisations of monotone polynomials are not unique; a point that is further discussed in Section 2.3.

We use Proposition 2.1.1, by fixing α in (2.3) to be either -1 or 1, depending on whether the polynomial should be monotone decreasing or monotone increasing, over $\mathcal{R}$. We denote the coefficients of $p_1(x)$ and $p_2(x)$ in (2.4)–(2.7) by $\boldsymbol{\beta}_1 = (\beta_{01}, \beta_{11}, \ldots, \beta_{q_1 1})^\top$ and $\boldsymbol{\beta}_2 = (\beta_{02}, \beta_{12}, \ldots, \beta_{q_2 2})^\top$, with $q_1, q_2 \in \{K - 1, K\}$. The coefficients $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ of the polynomials $p_1(x)^2$ and $p_2(x)^2$, can be calculated by convolution of $\boldsymbol{\beta}_1$ with itself and by convolution of $\boldsymbol{\beta}_2$ with itself. By adding the corresponding entries in $\boldsymbol{\gamma}_1$ and $\boldsymbol{\gamma}_2$ we can determine the coefficients $\boldsymbol{\gamma} = (\gamma_0, \ldots, \gamma_{q-1})^\top$ of the polynomial:

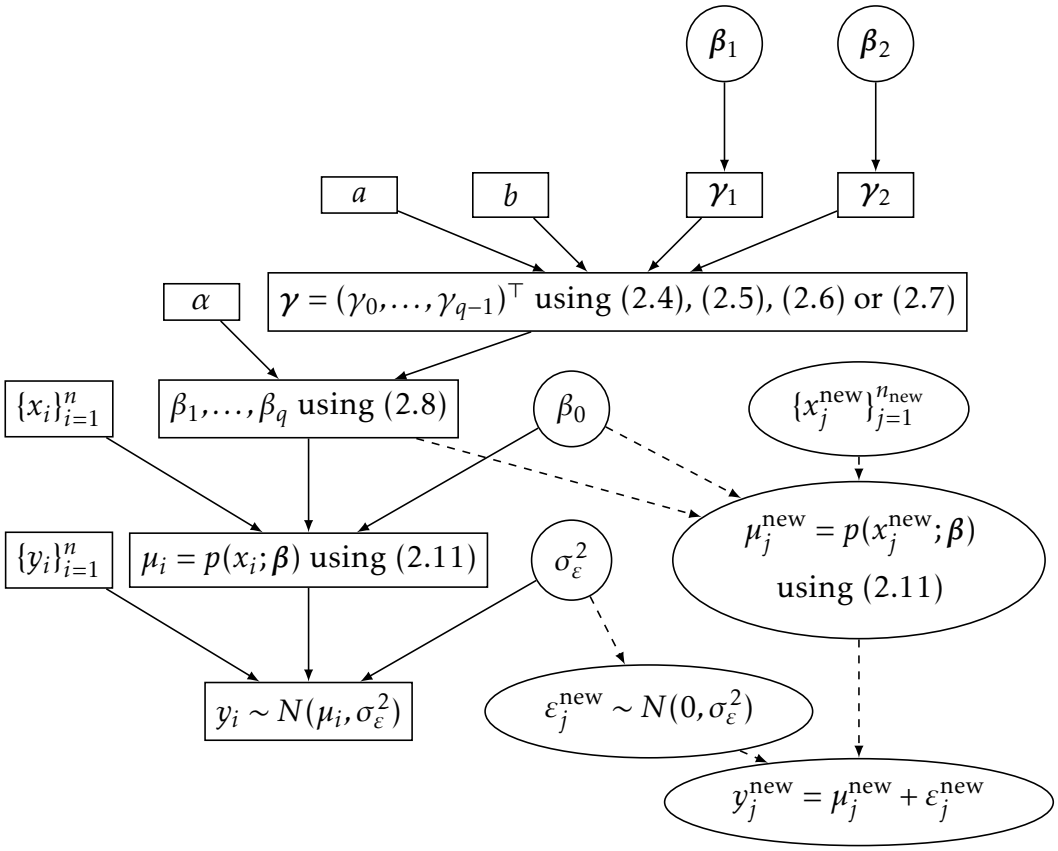


Figure 2.1: Directed acyclic graph of the fitted model. Quantities in circles are parameters of the model for which priors can be specified. Quantities in rectangles are either observed data or derived quantities. Quantities in ellipses are optional and would be used to calculate credibility intervals and/or prediction intervals.

$$\check{p}(x) = \gamma_0 + \gamma_1 x + \cdots + \gamma_{q-1} x^{q-1}.$$

From $\boldsymbol{\gamma}$ we can readily calculate $\boldsymbol{\beta}$ as:

$$\boldsymbol{\beta} = \left(\beta_0, \alpha\gamma_0, \alpha\frac{\gamma_1}{2}, \ldots, \alpha\frac{\gamma_{q-1}}{q}\right)^\top. \tag{2.8}$$

Once $\boldsymbol{\beta}$ is determined we can easily evaluate $p(x)$ for arbitrary values of x using Horner's scheme (see, among others, Fausett; 2003). This is done to achieve numerical stability, which can be problematic when evaluating high order polynomials,

see (2.11) in Section 2.2.4 for implementation details. Figure 2.1 depicts the model that we are fitting as a directed acyclic graph and illustrates the flow of calculations just discussed. Details of the implementation of these calculations in either BUGS or Stan are discussed in more detail in the next section.

2.2 Implementation

This section discusses some aspects of implementing the fitting of monotone polynomials in a Bayesian ramework using BUGS and Stan. Full listings of the code are given in the online supplementary material that accompanies the publication (Manderson et al.; 2017) of this chapter, and are also avaliable at https://github.com/hhau/BayesianMonPol. The code is also avaliable in Appendicies A.1.1–A.1.5. For the BUGS language, the five cases listed in Proposition 2.1.1 were implemented in separate files while for Stan, due to its better program flow commands, a singe file suffices. While several other programming languages can be interfaced to BUGS and Stan, we focus here on their interfaces to R, and describe in our online supplementary material appropriate R scripts.

2.2.1 Supplying the necessary data

BUGS: The code for the five BUGS files was designed such that they all have the same user interface. That is, the user will either have to specify the same information in the data section when running this code directly in BUGS, or pass the same data from R to BUGS when using any of the packages that interface R with WinBUGS (Sturtz et al.; 2005), OpenBUGS (Thomas et al.; 2006; Yan and Prates; 2013) or JAGS (Plummer; 2016). Specifically, the observed x and y values have to be passed to vectors called x and y, and the number of observed data points have to be passed to N. Given q and K are such that $q = 2K$ or $2K+1$, and q is the degree of the polynomial, then q and K should be passed to q and K (the JAGS implementations also define, for convenience, a node called Kp1 holding the value $K+1$; see the last lines of Appendices A.1.1–A.1.5; as will become clear from the discussion in

Section 2.2.2 this is not necessary for the implementations in `WinBUGS` or `OpenBUGS`). Furthermore, $\alpha \in \{-1, 1\}$ should be passed to `alpha`, and, if required (i.e. when $\mathcal{R}$ is compact or semi-compact), a and/or b should be passed to `a` and `b`. Finally, to obtain posterior means and/or predictions at specific points of the abscissa, these should be passed to an object named `xnew` and the number of such points to `Nnew`. This covers the discussion of all the observed data quantities in Figure 2.1. How to specify starting values for the parameter of the models will be discussed below when we propose suitable priors for these parameters in Section 2.2.5.

`Stan`: The `Stan` implementation requires the same input, and uses the same names for the objects, as the `BUGS` implementations; see the data section in lines 38–73 in Appendix A.2. However, for the `Stan` implementation values for a and b *always* have to be passed to `a` and `b`, respectively, although they can be specified as $-\infty$ or ∞. Our `Stan` implementation also requires the user to specify the form of the region over which the polynomial should be monotone, whether it be compact, semi-compact or unconstrained. This is done through an additional input parameter named `operation_mode` (the operation modes are 1, 2, 3 and 4, which correspond to the regions $(-\infty, \infty)$, $(-\infty, b]$, $[a, \infty)$ and $[a, b]$ respectively).

2.2.2 Determining γ_1 and γ_2 by convolution The coefficients of, say, the polynomial $p_1(x)^2$, i.e. the vector $\boldsymbol{\gamma}_1 = (\gamma_{01}, \gamma_{01}, \ldots, \gamma_{2q_1,1})^\top$, can be determined from the coefficients of the polynomial $p_1(x)$, the vector $\boldsymbol{\beta}_1 = (\beta_{01}, \beta_{11}, \ldots, \beta_{q_1 1})^\top$, by the following convolution formula:

$$\gamma_{j1} = \sum_{i=\max(0,j-q_1)}^{\min(q_1,j)} \beta_{i,1}\,\beta_{j-i,1} \qquad j = 0, 1, \ldots, 2q_1, \tag{2.9}$$

with $\boldsymbol{\gamma}_2$ being determined in the same manner. Since `BUGS` and `Stan` both use one-based indexing for vectors, matrices and arrays, (2.9) requires re-writing as it is stated for a language that uses zero-based indexing.

BUGS: Assume $q_1 = K$ and the $K+1$ entries of the vector β_1 are stored in a vector called b1 of length $K+1$. Then the $2q_1+1 = 2K+1$ entries of the vector γ_1, stored in the object g1, can be determined as follows:

$$\texttt{g1}[j] = \sum_{i=\max(0,j-(K+1))+1}^{\min(K+1,j)} \texttt{b1}[i] * \texttt{b1}[j-i+1] \qquad j = 1,2,\ldots,2K+1. \tag{2.10}$$

Essentially, each entry of γ_1 is the inner product of two specific slices of the vector β_1, with the order of the entries in the second slice being reversed. Unfortunately, currently neither BUGS nor Stan seem to have a function that reverses a (sub-)vector, and neither language seems to support sub-setting by range indexes that are in reverse order (i.e. unlike R, neither language supports sub-setting of the form b1[3:1]).

Thus, to implement (2.10) in BUGS we first create a matrix, say bnrev, to store the entries of β_1 and β_2 in reverse order. This matrix is created from the matrix bn which, is a $(K+1)\times 2$ matrix containing the entries of β_1 and β_2 (initial values for bn can be specified as further discussed in Section 2.2.5). The code for this reversing is straightforward:

```
for(j in 1:(K+1)){
  for(k in 1:2){
    bnrev[j,k] ← bn[Kp1-j+1,k]
  }
}
```

Subsequently, we can implement (2.9), for both β_1 and β_2, via the following code:

```
for(j in 1:q){
  g1[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
                 bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                 ,1])
  g2[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
```

```
                                    bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                                    ,2])
  }
```

Unfortunately, JAGS is the only implementation of the BUGS modelling language in which it is possible to calculate the indices with which to subset a vector or matrix in the manner above. WinBUGS and OpenBUGS are not able to compile and execute the above for loop. To run the code in WinBUGS or OpenBUGS, the for loop has to be explicitly unrolled. The following R snippet produces the necessary code that needs to be substituted for this for loop, and is illustrated for an odd degree polynomial with $q = 3$ and $K = 1$:

```
1  even.deg <- q%%2 == 0
2
3  for(j in 1:q){
4    print(bquote(g1[.(as.double(j))] <-
5            inprod(bn[.(max(0,j-Kp1)+1):.(min(Kp1,j)),1],
6                   bnrev[.(max(0,Kp1-j)+1):.(min(q+even.deg-j,K)+1),1])))
7    print(bquote(g2[.(as.double(j))] <-
8            inprod(bn[.(max(0,j-Kp1)+1):.(min(Kp1,j)),2],
9                   bnrev[.(max(0,Kp1-j)+1):.(min(q+even.deg-j,K)+1),2])))
10 }
```

To run this snippet, the user has to specify K and decide whether the polynomial should have an even degree ($q = 2K$) or an odd degree ($q = 2K + 1$) in the initial three lines of the code; the remainder of the code is generic. The output of this snippet has to replace the corresponding part in the files at lines 32–37 in Appendices A.1.1—A.1.4, and lines 37–42 in Appendix A.1.5.

Finally, once the coefficients of γ_1 and γ_2 are calculated from β_1 and β_2, the additional convolutions, if required, of γ_2 with $(-a, 1)$ ((2.5) and (2.7)) and γ_1 with $(b, -1)$ ((2.7)) can be implemented in a straightforward manner; see lines 27–30 in Appendices A.1.2—A.1.4. To implement (2.6), the code in Appendix A.1.5 calculates first the convolution of γ_2 with $(b, -1)$ and then the convolution of this

result with $(-a, 1)$; see lines 27–35 in Appendix A.1.5. This code also determines at the same time the vector γ, i.e. the coefficients of the polynomial $\check{p}(x)$, and stores them in the object called gamma.

Stan: In Stan it is more convenient to write a general convolution function in the function definition block of the code. Our implementation follows the ideas used for the BUGS implementation and is given in lines 2–23 in Appendix A.2.

Due to the more rigid structure of Stan, in which the order of statements matters, the remainder of the calculations described above for BUGS have to be distributed over several blocks. The transformed data block (lines 75–125 in Appendix A.2) defines variables p1_length, p2_length, gamma_1_length and gamma_2_length that contain the length of the vectors β_1, β_2, γ_1 and γ_2, respectively. In the case that additional convolutions as per (2.5)–(2.7) are necessary, the variable gamma_2_temp_length contains the length of the vector of coefficients of $(x-a)p_2(x)^2$ or $(x-a)(b-x)p_2(x)^2$, while the variable gamma_1_temp_length contains the length of the vector of coefficients of $(b-x)p_1(x)^2$. The exact values of these variables are calculated according to the inputs q, K and operation_mode in lines 85–116 of Appendix A.2, in line with Proposition 2.1.1. In this block we also define two vectors of length 2 named upper_bound_vector and lower_bound_vector which contain the coefficient of the polynomials $(b-x)$ and $(x-a)$ (lines 118–124 in Appendix A.2).

The actual space for the vectors β_1 and β_2 is allocated in the parameters block; see lines 127–133 of Appendix A.2. Other parameters defined in this block, as per Figure 2.1, are sd_y for σ_ε and beta_zero for β_0.

Finally, the space for the vectors γ_1, γ_2, γ and β, as well as space for some temporary vectors in the case that additional convolutions as per (2.5)–(2.7) are necessary, is allocated in the transformed parameters block; see lines 135–195 in Appendix A.2. After initialising some of these vectors to zero (lines 144–148) we calculate γ_1 by convolving β_1 with itself in line 151, and likewise for γ_2 and β_2 in line 152. In lines 157–186, we calculate γ from γ_1 and γ_2 while performing

additional convolutions as required per Proposition 2.1.1.

2.2.3 Determining β Once the vector γ is calculated it is easy to determine the components of β and store them in an object called `beta`. The object `beta0` (in `BUGS`, `beta_zero` in `Stan`) is another parameter for which initial values can be specified and it is further discussed in Section 2.2.5. The corresponding `BUGS` code is:

```
for(j in 1:q){
  beta[j+1] ← alpha * gamma[j]/j
}
beta[1] ← beta0
```

The `Stan` code is (lines 188–191 of Appendix A.2):

```
beta_final[1] = beta_zero;
for(i in 1:q) {
  beta_final[i+1] = alpha * gamma[i] / i;
}
```

2.2.4 Implementing Horner's scheme for evaluating polynomials Horner's scheme is a numerically stable method to efficiently evaluate polynomials, by re-arranging the calculations in the following manner:

$$\begin{aligned} p(x) &= \beta_0 + \beta_1 x + \beta_2 x^2 + \cdots + \beta_q x^q = \beta_q x^q + \beta_{q-1} x^{q-1} + \cdots + \beta_1 x + \beta_0 \\ &= (((\cdots(((\beta_q x + \beta_{q-1})x + \beta_{q-2})x + \beta_{q-3})\cdots)x + \beta_2)x + \beta_1)x + \beta_0. \end{aligned} \tag{2.11}$$

This is more numerically stable as it avoids the direct computation of x^q; for higher values of q and $|x| < 1$ can result in small values for x^q, which when added or subtracted from x^{q-1} results in significant, avoidable numerical error. Similar issues can be present for $|x| >> 1$. For further details on the reduction in numerical

imprecision, and increase in computational efficiency, we direct the reader to Higham's detailed analysis. [@higham:02].

BUGS: A difficulty when implementing this scheme in BUGS is that the language does not allow logical nodes to appear multiple times on the left hand side of an assignment. Hence, we have to create a matrix to hold all the intermediate results for evaluating $p(x)$ at the observed x values for the current value of β. The following code to implement Horner's scheme can be found in lines 2–10 of all files in Appendices A.1.1–A.1.5:

```
for(i in 1:N){
  y[i] ~ dnorm(mu[i], tauy)

  horner[i,1] ← beta[q+1]
  for(k in 1:q){
    horner[i,k+1] ← horner[i,k]*x[i] + beta[q+1-k]
  }
  mu[i] ← horner[i,q+1]
}
```

Note that we also define the model for the y observations at the same time, namely, according to (2.1), that $y_i \sim \mathrm{N}(\mu_i, \sigma_\varepsilon^2)$ where $\mu_i = p(x_i)$, $i = 1, \ldots, n$. However, note that BUGS parameterises the normal distribution using a precision parameter, as opposed to R which uses the standard deviation. That is, the quantity tauy holds the values $1/\sigma_\varepsilon^2$. A summary on how BUGS and R parameterise common distributions is given in LeBauer et al. (2013).

Stan: In Stan, it is most convenient to implement Horner's scheme via a function in the function definition block. The necessary code is a straightforward implementation of (2.11) and is given in lines 25–35 in Appendix A.2.

Note that in Stan the model for the y observations has to be specified in the model, given in lines 197–199 of Appendix A.2, and that in Stan, just as in R,

the (univariate) normal distribution is parameterised by the mean and standard deviation.

2.2.5 Starting values for MCMC chains As depicted in Figure 2.1, the parameters for which starting values for the MCMC chains may be specified by the user are $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$, β_0 and σ_ε^2 (or, equivalently, on the precision $\tau_y = 1/\sigma_\varepsilon^2$). One possibility for choosing such starting values is to fit, analogously to Murray et al. (2016), a specific subset of unconstrained polynomials to the data and derive some initial values for $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$ and β_0 from these fits; detailed formulae are given in Murray (2015). While Murray et al. (2016) only needed starting values for the regression parameters, in a Bayesian framework we may also choose to specify starting values for σ_ε^2. If so, these starting values can be conveniently initialised by the square of the residual sum of squares of the initial polynomial fits.

It is worth mentioning that within our current framework, which uses MCMC for model fitting, we have more liberty regarding the choice of starting values. In particular, it is possible to choose as starting values $\beta_0 = \bar{y}$, $\boldsymbol{\beta}_1 = \mathbf{0}$, $\boldsymbol{\beta}_2 = \mathbf{0}$ and σ_ε^2 as the sample variance of the y_is. In the framework of Murray et al. (2016) these starting values would not be suitable as they would start the numerical minimisation of the residual sum of squares at a saddle point, i.e. a critical point, of that objective function.

Finally, we recommend running at least two chains, starting from different initial values, when fitting monotone polynomials to data, as further discussed in Section 2.3.

`BUGS`: We recommend specifying starting values when running the `BUGS` code in Appendices A.1.1–A.1.5, regardless of whether it is run directly from one of the implementations of the `BUGS` language (`WinBUGS`, `OpenBUGS`, `JAGS`) or from one of the interfaces between `R` and one of these implementations. The `R` scripts provided in the supplementary material illustrate how initial values could be determined.

Initial values for β_0 should be passed to `beta0`, the initial values for $\boldsymbol{\beta}_1$ and $\boldsymbol{\beta}_2$ should be passed to the $(K+1)\times 2$ matrix `bn`, and initial values for $1/\sigma_\varepsilon^2$ should be passed to `tauy`.

`Stan`: Due to the more robust nature of `Stan`'s sampling algorithms, we can rely on its built in initialisation procedure. For specific details see Stan Development Team (2017, pp. 113-114).

2.2.6 Choice of priors For the purposes of this exposition, we propose to use by default relatively vague priors on all parameters. It is trivial for the user to change the hyperparameters of the priors, or even the priors themselves.

`BUGS`: In `BUGS`, priors can be specified by using relatively vague normal priors on the regression parameters $\boldsymbol{\beta}_1$, $\boldsymbol{\beta}_2$ and β_0 and a relatively vague gamma prior on the precision parameter τ_y (or, equivalently, an inverse gamma prior on σ_ε^2). These can be specified as follows:

```
beta0 ~ dnorm(0, 0.001)
for(j in 1:(K+1)){
  for(k in 1:2){
    bn[j,k] ~ dnorm(0, 0.001)
  }
}
tauy ~ dgamma(0.01, 0.01)
sigy <- 1/sqrt(tauy)
```

See the code towards the end of each listing in Appendices A.1.1–A.1.5. Alternatively, if the user prefers to put a vague prior on σ_ε, the last two lines in the above snippet could be replaced with the following code:

```
sigy ~ dunif(0,10)
tauy <- 1/pow(sigy, 2)
```

We found that these priors are suitably vague if the x and y observations are rescaled so that they fall roughly into the interval $[-1, 1]$. This is the rescaling also recommended by Murray et al. (2013, 2016) so as to avoid numerical precision problems.

As described in Section 2.2.2, the values of β_1 and β_2 are stored in a $(K+1)\times 2$ matrix called `bn` in `BUGS` and initial values for these vectors should be passed to this object. An issue not yet addressed arises from the fact that for some parameterisations in (2.5)–(2.7), either β_1 or β_2 (or possibly both) is only of degree $K-1$, i.e. has K parameters. In this case, the corresponding entry in row $(K+1)$ of `bn` has to be initialised to zero *and* has to be kept fixed at zero during the MCMC iterations. The latter can be achieved in various ways in `BUGS`, for example by putting a Bernoulli prior with success probability zero on the parameter(s) in question:

```
bn[K+1,1] ~ dbern(0)
bn[K+1,2] ~ dbern(0)
```

See lines 45–46 of Appendices A.1.2 and A.1.4, but also line 46 of Appendix A.1.3, and line 51 of Appendix A.1.5.

Stan: `Stan` uses by default "flat" (improper) priors (Stan Development Team; 2017, Chapter 8.3). When these priors are used, the results from an analysis using `Stan` are similar to that of an analysis with `BUGS` which utilises the vague (proper) priors described above, which is encouraging.

Alternatively, a user might want to specify other priors for some, or all, of the parameters β_1, β_2, β_0 and σ_ε. This should be done within the `model` block, given in lines 197–199 of Appendix A.2.

2.2.7 Credible intervals and prediction intervals In a Bayesian framework it is easy to calculate either credible intervals for $p(x)$ or prediction intervals for future observations, at arbitrary x values. The vector of x values at which we desire credible and/or prediction intervals should be passed to the object `xnew`

and the length of this vector to Nnew. Then, during the MCMC calculations, we evaluate the polynomial at all values in xnew given the current value of β in each MCMC iteration. These values are realisations of the posterior distribution of $p(x)$ and allow us to calculate credible intervals. By adding random noise to these evaluations, using in each MCMC iteration the current value for σ_ε^2 (or its appropriate transformation), we obtain realisations from the posterior predictive distribution of $Y = p(x) + \varepsilon$ which allows us to calculate prediction intervals.

BUGS: Given the discussion above and in the previous sections, the following BUGS code needed for these calculations should be self-explanatory:

```
for(i in 1:Nnew){
  hnew[i,1] ← beta[q+1]
  for(k in 1:q){
    hnew[i,k+1] ← hnew[i,k]*xnew[i] + beta[q+1-k]
  }
  mupred[i] ← hnew[i,q+1]
  ypred[i] ← mupred[i] + eps[i]
  eps[i] ~ dnorm(0, tauy)
}
```

See lines 12–20 in Appendices A.1.1–A.1.5. If the analyses do not require the determination of credible intervals for $p(x)$, or prediction intervals for new observations, these lines can be deleted from the code.

Stan: In Stan the quantities Nnew and xnew have to be defined in the data block (lines 38–73), see lines 71–72 of Appendix A.2. The code for generating a fitted value and a predicted value at each MCMC iteration for each element of xnew has to be specified in the generated quantities block; see lines 201–207 of Appendix A.2.

While Stan parameterises functions related to *univariate* normal distributions by the mean and *standard deviation*, the *multivariate* normal distributions are pa-

rameterised by the mean vector and the *covariance matrix*. Fortunately, for the latter there are also implementations that accept the Cholesky factorisation of the variance-covariance matrix as an argument. In our case this Cholesky factorisation is easily determined. Consequently, we use the function `multi_normal_cholesky_rng` instead of `multi_normal_rng`. Alternatively, we could replace line 206 with a loop similar to those in the examples of Stan's manual (Stan Development Team; 2017, pp. 150, 156, 157, 258 and 260):

```
for(i in 1:Nnew)
  ypred = normal_rng(mupred[i], sd_y);
```

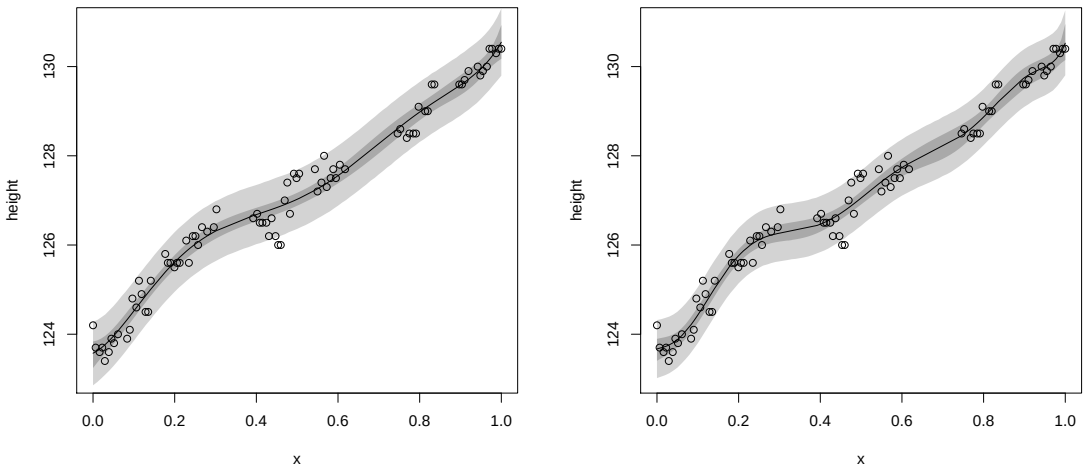


Figure 2.2: Plot of height of 10 year old boy against x, where x is scaled followup time over a one year period. Light grey areas indicate (pointwise) 95% prediction intervals while dark grey areas indicate (pointwise) 95% credible interval. The solid line is the posterior mean monotone polynomial regression curve. The left panel shows a polynomial of degree 7 that is montone on $[0, \infty)$, while the right panel uses a polynomial of degree 9.

2.3 Numerical experiments and results

We first illustrate our methods with the Stan implementation, using the data from Ramsay (1998) on the growth of a 10 year old boy. The measurements were taken over a period of 312 days during a school year, and there are a total of 83 height measurements. Given the period over which these measurements were taken, one would expect the underlying growth curve to be monotone. Due to the noise in the data, flexible regression methods that lack monotonicity constraints could result in periods of estimated decline in height. Figure 2.2 shows the data together with two fitted monotone polynomials of degree 7 and 9. Both fits also show (pointwise) 95% credible intervals and (pointwise) 95% prediction intervals. There is little difference between the two fitted regression curves, with the higher degree polynomial perhaps capturing the "flat" part around $x = 0.35$ more appropriately. Increasing the degree of polynomial further produced essentially the same fit as in the right panel of Figure 2.2, which is comparable to those obtained by monotone non-parametric fits to the same data; see, for example, Curtis (2008, Figure 4.3, p. 90).

When conducting the analysis in the foregoing example using the BUGS implementation, we obtained similar fits and credible/predictive intervals to those displayed in Figure 2.2. However, the traceplots of multiple chains of $\boldsymbol{\beta}$ showed insufficient mixing for the purposes of statistical inference on these parameters. Consequently, based on this and other numerical experiments (not reported here), we recommend fitting only relatively low order polynomials, up to around degree 5, in BUGS. We conjecture that the reason for this behaviour is that small changes to one parameter in either β_1 or β_2 can have a substantial effect on many parameters in $\boldsymbol{\beta}$ due to the way these quantities are related. The higher the degree of the monotone polynomial that is fitted to the data, the more parameters in $\boldsymbol{\beta}$ change when only one parameter is changed in either β_1 or β_2. Consequently, the sampling algorithms underlying BUGS do not effectively explore the parameter space of $\boldsymbol{\beta}$ for high degree monotone polynomials in our parameterisation. On the other hand,

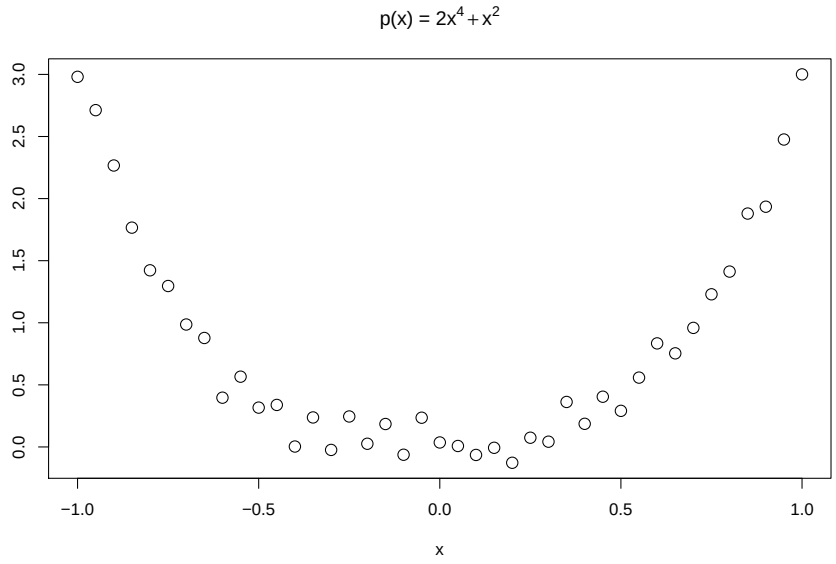


Figure 2.3: Data set simulated from the regression model $y = x^2 + 2x^4 + \varepsilon$, where ε is normally distributed with mean 0 and $\sigma_\varepsilon = 0.1$.

the Hamiltonian Monte Carlo sampling used by `Stan` does not suffer from this issue. We postulate that the more robust sampling algorithms that underlie `Stan`, including the use of gradient information in the proposal mechanism, are better able to explore the posterior parameter space appropriately.

As mentioned after Proposition 2.1.1, the parameterisation of non-negative polynomials via sum-of-squared polynomials is not necessarily unique. This is arguably most easily seen by considering the case $q = 1$ and $\mathcal{R} = (-\infty, \infty)$, when $p_1(x)$ and $p_2(x)$ in (2.4) are both constants. Clearly the parameterisation of monotone polynomials using this sum-of-squared polynomials is not unique. Consequently, monitoring for convergence of the Markov chains, we recommend monitoring only the parameters of interest, typically $\boldsymbol{\beta}$ and σ_ε. Futhermore, for high degree monotone polynomials the trace plots of the higher order coefficients in $\boldsymbol{\beta}$ might display wild fluctations, and as such it might be more sensible to monitor fitted and predicted values (`mupred` and `ypred` in the code).

Murray et al. (2016) reported experiencing fewer problems with local extrema in

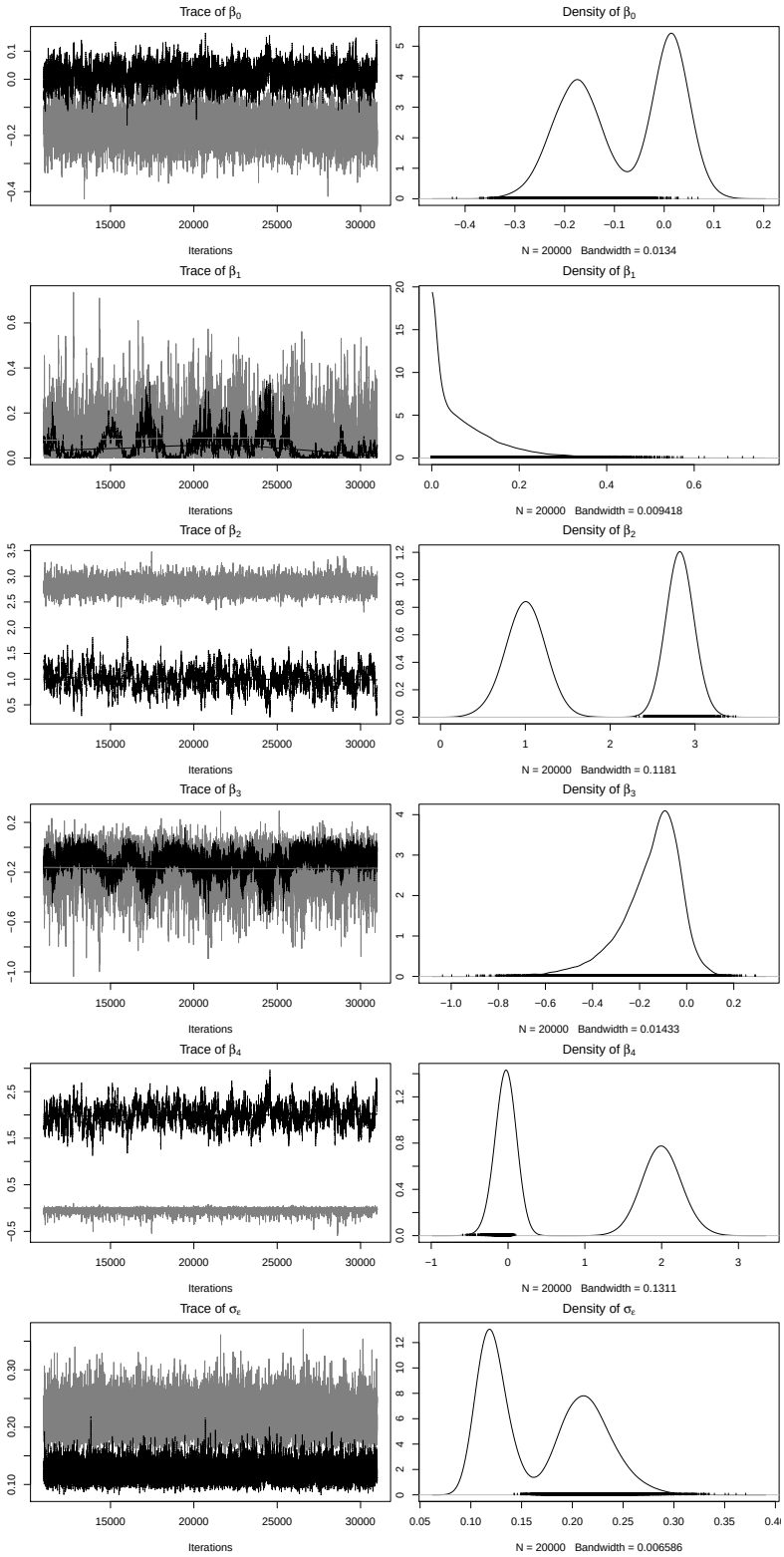


Figure 2.4: Diagnostic plots for parameter estimates when using two chains to fit a polynomial of degree 4 that is monotone on $[0,1]$ to the data in Figure 2.3. Panels in the first column show trace plots of the simulated values in both chains, while panels in the second column show density plots of these values. The parameters, arranged row-wise, are $\beta_0, \beta_1, \ldots, \beta_4$ and σ_ε.

the objective function when using the squared polynomial parameterisation of monotone polynomials for fitting such polynomials to data instead of Elphinstone-type parameterisations. Essentially, using the squared polynomial parameterisation, they did not experience problems with local extrema if $\mathcal{R} = (-\infty, \infty)$, rarely when $\mathcal{R}$ was a semi-compact interval and only occasionally when $\mathcal{R}$ was a compact interval. One would hope that in a Bayesian framework, due to the additional regularisation provided by the priors, there would be even fewer problems with multiple extrema in the objective function. However, at least when vague priors are used, the problem of multiple extrema still exists as illustrated here using the generated data set shown in Figure 2.3. This data set uses 41 equidistant x values in $[-1, 1]$ and y values were generated by the polynomial $x^2 + 2x^4$, adding normal noise with mean 0 and standard deviation $\sigma_\varepsilon = 0.1$. Figure 2.4 shows trace and density plots of the parameters $\beta_0, \beta_1, \ldots, \beta_4$ and σ_ε when running two chains in `BUGS` using code that fits polynomials which are monotone on $[0, 1]$. It can be clearly seen that one chain (black) fits a quartic polynomial to the data with a low residual noise, while the other chain (grey) fits a quadratic polynomial with a high residual noise to the data. Incidentally, when using these data with the function `monpol` of the `R` package `MonoPoly` (Turlach and Murray; 2016) and various random starting values, convergence to more than two local extrema were also observed.

Thus, as mentioned in Section 2.2.5, we recommend running several MCMC chains, in particular when fitting polynomials that are only constrained to be monotone over a compact or semi-compact interval. Not only does running multiple chains help with assessing whether the chains have converged and are mixing, it also helps with detecting problems due to multiple extrema in the objective function.

Finally, Murray et al. (2016) investigated bootstrap methods for statistical inference purposes, such as selecting the degree of the polynomial and obtaining (pointwise) confidence bands, for monotone polynomials. Here, we follow their terminology and say that a q degree monotone polynomial lies in $\mathcal{C}^q_{\mathcal{R}}$, which is the (non-pointed) closed convex cone that contains all possible values for $\boldsymbol{\beta}$, for

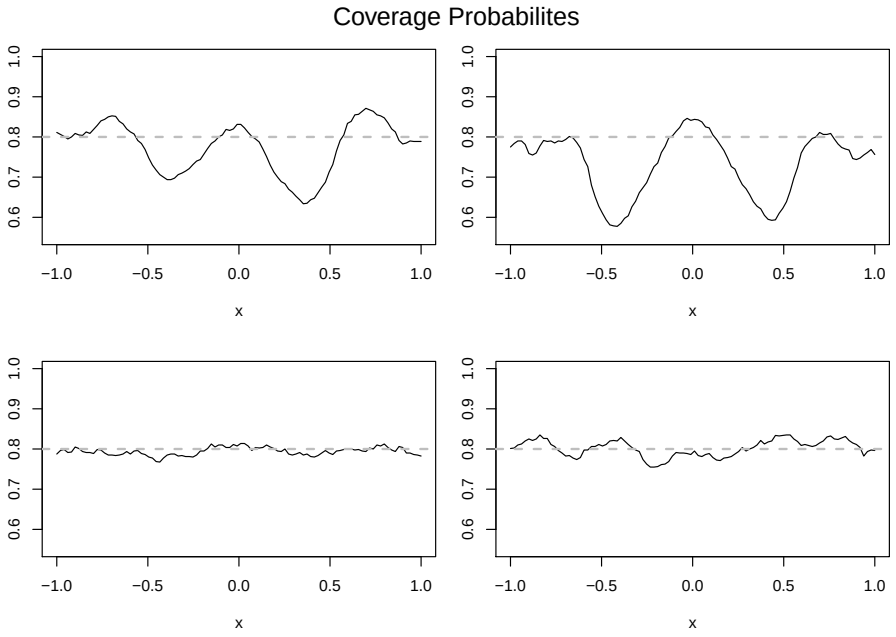


Figure 2.5: A demonstration of the calibration of the 80% credible intervals. For the top row the underlying polynomial is x^5 while for the lower row it is $3x + 2x^2 + x^5$. Actual (pointwise) coverage of the regression curve when fitting monotone polynomials of degree 5 are shown in the first column, while the second column is based on fitting monotone polynomials of degree 7.

which the resulting polynomial $p(x;\boldsymbol{\beta})$ is monotone on $\mathcal{R} \subseteq \mathbb{R}$. When using bootstrap methods for obtaining confidence bands for monotone polynomials, Murray et al. (2016) noticed that the confidence intervals have the wrong coverage if the underlying polynomial is of degree q and lies on the boundary of $\mathcal{C}^q_{\mathcal{R}}$. Thus it is of interest to see whether the credible intervals produced in the Bayesian framework are well calibrated.

A small simulation study to address this issue is summarised in Figure 2.5. For this simulation study we used the polynomials $p(x) = x^5$ and $p(x) = 3x + 2x^2 + x^5$. The former lies on the boundary of $\mathcal{C}^5_{\mathbb{R}}$ and $\mathcal{C}^7_{\mathbb{R}}$ while the latter lies inside $\mathcal{C}^5_{\mathbb{R}}$ but on the boundary of $\mathcal{C}^7_{\mathbb{R}}$. We repeatedly simulated data sets from these two polynomials and fitted monotone polynomials of degrees 5 and 7, as well as determining

80% credible intervals using a grid of x values. For each x value in this grid we determined the proportion of times that the true value of the regression curve at that point lay inside the 80% credible interval. The resulting proportions are shown in Figure 2.5, from which it is apparent that fitting monotone polynomials in a Bayesian framework, when using vague priors, suffers the same problem as fitting monotone polynomials in a frequentist framework using bootstrap methods for inference purposes. While these results confirm, somewhat reassuringly, the statistical folklore that Bayesian inference with uninformative priors is similar to frequentist inference using bootstrapping, it is mildly disappointing. Finding approaches that yield the correct coverage if the underlying polynomial lies on the boundary of its space remains an open research problem in both the Bayesian and frequentist frameworks.

2.4 Conclusions and future work

In this chapter we have demonstrated methods to fit monotone polynomials to data within a Bayesian framework using existing and readily available probabilistic languages such as BUGS and Stan. A byproduct of using a sum-of-squared polynomials parameterisation was that we could demonstrate how to implement a convolution operation in both languages; a computational operation with a non obvious implementation.

For our main topic of interest, monotone polynomials, we have provided easily implementable code, enabling the end user to adapt it to their particular scenario, and presented the results from numerical experiments applied to real and simulated data. The effectiveness of Stan was particularly pronounced for higher order polynomials, whilst issues with inter-chain mixing and convergence were identified in BUGS for such models. These issues were due to the sampling algorithms within BUGS being unable to appropriately explore a highly connected and correlated parameter space.

All code discussed here, and sample scripts that show how to use it from R, is freely available at `https://github.com/hhau/BayesianMonPol`.

Monotonic Polynomials with Random Effects in a Hierarchical Structure

The structure and implementation of the model described in Chapter 2 is an effective way to model a singular monotonic relationship between a response and a covariate. However, it is not obvious how one might use that structure to model repeated related responses as a function of a covariate. Take for example, the commonly used Berkeley growth data set recorded by Tuddenham and Snyder (1954), which consists of measurements of the height of 54 girls and 34 boys from age to 18, where each individual has 31 measurements. The measurements for the boys will be the primary data set of interest in this chapter, and are readily available in the `fda` package by Ramsay et al. (2014) in `R`. Naively, we might think to apply the methodology from Chapter 2 to each individual, but this would leave us unable to make population level inference, as there would be no mechanism with which to share information across individuals. This would also leave us unable to utilise information from other individuals when dealing with missing data, such as adjusting fitted curves for subjects who are only partially observed. Finally, we would be unable to statistically detect individual growth trajectories that differ substantially from the typical (population mean) curve. Such detection is vitally important in growth curve analysis.

In this chapter we present an augmentation to the monotonic structure presented in Chapter 2, where by we allow each individual's growth curve to differ via the use of random effects. This allows us to easily infer characteristics of the population curve, as well as pool together information about the shape of the typical growth curve, so as to be able to infer the likely behaviour of one or many individuals with missing data points. This is similar in nature to the methods of Bon et al. (2017), which develops a novel fitting algorithm for random effect models subject to shape constraints, such as the monotonic polynomial. However, the non-linear structure we employ serves as a substantial differentiator for this work as well as

our focus on appropriate performance in the presence of missing data.

We will formulate our extension as a Bayesian hierarchical model, which will allow us to exert control over the shrinkage behaviour that is ubiquitous in hierarchical modelling. This enables us to quickly detect individuals in a large data set that have deviated considerably from the population mean, in terms of both parameter estimates and fitted curves. It also enables the prediction of growth trajectories in the presence of missing data, without the need to impute values for missing covariates. This is comparable to the frequentist methodologies developed by Nummi (1997) and Hwang and Takane (2005), who extend non-linear growth curve models to include random effect terms, and handle missing data via a mixture of structural constraints and carefully constructed estimation algorithms. However, the estimation algorithms within Nummi (1997) and Hwang and Takane (2005) require additional prior knowledge, such as partial estimates of the random effects or variances thereof, which may not be available. The behaviour of the variance of the random effects in the presence of missing data is not obvious. By sampling the posterior distribution we can appropriately propagate any residual uncertainty that pertains to the fitted curve, which is particularly important around missing data points. The model methodology also allows for all parameters governing the fitted curve to be random, compared to the random intercept, slope, and quadratic parameters available in Bon et al. (2017).

This chapter begins by presenting the aforementioned augmented version of the mathematical structure presented in Chapter 2, which permits certain parameters to vary between subjects. The fitting methodology will then be discussed, along with the necessary data transformation and general fitting advice. The focus will then shift to performing inference in a variety of common scenarios, including a number of complete and missing data situations, as well as inference at both the population and individual levels. A wide variety of validation exercises are performed to demonstrate the appropriateness of the model's behaviour, including the calibration of its credible intervals and sensitivity to missing data. The notation

in this chapter is designed to aid the understanding of the implemenation avaliable at: `https://github.com/hhau/mono-random-effects`, and we direct readers to said implementation as the definitive reference.

3.1 Structural augmentations

The mathematical structure presented in Proposition 2.1.1 is appropriate when modelling basic monotonic relationships, such as one series of observations from a specific individual, or independent observations from many individuals. However, it does not naturally admit multiple similar sets of a known monotonic relationship. In order to appropriately model such a data set, we would like to augment the structure such that the parameters which govern the shape of the fitted curve are permitted to vary from subject to subject. Such subject-specific parameters are typically referred to as random effects, and mathematically we choose to represent them here with an additional subscript ρ.

To avoid obfuscating our proposed extension with the many of the cases of Proposition 2.1.1, we will instead only consider the case where we require the polynomial to be monotonic on $[0, \infty)$. The selection of this case was application motivated, as we are focused on human growth curve modelling. It also happens to coincide with our recommended data rescaling to $[0, 1]$, as the numerical algorithms underpinning `Stan` performs considerably better on this scale.

Recall that the structure in Proposition 2.1.1 ensured the monotonicity of our fitted polynomial, through the manipulation and combination of two, smaller degree polynomials. Consequently, we can make the coefficients of *one* of the constituent polynomials subject-specific, which results in all non-intercept coefficients of the fitted polynomial being subject-specific in nature. We then combine this with a random intercept to produce individual specific curves, governed wholly by subject-specific coefficients. However, we could also choose to only make a subset of the coefficients of the chosen constituent polynomial random, which

would result in *some* coefficients of the final polynomial remaining constant across individuals. As such, the number of subject-specific coefficients of the constituent polynomial acts as a tuning parameter, governing the flexibility of the subject-specific fitted polynomials.

Specifically, consider individual ρ from a model with r random effects per individual, with each individual modelled by a polynomial of degree q,

Proposition 3.1.1. a polynomial $\check{p}_\rho(x)$ of degree $q-1 \geq 0$ is non-negative for individual ρ on $\mathcal{R} = [a, \infty)$ if and only if it can be written as

$$\check{p}_\rho(x) = \mathrm{p}_{1,\rho}(x)^2 + (x-a)\mathrm{p}_2(x)^2, \quad \text{for all } x \in \mathbb{R} \tag{3.1}$$

where,

1. if $q = 2K+1$, $\mathrm{p}_{1,\rho}(x)$ and $\mathrm{p}_2(x)$ are polynomials whose degrees are at most K and $K-1$, respectively, and,
2. if $q = 2K$, both degrees are at most $K-1$.

We denote the coefficients of $\mathrm{p}_{1,\rho}(x)$ and $\mathrm{p}_2(x)$ in (3.1) by $\boldsymbol{\beta}$ in the following manner:

$$\boldsymbol{\beta}_{1,\rho} = (\beta_{01,\rho}, \beta_{11,\rho}, \dots, \beta_{r1,\rho}, \beta_{r+1}, \dots \beta_{q_1 1})^\top$$

$$\boldsymbol{\beta}_2 = (\beta_{02}, \beta_{12}, \dots, \beta_{q_2 2})^\top,$$

with $q_1, q_2 \in \{K-1, K\}$. The coefficients $\boldsymbol{\gamma}_{1,\rho}$ and $\boldsymbol{\gamma}_2$ of the polynomials $\mathrm{p}_{1,\rho}(x)^2$ and $\mathrm{p}_2(x)^2$, can again be calculated by convolving $\boldsymbol{\beta}_{1,\rho}$ with itself and by convolving $\boldsymbol{\beta}_2$ with itself. By adding the corresponding entries in $\boldsymbol{\gamma}_{1,\rho}$ and $\boldsymbol{\gamma}_2$ we can determine the coefficients $\boldsymbol{\gamma}_\rho = (\gamma_{0,\rho}, \dots, \gamma_{q-1,\rho})^\top$ of the polynomial:

$$\check{p}_\rho(x) = \gamma_{0,\rho} + \gamma_{1,\rho}x + \cdots + \gamma_{q-1,\rho}x^{q-1}.$$

From $\boldsymbol{\gamma}_\rho$ we can readily calculate $\boldsymbol{\beta}_\rho$ as:

$$\boldsymbol{\beta}_\rho = \left(\beta_{0,\rho}, \alpha\gamma_{0,\rho}, \alpha\frac{\gamma_{1,\rho}}{2}, \ldots, \alpha\frac{\gamma_{q-1,\rho}}{q}\right)^\top, \tag{3.2}$$

where α is as before, either 1 or -1 depending on whether or not we require the polynomial to be monotonically increasing or decreasing respectively.

The critical aspect of this adapted proposition is that $\boldsymbol{\beta}_{1,\rho}$ has at most $K+1$ or K coefficients, of which we have allowed the first r coefficients to be specific to individual ρ. The other $K+1-r$ or $K-r$ coefficients are constant across all individuals, and would typically be referred to as fixed effects. We have also added a subject-specific intercept in the form of $\beta_{0,\rho}$.

Note that due to the convolution operation, r does not relate to the number of random effects per individual in the manner one might anticipate. This is because r random terms in $\boldsymbol{\beta}_{1,\rho}$ result in $2q_1 - 1 - (q_1 - r) = r + q_1 - 1$ random terms in $\boldsymbol{\gamma}_{1,\rho}$, with the caveat that $r \in \{1, \ldots, q_1\}$. If r were to equal zero, then we would be fitting the same polynomial to each individual in the data set. Thus, due to this mapping between r and the number of random terms in the final polynomial, we effectively consider r to be a form of tuning parameter that allows the practitioner to adjust the flexibility of the model.

3.1.1 Hierarchical prior specification As we are framing our random effect model as a Bayesian hierarchical model, any extension to the parameter space requires us to specify a prior structure for these additional parameters. Such a prior structure should both appropriately capture any prior knowledge of our random effects, as well as induce a posterior distribution from which samples can be drawn. Fortunately, this is not a very restrictive set of requirements, as we know very little about the likely values of our random effects, and the powerful Hamiltonian Monte Carlo (HMC) sampling algorithm in `Stan` is able to effectively produce posterior samples. Considering the two requirements simultaneously

naturally results in the construction of a weakly informative prior structure. First, we specify the hierarchical structure for our new subject-specific parameters, beginning with the prior distribution for the r random elements of $\boldsymbol{\beta}_{1,\rho}$, herein denoted by $\boldsymbol{\beta}_{1,\rho}^{\text{random}}$:

$$\boldsymbol{\beta}_{1,\rho}^{\text{random}} \sim \mathrm{N}\left(\mu_{\beta_1}, \sigma_{\beta_1}^2\right),$$

and we do the same for the intercept term:

$$\beta_{0,\rho} \sim \mathrm{N}\left(\mu_{\beta_0}, \sigma_{\beta_0}^2\right).$$

The hyperparameter prior we are most interested in is the prior on $\sigma_{\beta_1}^2$, as this directly influences the amount of shrinkage we observe. If we select a prior that allocates more probability to values close to zero, we will see more shrinkage towards the mean curve in the fitted values. This is often desirable in applications with missing data, but less desirable in applications where individuals are distinct from each other. In the application considered in this chapter, we have a strong prior belief that individuals grow in a similar manner. Hence we place a reasonably tight, half-normal, prior on the random effect variance:

$$\sigma_{\beta_1}^2 \sim \mathrm{HN}\left(0, 0.1^2\right) \qquad \sigma_{\beta_1}^2 \in [0, \infty).$$

The small variance of this prior might appear incongruent with the notion of a weakly informative prior structure, however, it is not as restrictive as one might fear. This is because we rescale our data to $[0, 1]$ for numerical performance, which in turn shrinks the scale on which we anticipate our variance estimates. During model construction and validation, an $\mathrm{IG}(10, 0.1)$ prior[1] and a $\mathrm{HN}(0, 0.01^2)$ prior were also tested, which place considerably more mass near zero. Estimates were

[1] Using the same parameterisation as that of Stan (Stan Development Team; 2017), which uses shape and rate parameters.

very similar for all of the priors indicating that, in our application, the likelihood component contributes considerably more than the tested prior components. As such, we preferred the half-normal prior with the larger variance for computational reasons, which we would also expect to be a robust default prior for other applications.

The rest of the parameters in the model have diffuse, weakly informative priors on them, for which computational considerations (runtime, and effective sample size per nominal number of iterations) were the major motivation. On all parameters defined over the whole real line we place diffuse normal priors:

$$\begin{aligned} \mu_{\beta_1} &\sim \mathrm{N}\left(0, 100^2\right) & \mu_{\beta_0} &\sim \mathrm{N}\left(0, 100^2\right) \\ \beta_1^{\text{fixed}} &\sim \mathrm{N}\left(0, 100^2\right) & \beta_2 &\sim \mathrm{N}\left(0, 100^2\right), \end{aligned}$$

where β_1^{fixed} denotes the coefficients of β_1 that are the same for each individual. To avoid the model attributing all variability to the remaining variance parameter, we opt for half-normal priors as opposed to flat priors,

$$\begin{aligned} \sigma_\varepsilon^2 &\sim \mathrm{HN}\left(0, 1^2\right) & \sigma_{\beta_0}^2 &\sim \mathrm{HN}\left(0, 1^2\right) \\ & & \sigma_\varepsilon^2, \sigma_{\beta_0}^2 &\in [0, \infty). \end{aligned}$$

Specifically we need a non-flat prior, preferably with exponentially decaying tails, on σ_ε^2 to avoid the model attributing too much of the variation in the data to noise, which could result in nonsensical parameter estimates.

3.1.2 Posterior predictive distribution for individuals We sample the posterior predictive distribution (PPD) in order to make inference about the fitted polynomial at the individual level, typically using a set of new x values x_{new}, of which there are N_{new}:

$$\mu_{\text{new},\rho} = h\left(x_{\text{new}}, \boldsymbol{\beta}_\rho\right)$$
$$\left(\hat{y}_\rho \mid \boldsymbol{\beta}_\rho, \sigma_\varepsilon^2\right) \sim \mathrm{N}\left(\mu_{\text{new},\rho}\,,\, \sigma_\varepsilon^2 \mathcal{I}_{N_{\text{new}}}\right).$$

This defines the PPD for individual ρ with regression coefficients $\boldsymbol{\beta}_\rho$. Again we use Horner's scheme for evaluating the polynomial, which is discussed in Section 2.2.4 and denoted $h(\cdot,\cdot)$ here. As we are primarily interested in the model's ability to borrow information across individuals, this is the principal quantity of interest of our model.

3.1.3 Posterior predictive distribution for the population We would also like to sample the PPD at the population level, for example we would like to propagate μ_{β_1} through the hierarchical structure in lieu of $\boldsymbol{\beta}_\rho$, in order to calculate the mean population regression coefficients $\mu_{\boldsymbol{\beta}}$, and generate samples from the mean population curve:

$$\begin{aligned} \mu_{\text{new,pop}} &= h\left(x_{\text{new}}, \mu_{\boldsymbol{\beta}}\right) \\ \left(\hat{y}_{\text{pop}} \mid \boldsymbol{\beta}_{\text{pop}}, \sigma_\varepsilon^2\right) &\sim \mathrm{N}\left(\mu_{\text{new , pop}}, \sigma_\varepsilon^2 \mathcal{I}_{N_{\text{new}}}\right). \end{aligned} \tag{3.3}$$

However, we encounter a numerical issue when attempting to generate these quantities. As discussed in Section 2.3, the monotonicity enforcing structure is not necessarily unique. The addition of subject-specific terms at the $\mathrm{p}_{1,\rho}(x)$ level leads to regular disagreement across MCMC chains as to the posterior location of the subject specific terms. An example of this numerical behaviour is given in Appendix E, and appears to be similar to the 'label switching' behaviour often present in mixture models. The estimated value of μ_{β_1} is thus a poor representation of the actual distribution, as it has to average disparate within chain and across chain behaviour. Hence, when propagated through the hierarchical structure, it produces samples for $\hat{y}_{\text{pop}}$ that do not resemble a 'typical' individual.

This does *not* result in any issues at the level of the individual, as the fitted and predicted values, which we recommended monitoring in Section 2.3, behave

sensibly. That is, the samples for $\hat{y}_\rho$ are consistent within chains, as well as across multiple chains. However, it does leave us unable to perform PPD sampling at the population level in the manner we would like, described by Equation (3.3). We can still calculate appropriate *data set* level prediction intervals by considering the joint PPD of the individuals:

$$\Pr\left(\hat{y}_{\text{data}} \mid \boldsymbol{\beta}, \sigma^2\right) \sim \sum_{\rho} \Pr\left(\hat{y}_\rho \mid \boldsymbol{\beta}_\rho, \sigma^2\right), \tag{3.4}$$

Whilst the *data set* level is not strictly equivalent to *population* level inference in the typical random effect sense, if the individuals within the data set form a representative sample of the population of interest, inference should generalise appropriately. This is not an unreasonable assumption, as it underpins most statistical methodologies. The empirical behaviour of the joint distribution described in Equation (3.4) is consistent with the expected behaviour of a population level prediction interval, and is presented in Figure 3.2.

Some techniques that might resolve the identifiability issue will be suggested in Chapter 5, however the development and testing of such methodology is beyond the scope of this thesis.

3.1.4 Structural overview Figure 3.1 extends the graphical depiction of Figure 2.1, and has an additional layer that is combined to construct $\beta_{1,\rho}$. We have also switched to a diffuse $\text{N}(0, 100^2)$ prior for $\boldsymbol{\beta}_2$, as opposed to the improper flat prior used previously. All posterior quantities of interest are now typically subject specific.

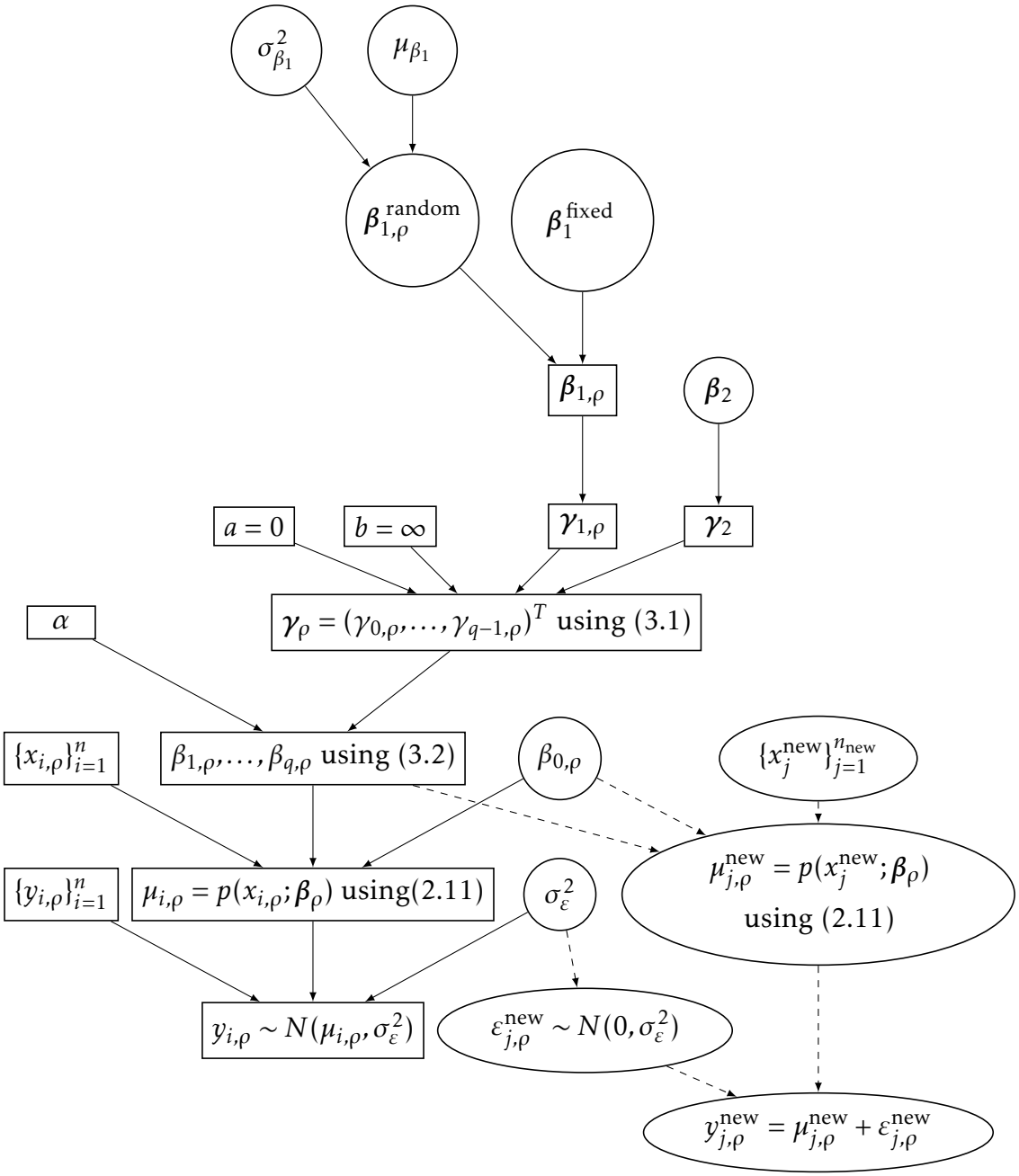


Figure 3.1: Directed acyclic graph of the fitted model, now extended to include random effects. Quantities in circles are parameters of the model for which priors can be specified. Quantities in rectangles are either observed data or derived quantities. Quantities in ellipses are optional and would be used to calculate credibility intervals and/or prediction intervals. Nodes for μ_{β_0} and $\sigma^2_{\beta_0}$ are not included in this graph, but can be thought of as circular nodes branching off $\beta_{0,\rho}$.

3.2 Fitting methodology

3.2.1 Practical considerations Statistical modelling scenarios for which random effect models are appropriate typically require 'ragged' data structures like lists to store the data of interest in, as methodological choices or missing data often results in an unequal number of data points per individual. This is easily dealt with using lists in R, however Stan lacks a native data type for ragged arrays. As such, we have to be somewhat thoughtful in order to correctly pass the data to the Stan model. We stack all the observations in one vector in a subject by subject manner, and construct an indexing vector that contains the indexes of each individual's first observation, with the last element being the final observation in the data set. This enables us to access the section of the observation vector that corresponds to a specific individual, whilst also allowing each individual to have a different number of observations.

Previously in this chapter we mentioned we rescale our x and y values to the $[0,1]$ interval, based on the maximum x and y observed across the entire data set. This increases the performance of the sampler considerably, compared to the unscaled data, which results in a shorter chain length (and hence runtime) for the same number of effective samples. This is due to the HMC methodology employed by Stan, as HMC samplers are not scale invariant.

The last of the practical considerations of note is the run time, particularly its sensitivity to choice of polynomial degree. For the data set of interest in this chapter, we would like to use a polynomial with relatively high degree, as human growth curves are known to have a number of inflection points. Though the exact number will vary from person to person, Murray (2015) demonstrates that individuals with 6 or 7 inflection points in their growth curve are quite common, which corresponds to fitted polynomials of degree 8 or 9. The sampler run time deteriorates for polynomials of the aforementioned degrees, with $1,000$ Stan iterations of the hierarchical model taking approximately 10 minutes for a degree 7 polynomial,

and approximately 1.5 hours for a degree 9 polynomial, for the same number of samples on the same (typical workstation) hardware. This results in a large loss of researcher time and unnecessarily wastes computational resources if an input is misspecified or the incorrect data is supplied. As such we would recommend that test fits are performed with a low degree polynomial, 3 or 4, before selecting the desired degree.

3.2.2 Minimal example A minimal, commented example is available at the Github repository for this chapter[2], which also contains the `Stan` model code, the `R` script that fits the model and code to produce some descriptive plots. The example will fit the the model to the full data set, produce a combined plot of every individual in the data set, as well as a subset of individuals for closer inspection.

3.2.3 Inference at the population level for a complete data set We first consider the output at the population level by examining the curve of the 'typical individual' depicted in Figure 3.2, which is equivalent to a population mean. The curve in this figure was produced using a polynomial of degree 8, with the maximum number of subject-specific terms in the underlying polynomial $p_{1,\rho}$, which is 4 in this instance. Unsurprisingly, the typical curve sits in the middle of the data. However, this curve lacks some features that are visible when the data is stratified by individual, notably it lacks growth spurts. This is a drawback of performing inference at this level, as we are inspecting an object that has already undergone some form of averaging. In this case we have implicitly averaged over the posterior distribution of our random effect terms, which has acted as a smoothing process, and has obscured patterns present in most individuals. As such, the following validation exercises will focus on quantities and behaviour at the individual level.

[2] `https://github.com/hhau/mono-random-effects`

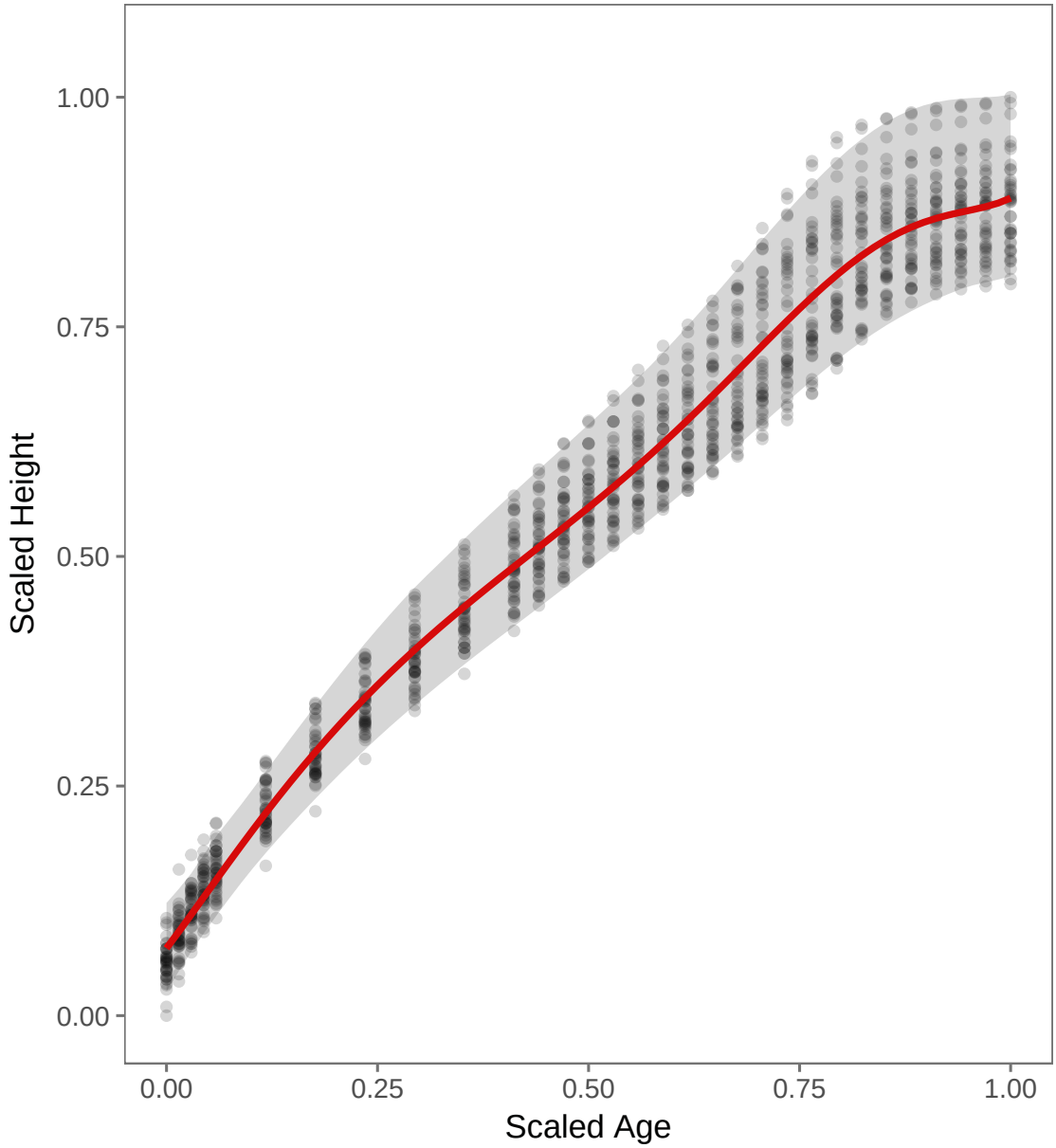


Figure 3.2: Plot of the 'typical individual' curve, i.e. the population posterior mean polynomial curve, atop the whole data set, with corresponding 95% prediction interval calculated using the method described in Section 3.1.3.

3.3 Validation exercises for missing data scenarios

In this section we will apply the model to data sets with missing data manifested in a variety of ways. This is a natural use of our hierarchical structure, as it enables

the borrowing of information across individuals, allowing for more appropriate predictive behaviour. This is particularly important in the presence of missing or withheld data, where this borrowing of information allows for appropriate estimation, via partial pooling, of all subject-specific parameters.

Results will be presented in a graphical manner, and to avoid visual overload we will limit the presentation of fitted curves and credible intervals to either a subset of the individuals, or one specific individual. We have included a figure of a fitted model, including fitted curves and credible intervals, to all individuals in Appendix C. This allows the reader to get a sense to the similarity between individuals, in both the original data and their fitted curves. The visual similarity should hopefully imbue a sense of believability in the model's pooled estimates, as well as its ability to appropriately infer curve behaviour for missing data points.

3.3.1 Additional point estimation For this first example we select one individual in our data set to be subject to our data removal process. This process begins by removing the final observation for our selected individual, and combining his (now truncated) data set with the full set of observations for the other 38 boys. We then fit our model and extract the relevant quantities for inference, including the prediction interval at the removed x value, as well as numerical summaries and other convergence diagnostics of the sampler. The next data set is then generated by removing one more observation from the end of our selected individual's data set, and the inference process is repeated at the removed x values. We continue this process until we have considered data sets where up to, and including, the final 15 data points are 'missing' for our selected individual. The motivation for performing this data removal one point at a time, is to observe the change in behaviour of the posterior predictive mean and 95% posterior prediction interval. Figure 3.3 displays the results of this exercise for `boy06` in the aforementioned `growth` data set, and we see improved mean tracking of this individuals trajectory and appropriate narrowing of the credible intervals as more data is included. This is ideal behaviour, particularly as this exercise closely corresponds to the common

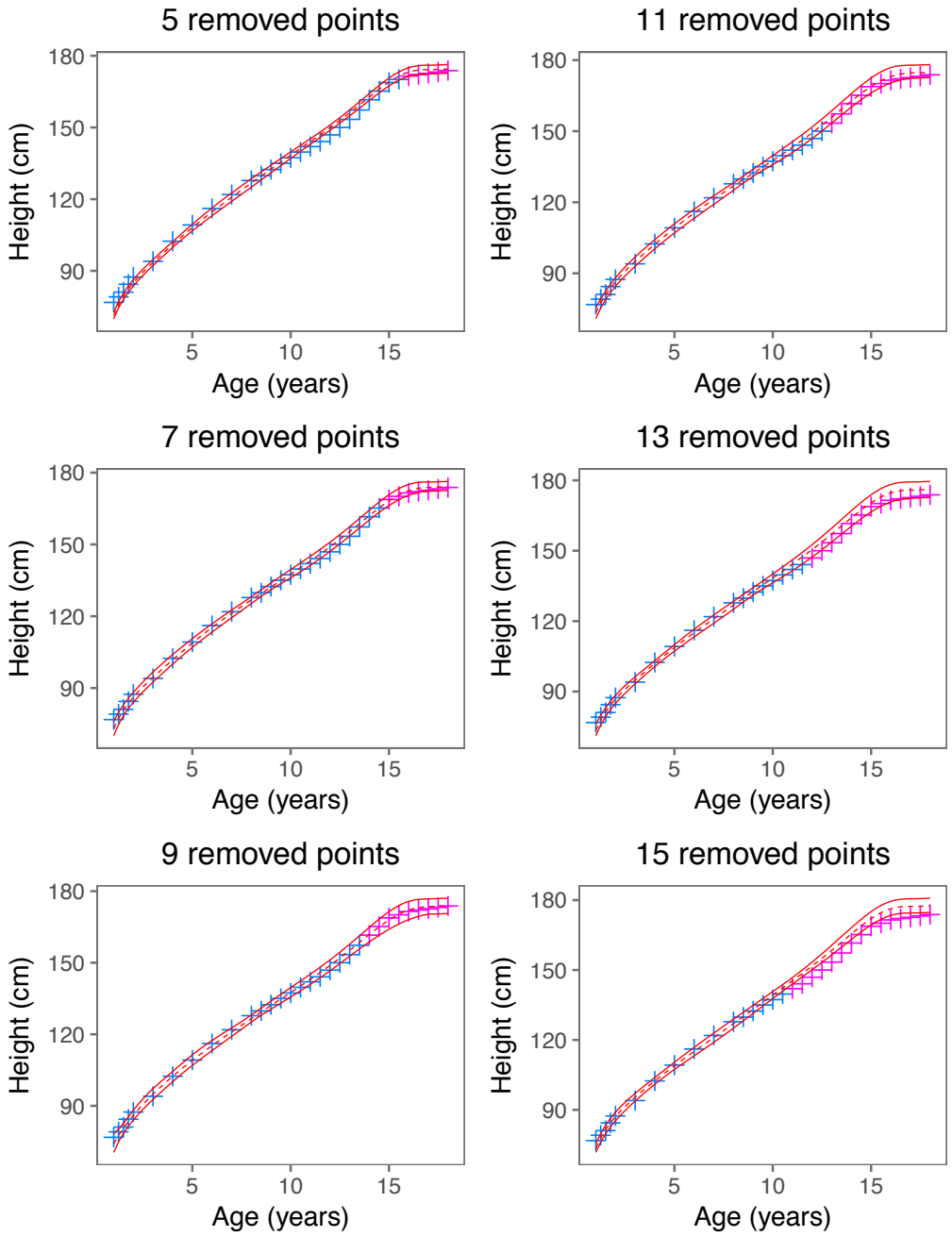


Figure 3.3: The posterior predictive mean curve and 95% prediction intervals for one individual in the data set, for a subset $(5, 7, 9, 11, 13, 15)$ of the varying number of data points removed from the end of their data set. The pink points correspond to the removed points.

clinical comparison of the growth of one young individual to the larger populace over a number of repeat visits to a clinician.

The lower right panel of Figure 3.3 does not display as credible intervals as wise as we would perhaps desire, which could be indicative of over-shrinkage. However, we will see in Section 3.3.3 that the model is capable of producing appropriately wide credible intervals in scenarios where available data is sparse. This re-enforces the fact that the appropriate amount of shrinkage is both data set *and* use case dependent, and is a decision that must be made in the modelling process. The prior for $\sigma^2_{\beta_1}$ most directly controls the magnitude of shrinkage, and users of this methodology should perform some form of sensitivity analysis for this prior, particularly if the data set of interest is relatively small.

3.3.2 Fixed distance coverage In this section we are again going to remove data from the end of the trajectories of individuals. However, instead of choosing one individual and varying the number of removed data points, we are going to fix the number of removed data points and do so for every individual. Here we chose to remove the last 15 data points, and iterate over the individuals one at a time, in a manner akin to cross-validation. For each iteration, we consider the mean and 95% prediction intervals for the individual with removed data, and we are interested in whether or not the removed data points are encompassed by (in- between the upper and lower limits of) the interval. If the points are, we consider them covered. Note that this process considers each removed point separately, and the average proportion of points that are covered is referred to as the model's *coverage*. We are interested in the coverage of the model as a function of the number of (potential) observations away from the last included data point. For example, if an individual had observations at $x = \{1, 2, 3, 4, 5\}$ and we removed the observations at $x = \{4, 5\}$, then the observation at $x = 4$ would be 1 observation away from the last included observation, and the observation at $x = 5$ would be 2 observations away from the last included observation.

Figure 3.4 displays a dip in coverage for points that are 4 to 9 observations away from the last included observation, which corresponds to ages between 12 and 15, typically when the largest growth spurt occurs. An important inferential

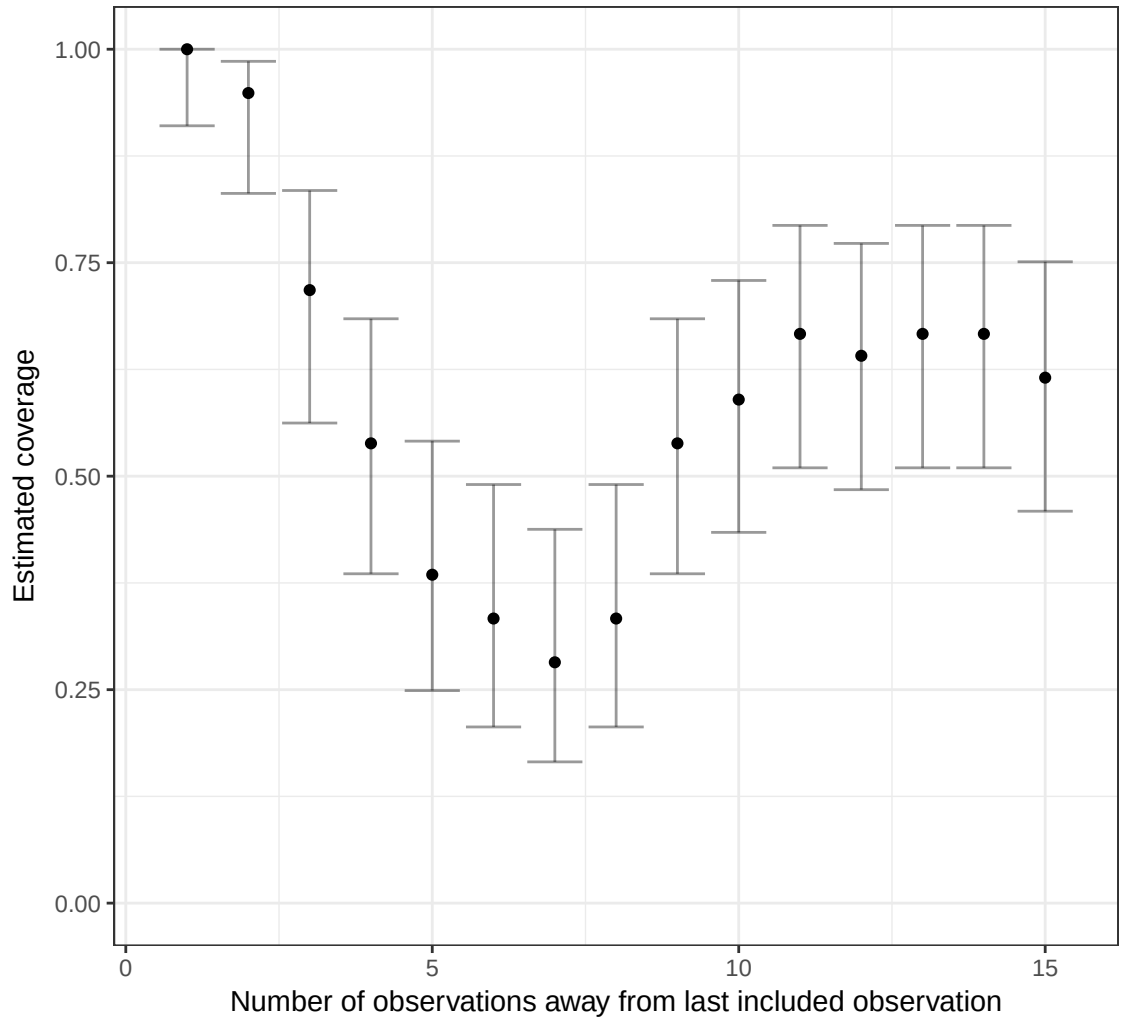


Figure 3.4: Coverage as a function of the distance (in terms of number of data points) between the x value of interest and the the last included x value. The credible intervals here are constructed using the Clopper-Pearson F quantiles, via the `binconf` function in the `Hmisc` package for R.

conclusion from Figure 3.4 is that at age 12 we are on average more able to predict the final height of the individual than when his growth spurt will occur. We also have, quite intuitively, more predictive capability for the two removed points closest to the last included observation. Thus, predicting in a one step ahead manner is most appropriate.

3.3.3 Interpolation in the presence of random missing data The previous exercises have dealt with small quantities of missing data, which was missing in a non-random manner. Here we explore the model's ability to borrow information across individuals to infer the likely shape of a growth curve, for data sets where a number of individuals possess considerable quantities of missing data, which is missing in a stochastic manner. We do this as it allows us to gain an understanding for how efficiently we can use information from a small number of complete, or even incomplete, individual curves to infer the shape of other individuals curves. To do so, we produce subsets of the `growth` data with an increasing proportion of missing data, and examine the coverage of the missing data points as a function of the quantity of missing data.

The method with which the data sets for this exercise are produced is nontrivial, and has three mutable parameters. We have a parameter that allows us to adjust the typical number of individuals in the data set who will have data removed, denoted p_{yn}. If we set $p_{\text{yn}} = 0.8$, then we would expect 80% of individuals in our data set to have some amount of missing data. We then choose the proportion of data points we would expect to be missing for these individuals, denoted p_{no}. Lastly we choose the maximum number of missing observations per person n_{mmis}, out of a possible 31 per person in this data set. These parameters are used to simulate values from the appropriate binomial, in the case ofp_{no} and n_{mmis}, and Bernoulli, for p_{yn}, distributions.

Specifically, we begin with the first individual in our full data set, and generate a sample from the appropriate Bernoulli distribution, here denoted A:

$$A \sim \text{Bernoulli}(p_{\text{yn}}). \tag{3.5}$$

If $A = 0$, then the first individual will have no missing data, and we move on to the next individual. If $A = 1$ we generate a sample from the appropriate binomial distribution, which selects the number of data points to remove, and we denote

such a sample by B:

$$B \sim \text{Binomial}(p_{\text{no}}, n_{\text{mmis}}). \tag{3.6}$$

Based on the specific value of B, we randomly select an initial data point to remove between 1 and $31 - B$, as each individual has 31 observations, and remove the initial data point and the $B - 1$ data points following it. The implementation of this subsetting regime can be found in the `R` function in Appendix D. This method for data removal is more likely to remove contiguous sections of data from the middle of individual's samples, which is intentional, as it best demonstrates the model's ability to pool information.

In this particular exercise we iterate over all possible combinations of the following values for our parameters:

$$p_{\text{yn}} \in \{0.6, 0.8, 0.9, 0.95, 1\}$$

$$p_{\text{no}} \in \{0.75, 0.85, 0.875, 0.9, 0.925, 0.95, 0.975\}$$

$$n_{\text{mmis}} \in \{25, 29\},$$

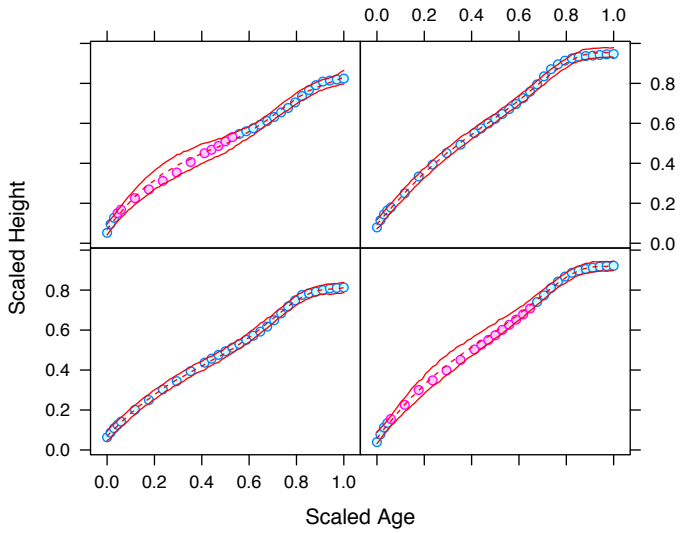
which allow us to examine the behaviour of the model for a wide variety of missing data scenarios. Particularly, it allows us to observe the model's behaviour when the quantity of missing data is extreme, and the nature of any undesirable behaviour. The way in which such behaviour becomes prevalent is called the model's *breakdown* behaviour, and specifically refers to any extremes in: sampler runtime, parameter estimate variability and deviations from nominal coverage. We are interested in how and where this (likely) breakdown occurs, as many applications have large segments of missing data, and we would like to ensure the model behaves as sensibly as possible in such use cases.

Before considering model performance breakdown however, we inspect the graph-

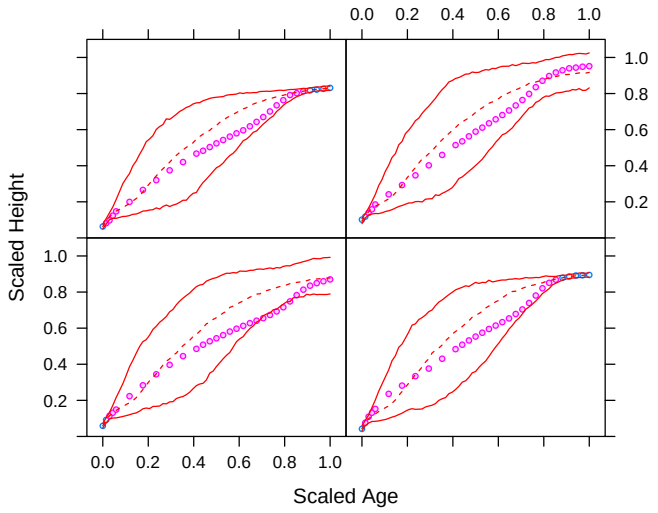
ical output of interest, namely the posterior predictive mean curves and prediction intervals for each individual. Figure 3.5 contains such an output for four individuals from the data set, in scenarios where all data points are present and where some data points are missing. Specifically, Figure 3.5a depicts a data set with a 'moderate' amount of missing data ($p_{\text{yn}} = 0.6$, $p_{\text{no}} = 0.85$ and $n_{\text{mmis}} = 25$). We see the posterior predictive intervals have widened slightly around the missing (pink) data points, appropriately representing the increased uncertainty about the fitted curve in those regions. Figure 3.5b also displays the widening of the prediction intervals, but in a much more extreme scenario ($p_{\text{yn}} = 1$, $p_{\text{no}} = 0.975$ and $n_{\text{mmis}} = 29$). We have retained the starting value for each individual, but not much more than that. A few individuals have retained values towards the end of their trajectories, which has caused a narrowing in the prediction intervals around $x \in [0.8, 1]$. Most interestingly, this narrowing has occurred for all individuals, not just those who retained observations in this region. This is an example of the borrowing of information across individuals in the presence of missing data.

The high level behaviour of coverage as a function of missing data is displayed in Figure 3.6, with the curved, dashed line and shaded region representing the anticipated behaviour. The aforementioned region is strictly *not* a credible region, merely an informed approximation, as simulating from the appropriate posterior for such a plot is computationally and, perhaps, methodologically infeasible. The behaviour of the model is (empirically) a fairly linear function of missing data until the extremes are reached. As such, the majority of simulation time was spent trying to identify the point at which model 'broke down', whereby the predictive coverage dropped off, or had considerably increased variability. The vertical, dotted line in Figure 3.6 represents the maximum possible quantity of missing data, as removing any more data corresponds to removing entire individuals from the data set, which would be inappropriate.

The data subsetting process introduces an extra level of variability in the observed coverage, as there are multiple paths for arriving at the same proportion of missing



(a) Posterior predictive mean curve and 95% prediction intervals for a data set with $p_{\text{yn}} = 0.6$, $p_{\text{no}} = 0.85$ and $n_{\text{mmis}} = 25$.



(b) Posterior predictive mean curve and 95% prediction intervals, for a data set with for a data set with $p_{\text{yn}} = 1$, $p_{\text{no}} = 0.975$ and $n_{\text{mmis}} = 29$.

Figure 3.5: An example of the behaviour of the posterior predictive distribution for a subset of four individuals, some with missing data, from the full data set. The pink data points are not known to the model.

data, as well as the various data subsets for a given set of simulation parameters. Some data points will also contain more information about the behaviour of the

population; data subsets that contain these high information points will have better coverage. This is a possible explanation for several points in Figure 3.6, where there is considerable variation in observed coverage for similar proportions of missing data. More generally, we would expect the model's coverage behaviour to be a function of the similarity between repeated measures. If individuals in a specific data set have very similar behaviour, then we would anticipate good interpolation behaviour. Conversely, we would anticipate the drop off to occur sooner and faster if individuals are quite distinct. Lastly, data sets with more missing data are associated with a dramatically increased run-times, as it is more difficult for `Stan` to explore the induced posterior distributions appropriately. As such, more time per sample is required, as the sampler tuning parameters that affect the time per sample have to be increased significantly.

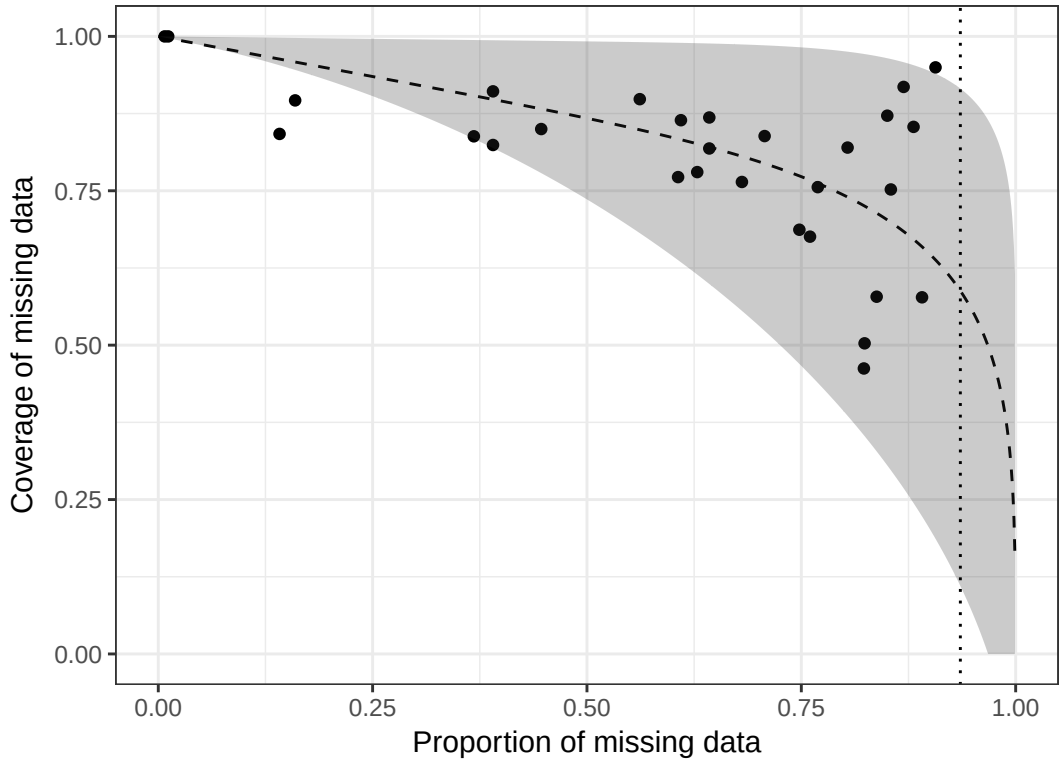


Figure 3.6: The proportion of covered missing points as a function of the proportion of missing points. The dotted line and shaded region indicate the approximate 'drop off' behaviour, and anticipated increase in variability of the coverage. Note that these are simply visual guides, not probabilistic quantities. The dotted vertical line at $x = 0.935$ corresponds to the maximum allowable quantity of data that could have been removed given the data subsetting process. The nominal coverage level was 95%.

3.4 Conclusion

In this chapter we have developed and implemented methodology for fitting monotonic polynomials with random effects in a Hierarchical Bayesian framework. To do so we have extended the mathematical structure presented in Chapter 2 to allow for subject-specific terms, and implemented one of the semi-compact cases of said structure in `Stan`. We demonstrated the ability of the model to provide population and individual level inference, and have shown how the hierarchical structure enables pooling of known information to appropriately interpolate missing data. Illustrations of pooling included predictions for a single individual with missing data in Section 3.3.1, as well as interpolation for many individuals with missing data in Section 3.3.3.

The model was shown to perform best in a one step ahead prediction setting, as opposed to five or ten steps ahead. It was also shown to perform well at interpolation in the presence of sizeable portions of missing data. However, the runtime required for acceptable inference grew considerably in the tested missing data scenarios.

The commentated example and `Stan` model code available at `https://github.com/hhau/mono-random-effects` should serve as a *walk-through* for any statistical practitioner who wishes to make use of the model developed here. Some possible extensions and other applications for this methodology will be detailed in Chapter 5.

Monotonic Polynomial Degree Selection using Reversible Jump

Selecting the appropriate polynomial degree is an important step in the process of modelling relationships present in data using polynomials. In a Bayesian framework, this involves fitting polynomials of various degrees to the data, and inspecting the fitted curves and credible intervals. Model selection criteria such as the Akaike information criterion (Akaike; 1973) or Bayesian information criterion (Schwarz; 1978) are often considered concurrently with the fitted polynomials to enable an appropriate selection. Alternatively, in unconstrained regression, one could construct an appropriate F-test to test the necessity of an additional polynomial term, and Hawkins (1994) suggests using such an F-test for monotonic polynomials, in combination with plots of the fitted curves, to perform model selection whilst imposing a monotonicity constraint. This approach can be informative despite the violation of the requisite regularity conditions of the F-test due to the monotonicity constraint, as the parameterisation used by Hawkins results in similar fitted curves for monotonic polynomials of increasing odd degree. Murray et al. (2016) also address monotonic polynomial degree selection, in the frequentist framework, with an 'm out of n' bootstrap methodology. There are a number of other selection criteria that employ a Bayesian methodology (see, for example, Abramovich and Grinshtein; 2013), however, to our knowledge, none address model selection in the presence of constraints that cannot be expressed as a fixed system of inequalities on the parameters, such as monotonicity (see Kato and Hoijtink 2006 for an example of model selection under inequality constraints). If prediction is the focus of the model, then predictive metrics such as the Bayesian predictive information criterion (Ando; 2007), or the coverage probabilities of withheld data points are also typically considered. An excellent summary of Bayesian methodology for predictive model selection can be found in Vehtari and Ojanen (2012).

However, all the aforementioned methods typically require fitting all the poly-

nomial degrees under consideration consecutively, or writing some sophisticated code to fit them all concurrently. The selection criteria each introduce a varying level of implicit bias to the selection process, as they all penalise different quantities in different ways, and hence will select different models in both the finite sample and asymptotic cases.

An alternative method for proportionally simulating from a family of possible models is given by the reversible jump Markov chain Monte Carlo (RJMCMC) sampling algorithm, first presented by Green (1995). The RJMCMC algorithm allows for the dimension of the parameter space to change between each iteration of the MCMC sampler. In the polynomial context this means the degree of polynomial can change between MCMC samples. The RJMCMC algorithm enables the selection of an appropriate model, as it will naturally produce more MCMC samples from the model with greatest posterior probability. The Bayesian specification of the family of allowable models also ensures contextual knowledge is made explicit in the form of the prior distribution over said model space. A correct implementation of a RJMCMC algorithm also has less implicit bias when used for model selection than the selection criteria previously discussed, as all non- model biases towards certain polynomial degrees should also be explicitly incorporated into the prior distribution.

In this chapter we develop an implementation of the reversible jump MCMC sampler first presented by Green (1995) and elaborated on in Hastie and Green (2012). Specifically, we use the implementation method presented in Chapter 3 of the *Handbook of Markov chain Monte Carlo* by Fan and Sisson (2011). We develop our own implementation, instead of building off `Stan` or `BUGS`, for a number of reasons. Firstly, `Stan` does not permit the use of discrete latent variables in models, as its sampler makes use of derivative based quantities to generate samples. This means we cannot have latent indicator variables in our model, which we would require if we were to use `Stan`. `BUGS` *has* been augmented by Lunn et al. (2008) to handle trans-dimensional models, however `BUGS` performed poorly with the non-

linear monotonic structure presented in Proposition 2.1.1, and allowing the model to change dimension is unlikely to improve this. In fact, regardless of the sampler, the non-linear structure does not readily permit changes in dimension. Hence, we make use of an orthonormal polynomial structure that more readily lends itself to changes of dimension, and enforce monotonicity via the prior distribution[1].

The details of our orthonormal polynomial structure are presented first, in Section 4.1. The specification of our RJMCMC sampler specifically for monotonic polynomials then follows in Section 4.2, with validation exercises thereafter in Section 4.3.

4.1 Orthonormal polynomials

Orthogonal polynomials have been utilized in RJMCMC schemes before (see Mallick (1998) for an example) due to the lack of correlation between the regression parameters. To construct an orthonormal polynomial, the QR decomposition detailed in Thisted (1988) is used, with an added reorthogonalisation step. Consider the usual monomial basis polynomial regression model:

$$\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon},$$

where the columns of $\boldsymbol{X}$ are the typical monomial basis, i.e. $\boldsymbol{X} = \left[\mathbf{1}, \boldsymbol{x}, \boldsymbol{x}^2, \ldots, \boldsymbol{x}^q\right]$. This is not an ideal basis to work with as the coefficients of $\boldsymbol{\beta}$ are highly correlated with each other, and removing one would cause a large change in the value of the remaining coefficients. Consider instead an orthonormal basis:

$$\boldsymbol{Y} = \boldsymbol{Q}\boldsymbol{\gamma} + \boldsymbol{\varepsilon}$$

where $\boldsymbol{X} = \boldsymbol{Q}\boldsymbol{R}$, $\boldsymbol{\beta} = \boldsymbol{R}^{-1}\boldsymbol{\gamma}$. The construction of $\boldsymbol{Q}$ and $\boldsymbol{R}$ is such that the columns of $\boldsymbol{Q}$ are orthonormal to each other, and $\boldsymbol{R}$ is an upper triangular matrix. This basis

[1] BUGS has no medium with which we could enforce monotonicity via the prior.

is ideal to work with as every time the dimension of the parameter space changes, the final column of the $\boldsymbol{Q}$ matrix is added/removed, and the final column and row of the $\boldsymbol{R}$ matrix is added/removed. The values of the remaining regression coefficients $\boldsymbol{\gamma}$ also do not change upon this addition/removal operation.

4.1.1 Orthonormal QR implementation Whilst base `R` (see R Core Team 2017) does include a QR decomposition routine, we instead use the routine presented in Thisted (1988) with an additional reorthogonalisation step to minimise the numerical effect adding/removing a coefficient might have. The recursive nature of this routine is also advantageous as it avoids computing high degree polynomial terms directly. This means a numerically unstable matrix, in either the $\boldsymbol{X}$ or $\boldsymbol{Q}$ bases, is never created, which allows the use of higher degree polynomial terms. The code is presented in Appendix F.

We also rescale both $\boldsymbol{x}$ and $\boldsymbol{y}$ to $[0,1]$ prior to computing our QR decomposition. This futher increases the numerical stablity of our method, and additionally makes the sampler easier to tune.

4.2 Implementation details

The RJMCMC sampler detailed here is an extension to the typical Metropolis-Hastings (MH) algorithm (see Hastings 1970 for the form most commonly presented) used to generate samples from the posterior of interest. The reversible jump aspect of the algorithm allows the dimension of the parameter space to change as part of each iteration. To do so, before generating a sample from the proposal distribution for the parameters, a proposal for the dimension of the parameter space must first be generated. This proposal is then generated, an augmented version of the typical MH acceptance probability is then calculated, and the proposal is accepted or rejected in the typical manner.

Notationally, in this section, the Markov chain is assumed to be at time point t and in degree/dimension d. The proposed values for the following time point

are indicated by an asterisk $(\boldsymbol{\theta}^*, d^*)$, and an apostrophe is used to indicate if the proposal exists in a different dimension to that of the previous value $(\boldsymbol{\theta}^{*'}, d^{*'})$. Note that $\boldsymbol{\theta}$ is a generic parameter vector that is assumed to contain all the parameters in that model, excluding the dimension of the current and proposed models.

4.2.1 Prior distributions

Regression coefficient prior The prior for the regression coefficients is flat, with an indicator function to ensure all parameter values correspond to a monotonic polynomial. Note that putting a flat prior on $\boldsymbol{\beta}$ is the same as putting a flat prior on $\boldsymbol{\gamma}$. As such it could be viewed as the Jeffreys prior. The way the monotonic indicator works is that it converts the $\boldsymbol{\gamma}$ coefficients back to $\boldsymbol{\beta}$ coefficients and checks the monotonicity of this set of coefficients using the `ismonotone` function from the `MonoPoly` package. This function works by calculating the roots of the polynomial, then checks to see if there are any real roots and what the multiplicity of these real roots is. If there are no real roots, or if all real roots have even multiplicity, then the polynomial is monotonic (on the subset of interest). Stated mathematically, the prior is:

$$\Pr(\boldsymbol{\beta}) \propto \mathbb{I}_{\{\boldsymbol{\beta} \text{ is monotonic}\}}. \tag{4.1}$$

Our preference for the flat prior in this scenario can be substituted, should the data set require a different prior for computational reasons, or we have sufficient knowledge to place to a non-flat prior on the polynomial coefficients. This prior is then multiplied by the monotonicity indicator, hence some care must be taken, as priors without sufficient mass in the monotonic region could result in the posterior being difficult to sample. It should be noted that such flat prior, and even weakly informative diffuse priors, can be overly influential in the calculation of posterior model probabilities. Care should be taken at all times, but especially in small data scenarios, where the impact of the prior is greater.

Variance prior Again, we prefer the flat prior in most cases, although here it is flat on the log scale:

$$\Pr(\sigma^2) \propto \frac{1}{\sigma^2}. \tag{4.2}$$

However, particularly in small data sets, a more informative prior is required. This is to avoid the model attributing all variation in the data to the variance parameter, which results in fitting a $d_{\min}$ degree polynomial to the data. This is easily accommodated in the code.

Polynomial degree prior We limit the support of d to be the inclusive set of integers between $d_{\min}$ and $d_{\max}$, and our prior must also be defined on this support. In practice we do this by generating a prior that appropriately represents our knowledge on $\{0,\ldots,d_{\max}\}$, ensuring the terms that are less than $d_{\min}$ do not have appreciable mass. We then remove the terms corresponding to degrees less than $d_{\min}$ and normalise the remaining terms. If we choose not to do this, then the prior is equivalent to a flat prior over the allowable dimensions, as the proposal distribution for d does not allow for values outside of $\{d_{\min},\ldots,d_{\max}\}$. A flat prior over the set of permissible polynomial degrees can be used. The `rjmonopoly` implementation also allows for a normalised binomial prior option, where the probability argument is used as a control parameter that allows us to express our knowledge about the likelihood of higher/lower polynomial degrees. This latter prior is preferable in situations where something is known about the likely polynomial degree, which is quite often the case.

4.2.2 Likelihood and posterior expression

Likelihood term As we are fundamentally performing only univariate polynomial regression, the likelihood term is relatively simple, and we can express it either in terms of the orthonormal, or monomial basis. We can write the distributional statements for a given polynomial degree as:

$$y \sim \mathrm{N}\left(\boldsymbol{Q\gamma}, \sigma^2 \mathcal{I}_d\right) \qquad y \sim \mathrm{N}\left(\boldsymbol{X\beta}, \sigma^2 \mathcal{I}_d\right) \tag{4.3}$$

and the likelihoods as:

$$\mathrm{L}(\boldsymbol{\theta}, d \mid \boldsymbol{Y}) = \mathrm{L}\left(d, \boldsymbol{\beta}, \sigma^2 \mid \boldsymbol{Y}\right) = \left(\sqrt{2\pi\sigma^2}\right)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{X\beta})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{X\beta})\right\} \tag{4.4}$$

$$\mathrm{L}(\boldsymbol{\theta}, d \mid \boldsymbol{Y}) = \mathrm{L}\left(d, \boldsymbol{\gamma}, \sigma^2 \mid \boldsymbol{Y}\right) = \left(\sqrt{2\pi\sigma^2}\right)^{-n/2} \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{Q\gamma})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{Q\gamma})\right\} \tag{4.5}$$

Posterior expression Finally, we can combine our likelihood with the prior distributions defined in Section 4.2.1 to create the expression to which the posterior is proportional to:

$$\begin{aligned}
\Pr(\boldsymbol{\theta}, d \mid \boldsymbol{Y}) \quad &\propto \quad \mathrm{L}(\boldsymbol{\theta}, d \mid \boldsymbol{Y}) \Pr(\boldsymbol{\theta}, d) \\
\Pr(d, \boldsymbol{\gamma}, \sigma^2 \mid \boldsymbol{Y}) \quad &\propto \quad \mathrm{L}\left(d, \boldsymbol{\gamma}, \sigma^2 \mid \boldsymbol{Y}\right) \Pr(d) \Pr(\boldsymbol{\gamma}) \Pr(\sigma^2) \\
\Pr(d, \boldsymbol{\gamma}, \sigma^2 \mid \boldsymbol{Y}) &\propto \left(\frac{1}{\sqrt{2\pi\sigma^2}}\right)^{n/2} \cdot \exp\left\{-\frac{1}{2\sigma^2}(\boldsymbol{y} - \boldsymbol{Q\gamma})^{\mathrm{T}}(\boldsymbol{y} - \boldsymbol{Q\gamma})\right\} \cdot \\
&\qquad \mathbb{I}_{\{\boldsymbol{\beta} \text{ is monotonic}\}} \cdot \frac{1}{\sigma^2}
\end{aligned}$$

4.2.3 Sampler Specification; Proposal distributions

Dimension proposal We specify upper and lower bounds for the proposal distribution of the degree of the polynomial, based on the prior values for $d_{\min}$ and $d_{\max}$. This is to ensure that the Markov chain in degree space is neither initialised in an inadmissible region (according to the prior), nor proposes polynomial degrees with zero prior mass. For example, we should never propose polynomials of degree smaller than $d_{\min}$. We also make use of the prior upper bound the polynomial degree. These upper and lower bounds are again denoted by $d_{\max}$ and $d_{\min}$. The existence of a $d_{\max}$ also allows for straightforward generation of initial values, which will be discussed later in Section 4.2.3.

When the Markov chain is in either the maximum or minimum dimension space, the proposal for d^* is a discrete uniform distribution on that current dimension and the dimension immediately lower or higher than it respectively. When not in the maximum or minimum dimension states, the proposal is a discrete uniform distribution on its current dimension, and the dimensions immediately above and below it. This distribution is best explained by the code used to generate proposals from it, which is shown in the following code snippet.

```
1     dimProposer <- function(d_current, d_min, d_max) {
2       if (d_current < d_min | d_current > d_max) {
3        stop("current dimension is outside of allowed bounds")
4       }
5
6       if (d_current == d_min) {
7         res <- sample(x = c(d_current, d_current + 1), size = 1)
8
9       } else if (d_current == d_max) {
10        res <- sample(x = c(d_current, d_current - 1), size = 1)
11
12      } else {
13        res <- sample(x = c(d_current - 1, d_current, d_current + 1),
14                      size = 1)
15      }
16
17      return(res)
18    }
```

This simple proposal distribution admits a straightforward term to the proposal ratio in the acceptance probability expression. It is either 1 if the previous degree and proposed degree are not $d_{\max}$ or $d_{\min}$, or it is $\frac{3}{2}$ or $\frac{2}{3}$ if either the proposed or previous degree are.

Regression coefficient proposal We generate our proposals for the regression coefficients in the orthonormal space, that is we generate them for $\boldsymbol{\gamma}$, as specified in Section 4.1. This is advantageous as all the coefficients are orthogonal to each

other, and as such a random walk with diagonal covariance matrix can be used to produce acceptable proposals. In the monomial space this would not be true, as the covariance structure for all $\boldsymbol{\beta}$ coefficients is hard to determine, and would also change considerably each time the degree of the polynomial changed.

When the proposed dimension is the same as that of the previous dimension, generating the proposal for the regression coefficients is straightforward:

$$\boldsymbol{\gamma}^{*} \sim \mathrm{N}\left(\boldsymbol{\gamma}^{[t-1]}, \sigma^2_{\gamma,\text{innov}}\mathcal{I}_d\right), \tag{4.6}$$

where $\sigma^2_{\gamma,\text{innov}}$ is the innovation variance associated with the proposal distribution of $\boldsymbol{\gamma}$, and $\mathcal{I}_d$ is the $d \times d$ identity matrix.

When the proposed dimension does not match the current dimension, things are more difficult. Consider a proposed dimension one greater that the previous dimension, $d^{*'} = d+1$. We can use a random walk proposal for the first d components of $\boldsymbol{\gamma}^{*'}$, however the $d^{*'}$th component does not exist in $\boldsymbol{\gamma}^{[t-1]}$. As such we need a different proposal distribution for this specific coefficient. It would be preferable to use a random walk with mean equal to the value of the last time we were in the $d^{*'}$ space:

$$\gamma^{*'}_{d^{*'}} \sim \mathrm{N}\left(\gamma^{[s]}_{d^{*'}}, \sigma^2_{\gamma,\text{innov}}\right),$$

where s is the time point where the chain was last in state $d^{*'}$. However, as this depends upon a time point prior to $t-1$ it constructs a sampling scheme that is no longer Markovian, and as such breaks the detailed balance required for the Metropolis-Hastings algorithm. As a result, we use an independent proposal for $\gamma^{*'}_{d^{*'}}$, with a very particular mean to ensure it proposes in the right region of space. To obtain this mean we first fit a polynomial of degree $d_{\max}$ using either the `Stan` implementation discussed in Chapter 2, or using the package `MonoPoly` by Turlach and Murray (2016). The estimates for $\boldsymbol{\beta}$ are then converted to their corresponding

orthonormal coefficients $\boldsymbol{\gamma}$ for use as means of independent proposal distributions. Mathematically:

$$\boldsymbol{\gamma}^{*'} \sim \mathrm{N}\left(\left[\boldsymbol{\gamma}^{[t-1]}, \mu_{\gamma,d^{*'}}\right]^T, \sigma^2_{\gamma,\mathrm{innov}}\mathcal{I}_{d^{*'}}\right), \tag{4.7}$$

where $\mu_{\gamma,d^{*'}}$ is the $d^{*'}$th component of the aforementioned initial $d_{\max}$ fit. There is an acceptance probability implication of this choice, as the independently proposed term no longer cancels in the ratio of proposal distributions. This will be discussed in more detail in Section 4.2.4.

The requirement for this proposal to be monotonic is expressed as part of our prior, and will be discussed in 4.2.1.

Variance proposal We use a log-normal random walk to propose values for σ^{2^*}:

$$\sigma^{2^*} \sim \mathrm{Lognormal}\left(\mathrm{Log}(\sigma^{2^{[t-1]}}), \sigma^2_{\sigma^2,\mathrm{innov}}\right). \tag{4.8}$$

as we need the value of σ^{2^*} to remain positive. An advantage with this proposal is to avoid the need to compute the Jacobian that would be induced by placing a random walk on $\log(\sigma^2)$. The downside is that the log-normal distribution is not symmetric, and thus does not cancel out in the proposal ratio of the acceptance probability.

4.2.4 Acceptance probability The general form of our acceptance probability is a combination of the standard Metropolis Hastings algorithm and the necessary reversible jump terms, and is presented in Fan and Sisson (2011). Note that $\mathrm{Q}(\cdot)$ here represents the proposal distribution for the quantity enclosed within it. We use $\alpha^*(\cdot)$ to denote the acceptance quantity prior to the application of the $\min\{1,\cdot\}$ operation, after which we denote the produced acceptance probability with $\alpha(\cdot)$.

$$\alpha^*\left(\left(d^{*'},\boldsymbol{\theta}^{*'}\right),\left(d,\boldsymbol{\theta}^{[t-1]}\right)\right) = \frac{\mathrm{L}(\boldsymbol{\theta}^{*'}\mid \boldsymbol{X})}{\mathrm{L}(\boldsymbol{\theta}^{[t-1]}\mid \boldsymbol{X})}\cdot\frac{\Pr(\boldsymbol{\theta}^{*'})}{\Pr(\boldsymbol{\theta}^{[t-1]})}\cdot\frac{\mathrm{Q}(\boldsymbol{\theta}^{[t-1]}\mid \boldsymbol{\theta}^{*'})}{\mathrm{Q}(\boldsymbol{\theta}^{*'}\mid \boldsymbol{\theta}^{[t-1]})}\cdot \frac{\mathrm{Q}(d^{*'}\rightarrow d)}{\mathrm{Q}(d\rightarrow d^{*'})}\cdot\frac{1}{\mathrm{Q}_{d\rightarrow d^{*'}}(u)}\cdot\left|\frac{\partial g_{d\rightarrow d^{*'}}(\boldsymbol{\theta}^*_{1:d},u)}{\partial(\boldsymbol{\theta}^*_{1:d},u)}\right| \tag{4.9}$$

$$\alpha\left(\left(d^{*'},\boldsymbol{\theta}^{*'}\right),\left(d,\boldsymbol{\theta}^{[t-1]}\right)\right) = \min\left\{1,\ \alpha^*\left(\left(d^{*'},\boldsymbol{\theta}^{*'}\right),\left(d,\boldsymbol{\theta}^{[t-1]}\right)\right)\right\} \tag{4.10}$$

Consider the previous state of the Markov chain $(d,\boldsymbol{\theta}^{[t-1]})$ with a proposed degree and state of $(d^{*'},\boldsymbol{\theta}^{*'})$ where $d^{*'} = d+1$, in line with our proposal distribution for d^*. Such a proposal would contain a proposed value for the $(d+1)^{\text{th}}$ coefficient of $\boldsymbol{\gamma}$, which we can think of as the extra random realisation needed to make the number of random objects *match* the proposed dimension. This realisation is denoted u in the acceptance probability. In this manner, it contributes to the acceptance probability not only in the Metropolis Hastings proposal term, the 3rd term in Expression 4.9, but also in the $\mathrm{Q}_{d\rightarrow d^{*'}}(u)$ term. We also require a specific definition for the function that maps the combination of the previous state and u to the proposed state:

$$g_{d\rightarrow d^{*'}}(\boldsymbol{\theta}^{[t-1]},u) = \begin{bmatrix}\boldsymbol{\theta}^{[t-1]}\\ 0\end{bmatrix} + \begin{bmatrix}\mathbf{0}\\ u\end{bmatrix}, \tag{4.11}$$

which is an ideal mapping function as it has a determinant of 1. This also enables us to perform both Metropolis Hastings updates and dimension jumps simultaneously, as we can feed in the proposed values for the first d coefficients, as well as u, to determine the acceptance probability of a move from our current position and dimension in parameter space, to another position and dimension. This is why the 6th and final term in the acceptance probability is written as a function of the first d components of our proposal *and* the component corresponding to u. This will decrease our typical acceptance probability, but will also decrease the number of likelihood evaluations needed per iteration. This is preferable in this situation,

as evaluating the likelihood to calculate the acceptance probability is the most computationally expensive action we undertake.

The remaining terms fall out fairly naturally now, as the term which concerns the dimension proposal is typically 1, and only differs when we are either in, or propose to be in, $d_{\max}$ or $d_{\min}$. The parameter proposal term (3$^{\text{rd}}$ term) contains only the lognormal σ^2 proposal and the independent proposal for u. The prior term and likelihood term are evaluated with respect to the distributions discussed in previous sections. For implementation details, please see the package source available at `https://github.com/hhau/rjmonopoly`.

Moves from d to $d^{*'} = d - 1$ are accepted with probability:

$$\alpha\left(\left(d^{*'}, \boldsymbol{\theta}^{*'}\right), \left(d, \boldsymbol{\theta}^{[t-1]}\right)\right) = \min\left\{1, \alpha^*\left(\left(d, \boldsymbol{\theta}^{[t-1]}\right), \left(d^{*'}, \boldsymbol{\theta}^{*'}\right)\right)^{-1}\right\}. \tag{4.12}$$

Note the swap in the order of the arguments to α^*. Calculating the acceptance probability in this manner is a very explicit use of the detailed balance assumption, as it tells us that the probability of moving from d to $d-1$ is the same as the inverse of the move from $d-1$ to d i.e. $\Pr(d \to d-1) = \Pr(d-1 \to d)^{-1}$. However, we must again consider the monotonic constraint, as non monotonic proposals will have $\alpha^*(\cdot) = 0$, which would lead to $\alpha(\cdot) = 1$, unless we check our monotonicity indicator after the inversion.

4.3 Model validation

This section contains several validation exercises, which were performed to test the model and implementation to ensure that known quantities could be appropriately estimated. It also explores the relationship between the posterior distribution of the polynomial degree and data size and variance. It should be noted that polynomials have few defining features as their degree increases. Namely, either the number of turning/inflection points increase, or the duration at which the polynomial appears 'flat' increases. As we are concerned with monotonicity, polynomials

with turning points inside the region of interest are inappropriate objects of study. Instead we focus on cases with large 'flat' regions, such as in Example 4.3.1, or with a distinct number of inflection points, such as in the Example 4.3.2.

4.3.1 The relationship between noise and estimated degree We first quantify the relationship between the quantity of noise added, and the estimated polynomial degree. This exercise is performed with $1,000$ x values between 0 and 1, and uses the polynomial $\mathrm{p}(x) = 2 + (2x-1)^{11} + 1.5x$ with a varying amount of noise added. The motivation for the additional linear term is to move the true polynomial sufficiently far away from the boundary of monotonic space, so as to avoid the issues discussed Chapter 2.3, as well as other numerical inefficiencies.

From Figure 4.1 we see that as we decrease the magnitude of the added noise, the polynomial degree increases towards the true polynomial degree. This is to be expected, as adding noise sufficiently blurs the location of the start and end of the 'flat' sections of the true polynomial, and as such it can be appropriately modelled by a lower degree polynomial. We can also see the posterior distribution appropriately favours odd degree polynomials, particularly in Figure 4.1d, where degree 9 and 7 are preferable to degree 8. This is again unsurprising given the fundamentally different asymptotic behaviour in even and odd degree polynomials, and the manner in which that affects the polynomial's ability to fit the data.

When fitting polynomials to comparatively noise free data, such as in Figures 4.1g and 4.1h, the choice of tuning parameters, specifically the innovation variances, becomes paramount. This is because the minimal noise added to the data manifests itself as not only a relative certainty of a small estimate for σ^2, but also in the relatively precise nature in which we know $\boldsymbol{\gamma}$. As such we need to considerably reduce the innovation variance for $\boldsymbol{\gamma}$, and not just the innovation variance for σ^2 as one might intuitively expect. This insight is relatively straightforward upon reflection, but is noted here for posterity.

We can compare the polynomial degree estimates above to the estimates obtained

using the 'm out of n' bootstrap methodology presented in Murray et al. (2016). This works by drawing m samples from the data with replacement, fitting all permissible polynomial degrees to the sampled data, and using the non sampled data as a predictive validation set. This constitutes one bootstrap sample, with the sampler selecting the polynomial degree with the lowest prediction error in each bootstrap sample. The output presented in Figure 4.2 was produced using 100 iterations for each m value. Reassuringly, the results from this are broadly similar to that of the reversible jump sampler. There are minor differences for a given level of noise, where the 'm out of n' bootstrap appears to favour slightly higher degree polynomials. For example, in Figure 4.2c, the bootstrap suggests a polynomial of at least degree 10, where as the comparable reversible jump result in Figure 4.1f is very confident in degree 9. This may be due to the difference in parameterisations, as the bootstrap methodology makes use of the parameterisation discussed in Chapter 2. However this is a very minor difference, as one would not expect the posterior distribution to perfectly match the sampling distribution induced by a mean square error based prediction metric.

The only considerable difference in the two methodologies is in the runtime, where the reversible jump sampler has a definite edge. Each of the images in Figure 4.2 took approximately 10 minutes to generate, using 100 bootstrap iterations for each m. Each row in Figure 4.1 was produced in approximately 20 seconds on the same hardware using the reversible jump sampler, using a chain length of $50,000$. This is not surprising, as the bootstrap has to fit all possible polynomial degrees at each iteration, whereas the reversible jump sampler only ends up sampling degrees according to the posterior distribution of the polynomial degree. As such, it can avoid fitting high degree polynomials in data sets where they are not required.

4.3.2 The relationship between the number of data points and estimated degree To explore this relationship, we assume the underlying polynomial is the following:

$$\mathrm{p}(x) = 12 + \frac{1}{500}\left(-225.24x^3 + 89.48x^4 - 13.72x^5 + 0.93x^6 - 0.02x^7\right), \qquad (4.13)$$

which subsequently has normally distributed noise, with zero mean and standard deviation 0.5, added to it. We generate a sample for x using values for x that range between 0 and 12. As opposed to having a large 'flat' section like the previous example, this monotonic polynomial has a specific number of inflection points. The method with which we derive the polynomial in Equation (4.13) is of note and consists of the following steps: We begin by specifying the second derivative of the polynomial via its roots

$$\mathrm{p}''(x) = (-x)(x - 2.4)(x - 5.7)(x - 8.9)(x - 11.1),$$

which we then expand into a monomial basis form

$$\mathrm{p}''(x) = -1351.45x + 1073.80x^2 - 274.47x^3 + 28.1x^4 - x^5,$$

and then integrate twice to arrive at the final monomial basis coefficients,

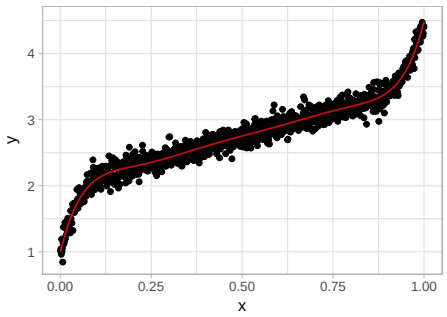
$$\mathrm{p}'(x) = -675.72x^2 + 357.93x^3 - 68.18x^4 + 5.62x^4 - 1.67x^6,$$
$$\mathrm{p}(x) = -225.24x^3 + 89.48x^4 - 13.72x^5 + 0.93x^6 - 0.02x^7.$$

Finally, the polynomial was rescaled and shifted for use as a comparative model in dental application detailed in Murray et al. (2016), which also has the added benefit of rendering the region of interest on a more natural scale.

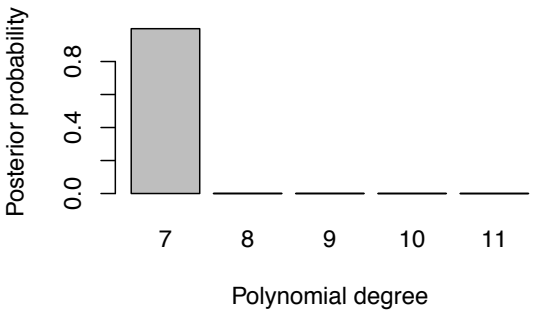
Repeated X values We first consider an example with a fixed number of possible x values, with either 3 or 9 measurements at each of the 13 distinct integer values. The posterior of the polynomial degree and the fit using the posterior mode are

displayed in Figure 4.3. The trend is as expected, in that our ability to estimate the appropriate degree, and our corresponding certainty of said estimate, increases as the number of data points per x value increases. Figure 4.3d is a good example of typical sampler behaviour in the presence of a comparatively 'flat' posterior for the polynomial degree, and is it reassuring to see the sampler is capable of exploring a wide variety of possible polynomial degrees under fixed values for the tuning parameters.

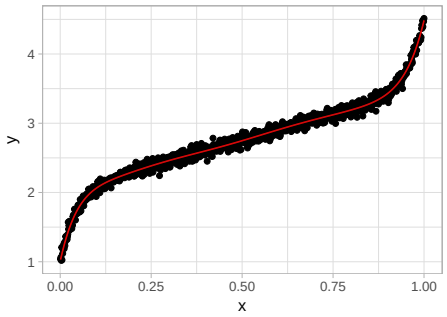
Unique X values In this exercise we consider the same number of data points as in the previous example, but distributed across unique x values instead. Figure 4.4 contains the corresponding output, and the pattern is unsurprisingly similar. Of interest in this exercise, is the apparent bimodality of the posterior distribution for the polynomial degree visible in Figure 4.4d. This is also evident in the traceplot where we can see the singular chain alternating between the lower polynomial degrees $(2, 3, 4)$ and the higher degrees $(5, 6, 7)$ in a very distinct manner. This demonstrates the sampler's ability to 'mode switch', albeit in a very correlated manner. If such behaviour is observed in a real-world scenario, the sampler should be run for considerably longer, in order to ensure the posterior modes have the appropriate number of samples drawn from them.



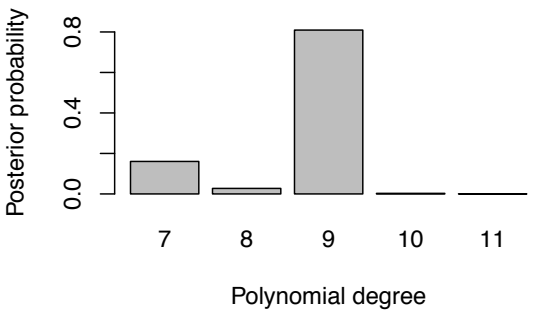
(a) Fit for $\sigma = 0.1$



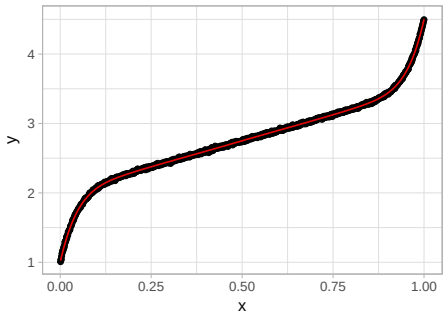
(b) Degree posterior distribution for $\sigma = 0.1$



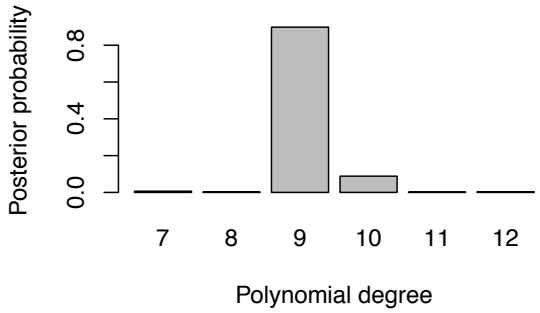
(c) Fit for $\sigma = 0.05$



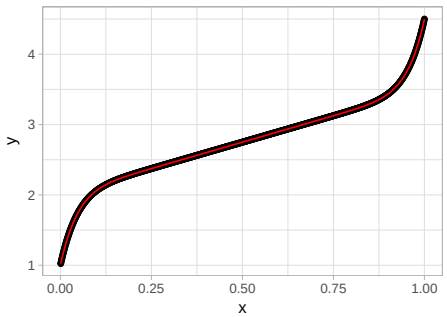
(d) Degree posterior distribution for $\sigma = 0.05$



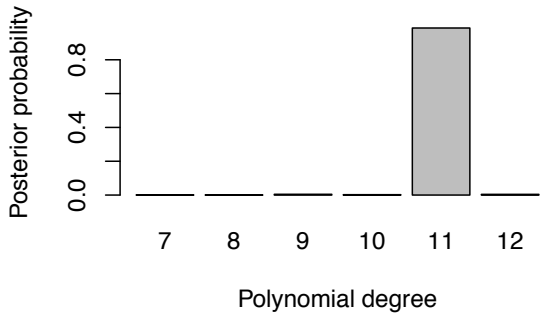
(e) Fit for $\sigma = 0.01$



(f) Degree posterior distribution for $\sigma = 0.01$

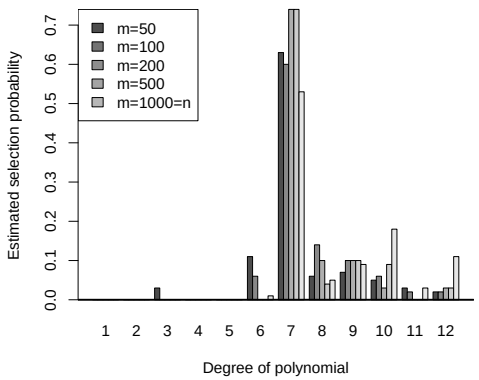


(g) Fit for $\sigma = 0.0001$

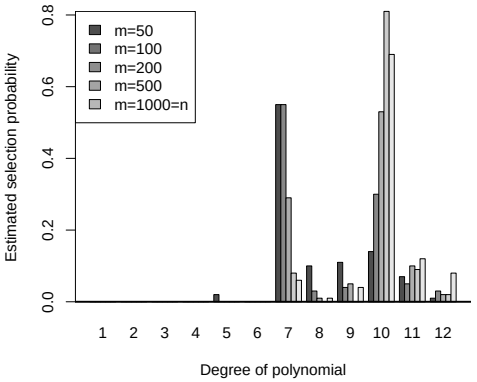


(h) Degree posterior distribution for $\sigma = 0.0001$

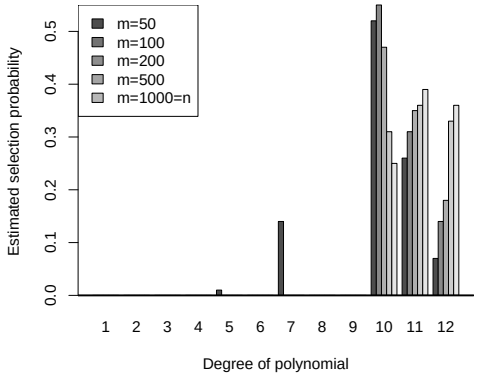
Figure 4.1: Simulation results using the reversible jump sampler, where the true polynomial is $\mathrm{p}(x) = 2 + (2x-1)^{11} + 1.5x$, with $\mathrm{N}\left(0, \sigma^2\right)$ noise.



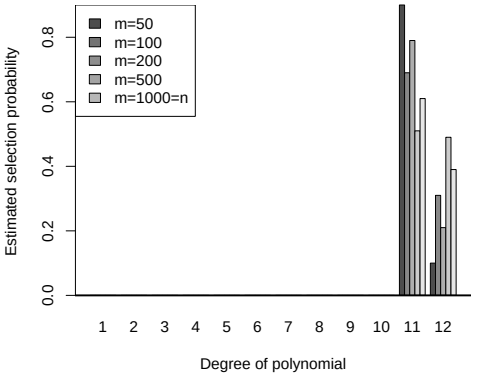
(a) Probability of degree selection for $\sigma = 0.1$



(b) Probability of degree selection for $\sigma = 0.05$

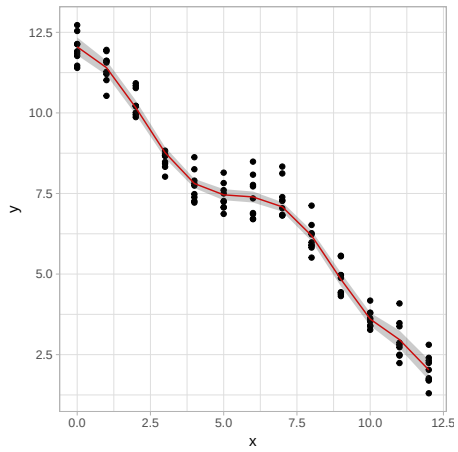


(c) Probability of degree selection for $\sigma = 0.01$

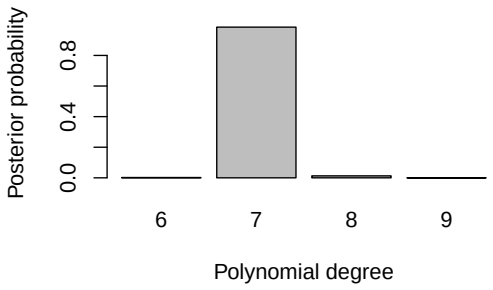


(d) Probability of degree selection for $\sigma = 0.0001$

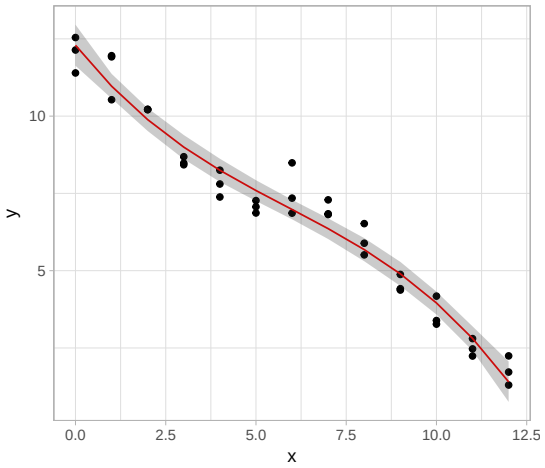
Figure 4.2: Simulation results using the 'm out of n' bootstrap methodology, where the true polynomial is $\mathrm{p}(x) = 2 + (2x-1)^{11} + 1.5x$, with $\mathrm{N}\left(0, \sigma^2\right)$ noise.



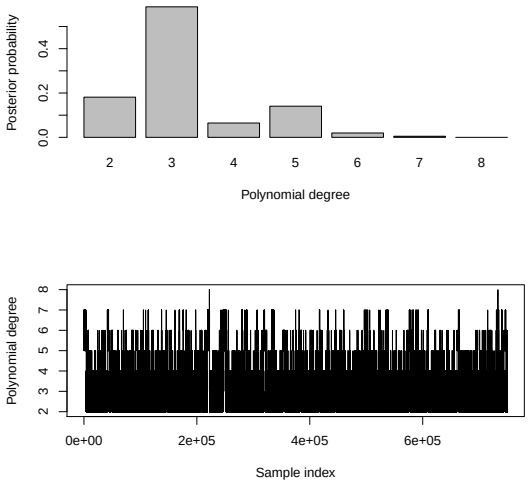
(a) Fit for 9 points per x value



(b) Degree posterior distribution for 9 points per x value

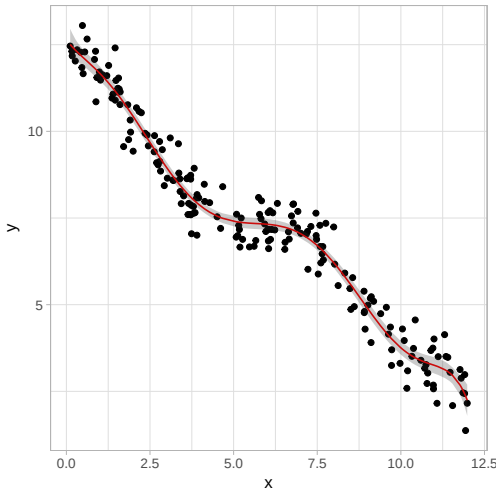


(c) Fit for 3 points per x value

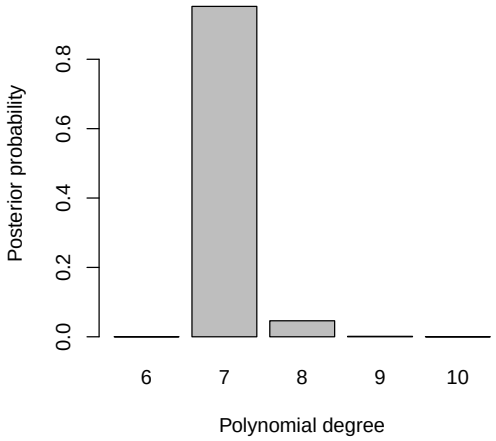


(d) Degree posterior distribution and trace plot (for initial 750,000 posterior samples) for 3 points per x value

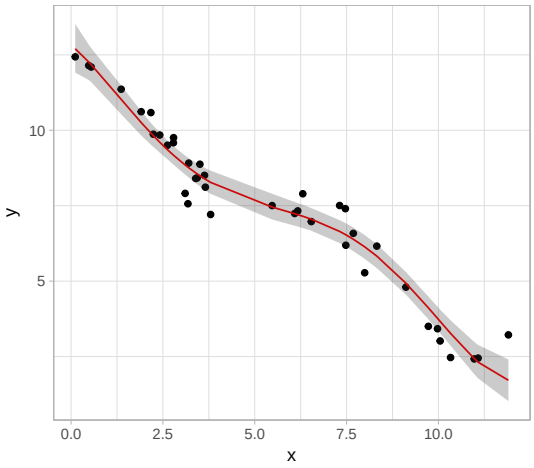
Figure 4.3: Simulation results for a varying number of data points per *non-unique* x value for the polynomial $\mathrm{p}(x) = 12 + \frac{1}{500}\left(-225.24x^3 + 89.48x^4 - 13.72x^5 + 0.93x^6 - 0.02x^7\right)$.



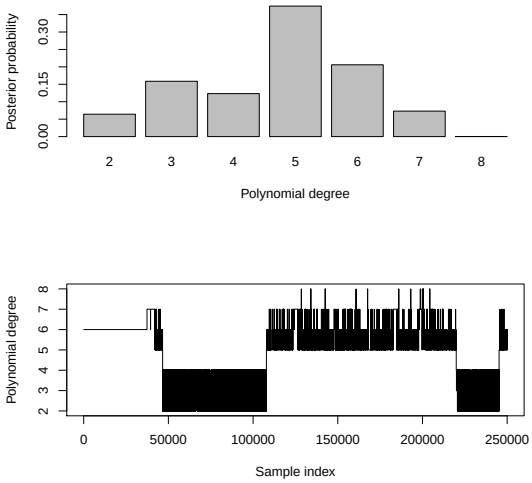
(a) Fit for 195 unique x values



(b) Degree posterior distribution for 195 unique x values



(c) Fit for 39 unique x values



(d) Degree posterior distribution and traceplot (for initial 250,000 samples) for 39 unique x values

Figure 4.4: Simulation results for a varying number of data points per *unique* x value $\mathrm{p}(x) = 12 + \frac{1}{500}\left(-225.24x^3 + 89.48x^4 - 13.72x^5 + 0.93x^6 - 0.02x^7\right)$

4.4 Inference at the data set level

In this chapter we have predominately focused on data sets consisting of one individual, that is, one set of $(\boldsymbol{X}, \boldsymbol{Y})$'s. However, in Chapter 3, the data set of interest consisted of a number of individuals. The manner in which we should apply our method for model selection to such a data set is not obvious. We could naively combine all individuals to form one large pair of $(\boldsymbol{X}, \boldsymbol{Y})$'s, however this typically results in the selection of a polynomial degree that is too low, as it is fundamentally incorrect to combine subject-specific measures in this manner. Instead we can rely on the speed of our implementation, and the functional nature of our R package, to quickly estimate the appropriate degree for each individual, which can then be used to select the appropriate degree for the model in Chapter 3.

We include the following section, which consists of descriptions around code snippets, to demonstrate the relative ease with which we can switch from single individuals to whole data sets. It also serves as an illustravtive example for future users. The fact that the code is available in an R package makes this task considerably easier, and we only have to be familiar with the apply family of functions within base R. Consider the following snippet:

```
1  library(rjmonopoly); library(fda)
2  library(ggplot2); library(gridExtra); library(knitr)
3
4  func_applicator <- function(y, x = growth$age, ...) {
5      res <- rjmonopoly::rjmonopoly(x = x, y = y,
6                 d_min = 5, d_max = 12, # tuning params
7                 prior_prob = 0.5,
8                 starting_var_val = 5e-5,
9                 control = list(
10                  innov_sd_var = 0.001,
11                  innov_sd_beta = 0.001
12                )
13              )
14     return(res)
```

```
15  }
16
17  y_mat <- growth$hgtm
18  res_list <- apply(y_mat, 2, function(X) func_applicator(y = X))
```

Here, we have applied our RJMCMC sampler to every male individual in the *growth* data set of the `fda` package. We can then quickly perform inference at the level of the whole data set. Note that we have adjusted some of the samplers tuning parameters in order to produce samples with acceptable diagnostics including, but not limited to, traceplots and acceptance rates. In the following snippet, we thin the output and inspect the fitted curves for every individual[2].

```
1  res_list <- lapply(res_list, function(X) {
2      rjmonopoly::thin(X, thin_period = 50, warm_up = 500)
3    })
4
5  fit_plots <- lapply(res_list, rjmonopoly::plotFit)
6  gridExtra::marrangeGrob(fit_plots, ncol = 4, nrow = 10)
```

We can also quickly inspect the polynomial degree posterior for each individual:

```
1  lapply(res_list, function(x) {
2    rjmonopoly::plotDegreePost(x, only = "barplot")
3  })
```

Although this is not always ideal, as ~ 40 barplots might be somewhat visually overwhelming. It does however, enable the quick visual identification of any individuals in our data set whose degree posterior is distinct. Alternatively, we could take one step back and consider the empirical distribution of the degree selected for each individual. Here we use the 99% quantile as our selection rule, so that we do not unnecessarily restrict the flexibilty of any models we may choose to fit as a result.

```
1  selected_degrees <- unlist(lapply(res_list, function(X) {
```

[2]The plots produced from the following snippets are not presented here, as fixed width & height, noninteractive conglomerates of subplots are not ideal for the printed format.

```
2     quantile(X$d_samples, 0.99)
3   }))
4   knitr::kable(as.data.frame(table(selected_degrees)))
```

Selected Degree	Frequency	Probability
6	1	0.03
7	4	0.10
8	26	0.67
9	8	0.21

Table 4.1: The selected polynomial degrees from all males in the *growth* data set of the fda package.

The output of Table 4.1 would probably lead us to select a degree of 9 for our model in Chapter 3, as a substaintial number of individuals in the dataset have a selected degree of 9, and we lose some model flexibility to the hierarchical structure.

4.5 Concluding remarks

In this chapter we have formulated and implemented a valid reversible jump sampler for monotonic polynomials of varying degree. The sampler's ability to recover the truth given a sufficient number of data points or a maximum quantity of noise were the validation metrics of interest. Whilst ideally we would be able to validate the implementation of the sampler in a manner akin to Cook et al. (2006), such methodology for models of varying dimension does not yet exist[3]. Methods to algorithmically validate MCMC implementations are a matter of ongoing research.

The reversible jump sampler has a speed advantage over the '*m* out of *n*' bootstrap methodology. The implementation of the sampler as an R package should hopefully

[3]The methodology described in Cook, Gelman and Rubin (2006) is broadly correct, however it does contain a small error detailed at http://andrewgelman.com/2017/08/10/bigshot-statistician-keeps-publishing-papers-errors-anything-can-get-stop/. A corrigendum is anticipated to appear shortly.

enable any end user to easily apply the methodology to their data set of interest, without having to concern themselves as to its inner workings. The `R` package is available at `https://github.com/hhau/rjmonopoly`.

Conclusion

In this thesis we developed novel Bayesian methodology, for both basic and random effect style monotonic polynomial models, and addressed the topic of model selection for monotonic polynomials in a Bayesian framework. In order to achieve this, the parameterisation of Murray et al. (2016) was implemented in the popular probabilistic programming languages BUGS and Stan. This parameterisation was extended to include subject specific terms, and was tested in various missing data scenarios. The large number of parameters in the subject specific model necessitated an implementation in Stan. Model selection in a Bayesian monotonic polynomial context was addressed by implementing a reversible jump Markov chain Monte Carlo (RJMCMC) sampler, which allows for quick and direct inference of the appropriate polynomial degree.

The implementation of our methodology in BUGS and Stan had numerous benefits associated with it, primarily pertaining to practitioner ease of use. By implementing our methodology in popular probabilistic languages it is immediately accessible to statisticians familiar with said languages. There are also numerous statistical interfaces and post-processing libraries surrounding these languages, which allow the applied statistician to focus almost entirely on their specific application and not the idiosyncrasies of fitting the chosen model to the data. Additionally, a powerful sampling technique, such as the Hamiltonian Monte Carlo algorithm within Stan, is practically a necessity when using models with subject specific parameters. The large number of highly correlated posterior distributions are seldom sampled appropriately by, more traditional, random walk Metropolis Hastings style methods. However, we encountered the limitations of these languages when implementing our preferred form of model selection, the reversible jump Markov chain Monte Carlo sampler. We addressed this by building our own sampler in R, which is now available as an R package entitled rjmonopoly. This format is also popular amongst applied researchers, but does forfeit the automatic

tuning procedures within the probabilistic languages.

We demonstrated our methodologies with various data sets and for several use cases, including common missing data scenarios. Validation exercises for the random effect model had a particular focus on the appropriate borrowing of information across individuals in the presence of missing data. These exercises were largely successful, but did highlight potential areas for future improvements. Some difficulty in reliably estimating functions on, or close to, the boundary of monotonic space was observed in Chapter 2, and we also encountered some structural limitations, due to the non uniqueness of the parameterisation, when attempting to perform some high level, generative types of population level inference in Chapter 3. Solutions to these issues are potential topics for future research in this area, and will be detailed in the following section. Many of the possible extensions would be best explored in applications that have yet to be optimally modelled, and we hope our consideration of practitioner accessibility aids in the discovery thereof.

5.1 Possible future work

As we demonstrated in Chapter 2, a challenging problem in monotonic polynomial model specification is that of defining suitable priors that address the problems with the calibration of credible intervals if the underlying polynomial lies on, or close to, the boundary of monotonic space. In the frequentist framework, Murray et al. (2016) proposed an ad-hoc way of correcting the coverage problem of the bootstrap confidence intervals. However, a Bayesian framework necessitates a more principled solution. It would also be relatively easy to replace the normal distribution for the error term, (specified in Equation (2.1)), with a t-distribution. The shape parameter of the the t-distribution could then have an appropriate prior specified, and posterior samples could be drawn as part of the MCMC procedure. Such methodology would render the fitted regression curve less sensitive to outliers, however there would need to be sufficient motivation from the data set to

adopt such a methodology.

The model described in Chapter 3 possesses the ability to perform generative inference at the same level as the collected data, by considering the joint posterior predictive distribution of all individuals. However, higher level generative inference was somewhat impeded by identifiability issues. We were unable to generate new individuals from the hierarchical random effect distribution, and hence unable to generate a new series of observations from said new individual. A possible solution is to impose a positivity constraint on the distribution of one subject specific term from one individual. However, initial testing suggests that it make not be sufficient to render the model identifiable. Substantial post processing of the MCMC samples is an option, but would be difficult as the label switching like behaviour is itself quite variable. Future work could investigate various identifiability measures, such as soft centring priors or alternative hierarchical parameterisations with sum to zero constraints.

A lofty goal for future work would be to combine an adapted version of the hierarchical structure developed in Chapter 3, with a more sophisticated reversible jump sampler. The resulting sampler should be capable of providing inference about two important quantities: the appropriate number of random effects, and the appropriate polynomial degree. Ideally the sampler would allow both quantities to vary simultaneously. This would be an incredibly useful piece of methodology, as it could 'automatically' fit the optimal model. However the development would be very challenging, as our current hierarchical random effects model does not easily allow for changes in dimension.

Most MCMC samplers require some form of 'tuning', whereby the parameters of the proposal distributions must be adjusted for each application, and our RJMCMC sampler is no exception. By rescaling the data to $[0, 1]$, using an orthonormal basis, and providing sensible defaults, we reduced the amount of testing time spent on sampler tuning. However, it would further improve practitioner accessibility if the tunable proposal parameters were coupled with some form of adaptive method-

ology, to remove the need for sampler tuning entirely. Elements of the adaptive Metropolis Hastings methodology of Haario et al. (2001) could be incorporated. However, care would need to be taken as the change of model dimension may lead to poor adaptive behaviour.

Finally, note that we have been brief in our exposition of the possible extensions, and have undoubtedly concealed numerous difficulties. A thorough exploration of each of these ideas would be required to address them suitably. However, we believe such future work would be valuable, and would compliment the ideas and achievements of this thesis.

Bibliography

Abramovich, F. and Grinshtein, V. (2013). Model selection in regression under structural constraints, *Electronic Journal of Statistics* **7**: 480–498.
URL: *https://doi.org/10.1214/13-EJS780*

Akaike, H. (1973). Information theory and an extension of the maximum likelihood principle, *Second International Symposium on Information Theory (Tsahkadsor, 1971)* pp. 267–281.

Ando, T. (2007). Bayesian predictive information criterion for the evaluation of hierarchical Bayesian and empirical Bayes models, *Biometrika* **94**(2): 443–458.
URL: *http://dx.doi.org/10.1093/biomet/asm017*

Barbara, L., Benzi, G., Gaiani, S., Fusconi, F., Zironi, G., Siringo, S., Rigamonti, A., Barbara, C., Grigioni, W., Mazziotti, A. and Bolondi, L. (1992). Natural history of small untreated hepatocellular carcinoma in cirrhosis: A multivariate analysis of prognostic factors of tumor growth rate and patient survival, *Hepatology* **16**(1): 132–137.
URL: *http://dx.doi.org/10.1002/hep.1840160122*

Barlow, R. E., Bartholomew, D. J., Bremner, J. M. and Brunk, H. D. (1972). *Statistical Inference under Order Restrictions*, John Wiley & Sons, London.

Barlow, R. E. and Brunk, H. D. (1972). The isotonic regression problem and its dual, *Journal of the American Statistical Association* **67**(337): 140–147.
URL: *http://dx.doi.org/10.2307/2284712*

Bon, J. J., Murray, K. and Turlach, B. A. (2017). Fitting monotone polynomials in mixed effects models, *Statistics and Computing* .
URL: *https://doi.org/10.1007/s11222-017-9797-8*

Brezger, A. and Steiner, W. J. (2008). Monotonic regression based on Bayesian p splines, *Journal of Business & Economic Statistics* **26**(1): 90–104.
URL: *https://doi.org/10.1198/073500107000000223*

Brickman, L. and Steinberg, L. (1962). On nonnegative polynomials, *The American Mathematical Monthly* **69**(3): 218–221.
URL: *http://www.jstor.org/stable/2311058*

Carpenter, B., Gelman, A., Hoffman, M. et al. (2017). Stan: A probabilistic programming language, *Journal of Statistical Software* **76**(1): 1–32.
URL: *http://dx.doi.org/10.18637/jss.v076.i01*

Cole, T. J., Freeman, J. V. and Preece, M. A. (1998). British 1990 growth reference centiles for weight, height, body mass index and head circumference fitted by maximum penalized likelihood, *Statistics in Medicine* **17**(4): 407–429.
URL: *http://dx.doi.org/10.1002/(SICI)1097-0258(19980228)17:4<407::AID-SIM742>3.0.CO;2-L*

Cook, S. R., Gelman, A. and Rubin, D. B. (2006). Validation of software for Bayesian models using posterior quantiles, *Journal of Computational and Graphical Statistics* **15**(3): 675–692.
URL: *https://doi.org/10.1198/106186006X136976*

Curtis, S. M. (2008). *Variable Selection Methods with Applications to Shape Restricted Regression*, PhD thesis, Department of Statistics, North Carolina State University.
URL: *https://repository.lib.ncsu.edu/handle/1840.16/4481*

Dette, H. and Studden, W. J. (1997). *The Theory of Canonical Moments with Applications in Statistics, Probability, and Analysis*, Wiley Series in Probability and Statistics, Wiley & Sons, New York.

Dierckx, P. (1980). Algorithm/algorithmus 42 an algorithm for cubic spline fitting with convexity constraints, *Computing* **24**(4): 349–371.
URL: *https://dx.doi.org/10.1007/BF02237820*

Elphinstone, C. D. (1983). A target distribution model for non-parametric density estimation, *Communications in Statistics - Theory and Methods* **12**(2): 161–198.
URL: *https://doi.org/10.1080/03610928308828450*

Fan, Y. and Sisson, S. A. (2011). *in* S. Brooks, A. Gelman, G. Jones and X. Meng (eds), *Handbook of Markov Chain Monte Carlo*, Chapman & Hall/CRC Handbooks of Modern Statistical Methods, CRC Press, chapter 3, pp. 67–91.
URL: *http://dx.doi.org/10.1201/b10905*

Fausett, L. V. (2003). *Numerical Methods: Algorithms and Applications*, Prentice Hall, Upper Saddle River, NJ.

Firmin, L., Müller, S. and Rösler, K. M. (2011). A method to measure the distribution of latencies of motor evoked potentials in man, *Clinical Neurophysiology* **122**(1): 176–182.
URL: *http://dx.doi.org/10.1016/j.clinph.2010.05.034*

Firmin, L., Müller, S. and Rösler, K. M. (2012). The latency distribution of motor evoked potentials in patients with multiple sclerosis, *Clinical Neurophysiology* **123**(12): 2414–2421.
URL: *http://dx.do.org/10.1016/j.clinph.2012.05.008*

Friedman, J. H. and Tibshirani, R. (1984). The monotone smoothing of scatterplots, *Technometrics* **26**(3): 243–250.
URL: *http://dx.doi.org/10.2307/1267550*

Gelfand, A. E., Smith, A. F. M. and Lee, T.-M. (1992). Bayesian analysis of constrained parameter and truncated data problems using Gibbs sampling, *Journal of the American Statistical Association* **87**(418): 523–532.
URL: *http://www.jstor.org/stable/2290286*

Geweke, J. (1986). Exact inference in the inequality constrained normal linear regression model, *Journal of Applied Econometrics* **1**(2): 127–141.
URL: *http://www.jstor.org/stable/2096611*

Golchi, S. and Campbell, D. A. (2014). Sequentially Constrained Monte Carlo, *ArXiv e-prints* .
URL: *https://arxiv.org/abs/1410.8209*

Green, P. J. (1995). Reversible jump Markov chain Monte Carlo computation and Bayesian model determination, *Biometrika* **82**(4): 711.
URL: *http://dx.doi.org/10.1093/biomet/82.4.711*

Haario, H., Saksman, E. and Tamminen, J. (2001). An adaptive metropolis algorithm, *Bernoulli* **7**(2): 223–242.
URL: *http://www.jstor.org/stable/3318737*

Hastie, D. I. and Green, P. J. (2012). Model choice using reversible jump Markov chain Monte Carlo, *Statistica Neerlandica* **66**(3): 309–338.
URL: *http://dx.doi.org/10.1111/j.1467-9574.2012.00516.x*

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* **57**(1): 97–109.
URL: *http://dx.doi.org10.2307/2334940*

Hawkins, D. M. (1994). Fitting monotonic polynomials to data, *Computational Statistics* **9**(3): 233–247.

Hazelton, M. L. and Turlach, B. A. (2011). Semiparametric regression with shape constrained penalized splines, *Computational Statistics & Data Analysis* **55**(10): 2871–2879.
URL: *http://dx.doi.org/10.1016/j.csda.2011.04.018*

Heinzmann, D. (2008). A filtered polynomial approach to density estimation, *Computational Statistics* **23**(3): 343–360.
URL: *https://doi.org/10.1007/s00180-007-0070-z*

Henningsen, A. and Henning, C. H. C. A. (2009). Imposing regional monotonicity on translog stochastic production frontiers with a simple three-step procedure, *Journal of Productivity Analysis* **32**(3): 217.
URL: *https://doi.org/10.1007/s11123-009-0142-x*

Hermann, S., Ickstadt, K. and Müller, C. H. (2016). Bayesian prediction of crack growth based on a hierarchical diffusion model, *Applied Stochastic Models in*

Business and Industry **32**(4): 494–510. asmb.2175.
URL: *http://dx.doi.org/10.1002/asmb.2175*

Hoeting, J. A., Madigan, D., Raftery, A. E. and Volinsky, C. T. (1999). Bayesian model averaging: A tutorial, *Statistical Science* **14**(4): 382–401.
URL: *http://www.jstor.org/stable/2676803*

Holmes, C. C. and Heard, N. A. (2003). Generalized monotonic regression using random change points, *Statistics in Medicine* **22**(4): 623–638.
URL: *http://dx.doi.org/10.1002/sim.1306*

Hwang, H. and Takane, Y. (2005). An extended multivariate random-effects growth curve model, *Behaviormetrika* **32**(2): 141–153.
URL: *https://dx.doi.org/10.2333/bhmk.32.141*

Karlin, S. and Studden, W. J. (1966). *Tchebycheff Systems: With Applications in Analysis and Statistics*, Wiley & Sons, New York.

Kass, R. E. and Raftery, A. E. (1995). Bayes factors, *Journal of the American Statistical Association* **90**(430): 773–795.
URL: *http://www.jstor.org/stable/2291091*

Kato, B. S. and Hoijtink, H. (2006). A Bayesian approach to inequality constrained linear mixed models: estimation and model selection, *Statistical Modelling* **6**(3): 231–249.
URL: *https://doi.org/10.1191/1471082X06st119oa*

LeBauer, D., Dietze, M. and Bolker, B. (2013). Translating probability distributions: From R to BUGS and back again, *The R Journal* **5**(1): 207–210.
URL: *http://journal.r-project.org/archive/2013-1/lebauer-dietze-bolker.pdf*

Lin, L. and Dunson, D. B. (2013). Bayesian Monotone Regression using Gaussian Process Projection, *ArXiv e-prints* .
URL: *https://arxiv.org/abs/1306.4041*

Lunn, D. J., Best, N. and Whittaker, J. C. (2008). Generic reversible jump mcmc

using graphical models, *Statistics and Computing* **19**(4): 395.
URL: *https://doi.org/10.1007/s11222-008-9100-0*

Lunn, D. J., Spiegelhalter, D., Thomas, A. and Best, N. (2009). The BUGS project: Evolution, critique and future directions (with discussion), *Statistics in Medicine* **28**(25): 3049–3082.
URL: *https://doi.org/10.1002/sim.3680*

Lunn, D. J., Thomas, A., Best, N. and Spiegelhalter, D. (2000). WinBUGS – a Bayesian modelling framework: concepts, structure, and extensibility, *Statistics and Computing* **10**(4): 325–337.

Lunn, D., Jackson, C., Best, N., Thomas, A. and Spiegelhalter, D. (2012). *The BUGS Book: A Practical Introduction to Bayesian Analysis*, Texts in Statistical Science, Chapman & Hall/CRC, Boca Raton.

Mallick, B. (1998). Bayesian curve estimation by polynomial of random order, *Journal of Statistical Planning and Inference* **70**(1): 91 – 109.
URL: *http://dx.doi.org/10.1016/S0378-3758(97)00179-1*

Manderson, A. A., Cripps, E., Murray, K. and Turlach, B. A. (2017). Monotone polynomials using BUGS and Stan, *Australian & New Zealand Journal of Statistics* **59**(4): 353–370.
URL: *http://dx.doi.org/10.1111/anzs.12207*

Marshall, M. (2008). *Positive Polynomials and Sums of Squares*, Vol. 146 of *Mathematical Surveys and Monographs*, American Mathematical Society.

Millner, A. (2010). Modeling lithium ion battery degradation in electric vehicles, *2010 IEEE Conference on Innovative Technologies for an Efficient and Reliable Electricity Supply*, pp. 349–356.

Murray, K. (2015). *Improved Monotone Polynomial Fitting with Applications and Variable Selection*, PhD thesis, School of Mathematics and Statistics, University

of Sydney.
URL: *http://hdl.handle.net/2123/13633*

Murray, K., Müller, S. and Turlach, B. A. (2013). Revisiting fitting monotone polynomials to data, *Computational Statistics* **28**(5): 1989–2005.
URL: *https://doi.org/10.1007/s00180-012-0390-5*

Murray, K., Müller, S. and Turlach, B. A. (2016). Fast and flexible methods for monotone polynomial fitting, *Journal of Statistical Computation and Simulation* **86**(15): 2946–2966.
URL: *https://doi.org/10.1080/00949655.2016.1139582*

Neelon, B. and Dunson, D. B. (2004). Bayesian isotonic regression and trend analysis, *Biometrics* **60**(2): 398–406.
URL: *http://www.jstor.org/stable/3695767*

Nummi, T. (1997). Estimation in a random effects growth curve model, *Journal of Applied Statistics* **24**(2): 157–168.
URL: *https://doi.org/10.1080/02664769723774*

O'Donnell, C. J. and Coelli, T. J. (2005). A Bayesian approach to imposing curvature on distance functions, *Journal of Econometrics* **126**(2): 493 – 523. Current developments in productivity and efficiency measurement.
URL: *http://www.sciencedirect.com/science/article/pii/S0304407604001216*

Ohlssen, D. and Racine, A. (2015). A flexible Bayesian approach for modeling monotonic dose–response relationships in drug development trials, *Journal of Biopharmaceutical Statistics* **25**(1): 137–156.
URL: *https://doi.org/10.1080/10543406.2014.919931*

Plummer, M. (2016). *rjags: Bayesian Graphical Models using MCMC*. R package version 4-6.
URL: *https://CRAN.R-project.org/package=rjags*

R Core Team (2017). *R: A Language and Environment for Statistical Computing*, R

Foundation for Statistical Computing, Vienna, Austria.
URL: *https://www.R-project.org/*

Raftery, A. E. (1996). Approximate Bayes factors and accounting for model uncertainty in generalised linear models, *Biometrika* **83**(2): 251–266.
URL: *http://dx.doi.org/10.1093/biomet/83.2.251*

Ramsay, J. O. (1998). Estimating smooth monotone functions, *Journal of the Royal Statistical Society, Series B* **60**(2): 365–375.

Ramsay, J. O., Wickham, H., Graves, S. and Hooker, G. (2014). *fda: Functional Data Analysis*. R package version 2.4.4.
URL: *https://CRAN.R-project.org/package=fda*

Saarela, O. and Arjas, E. (2011). A method for Bayesian monotonic multiple regression, *Scandinavian Journal of Statistics* **38**(3): 499–513.
URL: *http://dx.doi.org/10.1111/j.1467-9469.2010.00716.x*

Schwarz, G. (1978). Estimating the dimension of a model, *The Annals of Statistics* **6**(2): 461–464.
URL: *http://www.jstor.org/stable/2958889*

Shively, T. S., Sager, T. W. and Walker, S. G. (2009). A Bayesian approach to non-parametric monotone function estimation, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **71**(1): 159–175.
URL: *http://dx.doi.org/10.1111/j.1467-9868.2008.00677.x*

Spiegelhalter, D. J., Best, N. G., Carlin, B. P. and Van Der Linde, A. (2002). Bayesian measures of model complexity and fit, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* **64**(4): 583–639.
URL: *http://dx.doi.org/10.1111/1467-9868.00353*

Stan Development Team (2017). *Stan Modeling Language: User's Guide and Reference Manual*. Stan Version 2.17.0.
URL: *http://mc-stan.org/documentation/*

Sturtz, S., Ligges, U. and Gelman, A. (2005). R2WinBUGS: A package for running WinBUGS from R, *Journal of Statistical Software* **12**(1): 1–16.
URL: *https://dx.doi.org/10.18637/jss.v012.i03*

Terrell, D. (1996). Incorporating monotonicity and concavity conditions in flexible functional forms, *Journal of Applied Econometrics* **11**(2): 179–194.
URL: *http://dx.doi.org/10.1002/(SICI)1099-1255(199603)11:2<179::AID-JAE389>3.0.CO;2-G*

Thisted, R. A. (1988). *Elements of Statistical Computing: Numerical Computation*, Taylor & Francis.

Thomas, A., O'Hara, B., Ligges, U. and Sturtz, S. (2006). Making BUGS open, *R News* **6**(1): 12–17.

Tuddenham, R. and Snyder, M. (1954). *Physical Growth of California Boys and Girls from Birth to Eighteen Years*, Publications in child development, University of California Press.

Turlach, B. A. and Murray, K. (2016). *MonoPoly: Functions to Fit Monotone Polynomials*. R package version 0.3-8.
URL: *http://cloud.r-project.org/package=MonoPoly*

Utreras, F. I. (1985). Smoothing noisy data under monotonicity constraints: Existence, characterization and convergence rates, *Numerische Mathematik* **47**: 611–625.
URL: *https://doi.org/10.1007/BF01389460*

Vehtari, A. and Ojanen, J. (2012). A survey of Bayesian predictive methods for model assessment, selection and comparison, *Statist. Surv.* **6**: 142–228.
URL: *https://doi.org/10.1214/12-SS102*

Wang, X. and Berger, J. O. (2016). Estimating shape constrained functions using Gaussian processes, *SIAM/ASA Journal on Uncertainty Quantification* **4**(1): 1–25.
URL: *https://doi.org/10.1137/140955033*

Yan, J. and Prates, M. (2013). *rbugs: Fusing R and OpenBugs and Beyond*. R package version 0.5-9.

URL: *http://cloud.r-project.org/package=rbugs*

BUGS and Stan models for Chapter 2

A.1 BUGS implementation

A.1.1 Monotone on the real line

```
model{
  for(i in 1:N){
    y[i] ~ dnorm(mu[i], tauy)

    horner[i,1] ← beta[q+1]
    for(k in 1:q){
      horner[i,k+1] ← horner[i,k]*x[i] + beta[q+1-k]
    }
    mu[i] ← horner[i,q+1]
  }

  for(i in 1:Nnew){
    hnew[i,1] ← beta[q+1]
    for(k in 1:q){
      hnew[i,k+1] ← hnew[i,k]*xnew[i] + beta[q+1-k]
    }
    mupred[i] ← hnew[i,q+1]
    ypred[i] ← mupred[i] + eps[i]
    eps[i] ~ dnorm(0, tauy)
  }

```

```
22  for(j in 1:q){
23    beta[j+1] ← alpha * gamma[j]/j
24  }
25  beta[1] ← beta0
26
27  for(j in 1:q){
28    gamma[j] ← g1[j] + g2[j]
29  }
30
31
32  for(j in 1:q){
33    g1[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
34                   bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                         ,1])
35    g2[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
36                   bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                         ,2])
37  }
38
39  for(j in 1:(K+1)){
40    for(k in 1:2){
41      bnrev[j,k] ← bn[Kp1-j+1,k]
42    }
43  }
44
45  beta0 ~ dnorm(0, 0.001)
46  for(j in 1:(K+1)){
47    for(k in 1:2){
48      bn[j,k] ~ dnorm(0, 0.001)
49    }
```

```
50    }
51    tauy ~ dgamma(0.01, 0.01)
52    sigy <- 1/sqrt(tauy)
53
54    Kp1 <- K+1
55  }
```

A.1.2 Even degree polynomial monotone on $[a, \infty)$

```
1  model{
2    for(i in 1:N){
3      y[i] ~ dnorm(mu[i], tauy)
4
5      horner[i,1] <- beta[q+1]
6      for(k in 1:q){
7        horner[i,k+1] <- horner[i,k]*x[i] + beta[q+1-k]
8      }
9      mu[i] <- horner[i,q+1]
10   }
11
12   for(i in 1:Nnew){
13     hnew[i,1] <- beta[q+1]
14     for(k in 1:q){
15       hnew[i,k+1] <- hnew[i,k]*xnew[i] + beta[q+1-k]
16     }
17     mupred[i] <- hnew[i,q+1]
18     ypred[i] <- mupred[i] + eps[i]
```

```
19      eps[i] ~ dnorm(0, tauy)
20    }
21
22    for(j in 1:q){
23      beta[j+1] <- alpha*gamma[j]/j
24    }
25    beta[1] <- beta0
26
27    for(j in 2:q){
28      gamma[j] <- g1[j] + g2[j-1] - a*g2[j]
29    }
30    gamma[1] <- g1[1] - a * g2[1]
31
32    for(j in 1:q){
33      g1[j] <- inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
34                      bnrev[(max(0,Kp1-j)+1):(min(q+1-j,K)+1)
                            ,1])
35      g2[j] <- inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
36                      bnrev[(max(0,Kp1-j)+1):(min(q+1-j,K)+1)
                            ,2])
37    }
38
39    for(j in 1:(K+1)){
40      for(k in 1:2){
41        bnrev[j,k] <- bn[Kp1-j+1,k]
42      }
43    }
44
45
46    bn[K+1,1] ~ dbern(0)
```

```
47    bn[K+1,2] ~ dbern(0)
48    beta0 ~ dnorm(0, 0.001)
49    for(j in 1:K){
50      for(k in 1:2){
51        bn[j,k] ~ dnorm(0, 0.001)
52      }
53    }
54    tauy ~ dgamma(0.01, 0.01)
55    sigy <- 1/sqrt(tauy)
56
57    Kp1 <- K+1
58  }
```

A.1.3 Odd degree polynomial monotone on $[a, \infty)$

```
1   model{
2     for(i in 1:N){
3       y[i] ~ dnorm(mu[i], tauy)
4
5       horner[i,1] <- beta[q+1]
6       for(k in 1:q){
7         horner[i,k+1] <- horner[i,k]*x[i] + beta[q+1-k]
8       }
9       mu[i] <- horner[i,q+1]
10    }
11
12    for(i in 1:Nnew){
```

```
13      hnew[i,1] ← beta[q+1]
14      for(k in 1:q){
15        hnew[i,k+1] ← hnew[i,k]*xnew[i] + beta[q+1-k]
16      }
17      mupred[i] ← hnew[i,q+1]
18      ypred[i] ← mupred[i] + eps[i]
19      eps[i] ~ dnorm(0, tauy)
20    }
21
22    for(j in 1:q){
23      beta[j+1] ← alpha*gamma[j]/j
24    }
25    beta[1] ← beta0
26
27    for(j in 2:q){
28      gamma[j] ← g1[j] + g2[j-1] - a*g2[j]
29    }
30    gamma[1] ← g1[1] - a * g2[1]
31
32    for(j in 1:q){
33      g1[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
34                     bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                           ,1])
35      g2[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
36                     bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                           ,2])
37    }
38
39    for(j in 1:(K+1)){
40      for(k in 1:2){
```

```
41          bnrev[j,k] ← bn[Kp1-j+1,k]
42        }
43      }
44
45      bn[K+1,1] ~ dnorm(0, 0.001)
46      bn[K+1,2] ~ dbern(0)
47      beta0 ~ dnorm(0, 0.001)
48      for(j in 1:K){
49        for(k in 1:2){
50          bn[j,k] ~ dnorm(0, 0.001)
51        }
52      }
53      tauy ~ dgamma(0.01, 0.01)
54      sigy ← 1/sqrt(tauy)
55
56      Kp1 ← K+1
57  }
```

A.1.4 Even degree polynomial monotone on $[a, b]$

```
1   model{
2     for(i in 1:N){
3       y[i] ~ dnorm(mu[i], tauy)
4
5       horner[i,1] ← beta[q+1]
6       for(k in 1:q){
7         horner[i,k+1] ← horner[i,k]*x[i] + beta[q+1-k]
```

```
8       }
9       mu[i] ← horner[i,q+1]
10    }
11
12    for(i in 1:Nnew){
13      hnew[i,1] ← beta[q+1]
14      for(k in 1:q){
15        hnew[i,k+1] ← hnew[i,k]*xnew[i] + beta[q+1-k]
16      }
17      mupred[i] ← hnew[i,q+1]
18      ypred[i] ← mupred[i] + eps[i]
19      eps[i] ~ dnorm(0, tauy)
20    }
21
22    for(j in 1:q){
23      beta[j+1] ← alpha*gamma[j]/j
24    }
25    beta[1] ← beta0
26
27    for(j in 2:q){
28      gamma[j] ← b*g1[j] - g1[j-1] + g2[j-1] - a*g2[j]
29    }
30    gamma[1] ← b*g1[1] - a * g2[1]
31
32    for(j in 1:q){
33      g1[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
34                     bnrev[(max(0,Kp1-j)+1):(min(q+1-j,K)+1)
                         ,1])
35      g2[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
```

```
36                          bnrev[(max(0,Kp1-j)+1):(min(q+1-j,K)+1)
                                ,2])
37    }
38
39    for(j in 1:(K+1)){
40      for(k in 1:2){
41        bnrev[j,k] <- bn[Kp1-j+1,k]
42      }
43    }
44
45
46    bn[K+1,1] ~ dbern(0)
47    bn[K+1,2] ~ dbern(0)
48    beta0 ~ dnorm(0, 0.001)
49    for(j in 1:K){
50      for(k in 1:2){
51        bn[j,k] ~ dnorm(0, 0.001)
52      }
53    }
54    tauy ~ dgamma(0.01, 0.01)
55    sigy <- 1/sqrt(tauy)
56
57    Kp1 <- K+1
58  }
```

A.1.5 Odd degree polynomial monotone on $[a, b]$

```
model{
  for(i in 1:N){
    y[i] ~ dnorm(mu[i], tauy)

    horner[i,1] ← beta[q+1]
    for(k in 1:q){
      horner[i,k+1] ← horner[i,k]*x[i] + beta[q+1-k]
    }
    mu[i] ← horner[i,q+1]
  }

  for(i in 1:Nnew){
    hnew[i,1] ← beta[q+1]
    for(k in 1:q){
      hnew[i,k+1] ← hnew[i,k]*xnew[i] + beta[q+1-k]
    }
    mupred[i] ← hnew[i,q+1]
    ypred[i] ← mupred[i] + eps[i]
    eps[i] ~ dnorm(0, tauy)
  }

  for(j in 1:q){
    beta[j+1] ← alpha*gamma[j]/j
  }
  beta[1] ← beta0

  for(j in 2:q){
    gamma[j] ← g1[j] + g2b[j-1] - a*g2b[j]
  }
  gamma[1] ← g1[1] - a * g2b[1]
```

```
31
32    for(j in 2:q){
33      g2b[j] ← b*g2[j] - g2[j-1]
34    }
35    g2b[1] ← b*g2[1]
36
37    for(j in 1:q){
38      g1[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),1],
39                     bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                           ,1])
40      g2[j] ← inprod(bn[(max(0,j-Kp1)+1):(min(Kp1,j)),2],
41                     bnrev[(max(0,Kp1-j)+1):(min(q-j,K)+1)
                           ,2])
42    }
43
44    for(j in 1:(K+1)){
45      for(k in 1:2){
46        bnrev[j,k] ← bn[Kp1-j+1,k]
47      }
48    }
49
50    bn[K+1,1] ~ dnorm(0, 0.001)
51    bn[K+1,2] ~ dbern(0)
52    beta0 ~ dnorm(0, 0.001)
53    for(j in 1:K){
54      for(k in 1:2){
55        bn[j,k] ~ dnorm(0, 0.001)
56      }
57    }
58    tauy ~ dgamma(0.01, 0.01)
```

```
59     sigy ← 1/sqrt(tauy)
60
61     Kp1 ← K+1
62   }
```

A.2 Stan implementation

```
1  functions {
2    vector convolve(vector a, vector b){
3      int na;
4      int nb;
5      int nc;
6      vector[rows(b)] rev_b;
7      vector[rows(a)+rows(b)-1] c;
8
9      na = rows(a);
10     nb = rows(b);
11     nc = na+nb-1;
12     for(i in 1:nb){
13       rev_b[i] = b[nb-i+1];
14     }
15     for(j in 1:nc){
16       int istart;
17       int istop;
18       istart = max(0, j-nb) + 1;
19       istop = min(na, j);
20       c[j] = sum(a[istart:istop] .* rev_b[(nb-j+istart):(nb
             -j+istop)]);
21     }
22     return c;
23   }
24
25   vector horner(vector x, vector beta){
26     int nb;
27     vector[rows(x)] res;
```

```
28
29      nb = rows(beta);
30      res = rep_vector(beta[nb], rows(x));
31      for(i in 1:(nb-1)){
32        res = res .* x + beta[nb-i];
33      }
34      return res;
35    }
36  }
37
38  data {
39    # number of data points
40    int <lower=1> N;
41
42    # degree of polynomial we want to fit
43    int <lower=1> q;
44
45    # degree of sub polynomials, q = 2K or q = 2K + 1
46    int <lower=0> K;
47
48    # mode of operation, derived in the R code
49    # 1 == whole real line
50    # 2 == (-inf, b]
51    # 3 == [a, Inf)
52    # 4 == [a, b]
53    int <lower = 1, upper = 4> operation_mode;
54
55    # y values
56    vector[N] y;
57
```

```
58    # x values
59    vector[N] x;
60
61    # lower bound (could be -Infty / negative_infinity() )
62    real a;
63
64    # upper bound (could be Infty / positive_infinity() )
65    real <lower=a> b;
66
67    # alpha (polynomial is increasing or decreasing)
68    int <lower=-1, upper=1> alpha;
69
70    # number of new data points at which to predict
71    int <lower=1> Nnew;
72    vector[Nnew] xnew;
73  }
74
75  transformed data {
76    int p1_length;
77    int p2_length;
78    int gamma_1_length;
79    int gamma_2_length;
80    int gamma_1_temp_length;
81    int gamma_2_temp_length;
82    vector[2] lower_bound_vector;
83    vector[2] upper_bound_vector;
84
85    gamma_1_temp_length = 0;
86    gamma_2_temp_length = 0;
87    if ((operation_mode == 1) && (q % 2 != 0)) {
```

```
88      p1_length = K + 1;
89      p2_length = K + 1;
90    } else if (operation_mode == 2 || operation_mode == 3) {
91        if (q % 2 == 0) {
92          p1_length = K;
93          p2_length = K;
94        } else {
95          p1_length = K + 1;
96          p2_length = K;
97        }
98        gamma_2_temp_length = 1;
99    } else {
100       if (q % 2 == 0) {
101         p1_length = K;
102         p2_length = K;
103         gamma_1_temp_length = 1;
104         gamma_2_temp_length = 1;
105       } else {
106         p1_length = K + 1;
107         p2_length = K;
108         gamma_2_temp_length = 2;
109       }
110   }
111   gamma_1_length = (2 * p1_length) - 1;
112   gamma_2_length = (2 * p2_length) - 1;
113   if( gamma_1_temp_length != 0)
114     gamma_1_temp_length =  gamma_1_temp_length +
          gamma_1_length;
115   if( gamma_2_temp_length != 0)
116     gamma_2_temp_length =  gamma_2_temp_length +
```

```
        gamma_2_length;
117
118   # This should be the vector (-a, 1) for (a - x)
        convolution
119   lower_bound_vector[1] = -a;
120   lower_bound_vector[2] = 1;
121
122   # This should be the vector (b, -1) for (b - x)
        convolution
123   upper_bound_vector[1] = b;
124   upper_bound_vector[2] = -1;
125 }
126
127 parameters {
128   vector[p1_length] beta_1;
129   vector[p2_length] beta_2;
130
131   real<lower=0> sd_y;
132   real beta_zero;
133 }
134
135 transformed parameters {
136   vector[gamma_1_length] gamma_1;
137   vector[gamma_2_length] gamma_2;
138   vector[gamma_1_temp_length] gamma_1_temp;
139   vector[gamma_2_temp_length] gamma_2_temp;
140   vector[q] gamma;
141   vector[q + 1] beta_final;
142   vector[N] mu;
143
```

```
144   gamma_1 = rep_vector(0, gamma_1_length);
145   gamma_2 = rep_vector(0, gamma_2_length);
146   gamma_1_temp = rep_vector(0, gamma_1_temp_length);
147   gamma_2_temp = rep_vector(0, gamma_2_temp_length);
148   gamma = rep_vector(0, q);
149
150   # self convolute betas to get gammas
151   gamma_1 = convolve(beta_1, beta_1);
152   gamma_2 = convolve(beta_2, beta_2);
153
154   # convolute with lower and upper bounds.
155   # combine together to get gamma
156   if (operation_mode == 1) {
157     gamma = gamma_1 + gamma_2;
158   } else if (operation_mode == 2) {
159       gamma_2_temp = convolve(upper_bound_vector, gamma_2);
160       if(q % 2 == 0) {
161         gamma[1:(gamma_1_length)] = gamma_1;
162         gamma = gamma + gamma_2_temp;
163       } else {
164         gamma[1:(gamma_2_temp_length)] = gamma_2_temp;
165         gamma = gamma + gamma_1;
166       }
167   } else if (operation_mode == 3) {
168       gamma_2_temp = convolve(lower_bound_vector, gamma_2);
169       if(q % 2 == 0) {
170         gamma[1:(gamma_1_length)] = gamma_1;
171         gamma = gamma + gamma_2_temp;
172       } else {
173         gamma[1:(gamma_2_temp_length)] = gamma_2_temp;
```

```
174           gamma = gamma + gamma_1;
175         }
176     } else {
177         if (q % 2 == 0) {
178           gamma_1_temp = convolve(upper_bound_vector, gamma_1
                  );
179           gamma_2_temp = convolve(lower_bound_vector, gamma_2
                  );
180           gamma = gamma_1_temp + gamma_2_temp;
181         } else {
182           gamma_2_temp = convolve(upper_bound_vector,
                  convolve(lower_bound_vector, gamma_2));
183           gamma = gamma_1 + gamma_2_temp;
184         }
185     }
186
187     beta_final[1] = beta_zero;
188     for(i in 1:q) {
189       beta_final[i+1] = alpha * gamma[i] / i;
190     }
191
192     # use horner() to safely evaluate and get mu
193     mu = horner(x, beta_final);
194 }
195
196 model {
197   y ~ normal(mu, sd_y);
198 }
199
200 generated quantities {
```

```
201     vector[Nnew] mupred;
202     vector[Nnew] ypred;
203
204     mupred = horner(xnew, beta_final);
205     ypred = multi_normal_cholesky_rng(mupred, diag_matrix(
            rep_vector(sd_y, Nnew)));
206   }
```

Stan code for random effects model

```
functions {
  vector convolve(vector a, vector b) {
    int na;
    int nb;
    int nc;
    vector[rows(b)] rev_b;
    vector[rows(a)+rows(b)-1] c;

    na = rows(a);
    nb = rows(b);
    nc = na+nb-1;
    for(i in 1:nb){
      rev_b[i] = b[nb-i+1];
    }
    for(j in 1:nc){
      int istart;
      int istop;
      istart = max(0, j-nb) + 1;
      istop = min(na, j);
      c[j] = sum(a[istart:istop] .* rev_b[(nb-j+istart):(nb-j+istop)]);
    }
    return c;
  }

  vector horner(vector x, vector beta) {
    int nb;
    vector[rows(x)] res;

    nb = rows(beta);
    res = rep_vector(beta[nb], rows(x));
    for(i in 1:(nb-1)){
      res = res .* x + beta[nb-i];
    }
    return res;
```

```
35   }
36 }
37
38 data {
39   // total number of observations
40   int <lower=0> N_x;
41
42   // number of individuals
43   int <lower=0> N_i;
44
45   // vectors of x values and y values
46   vector [N_x] x_vector;
47   vector [N_x] y_vector;
48
49   // A vector of positions that has the starting points of each of the
50   // individual's data runs. Hence if the first data run is
        observations
51   // 1 to 31, the position_vector will have elements [1, 32, ...].
52   // It will also have the final value (i.e N_x) in it.
53   int <lower=1>  position_vector[N_i + 1];
54
55   // overall polynomial order
56   int <lower=0> q;
57
58   // sub polynomial order.
59   int <lower=0> k_1;
60   int <lower=0> k_2;
61
62   // Monotonicity direction, 1 for increasing, -1 for decreasing
63   int alpha;
64
65   // The number of coefficients of one of the underlying vectors that
        are
66   // going to be random. If it is k + 1, all of the coefficients are
        random.
67   int <lower=1, upper=(k_1 + 1)> rand_params;
```

```
68
69   // prediction interval x value vector
70   int <lower=0> N_x_new;
71   vector [N_x_new] x_new_vector;
72
73 }
74
75 transformed data {
76   // Construct vector equivalent of convolution with (x - a).
77   // Here hardcoded to a = 0, but could easily be changed and passed in
         as data.
78   vector[2] lower_bound_vector;
79   lower_bound_vector[1] = 0;
80   lower_bound_vector[2] = 1;
81
82 }
83
84 parameters {
85   real <lower=0> sd_y;
86
87   vector [N_i] beta_zero;
88   real beta_zero_mean;
89   real<lower=0> beta_zero_sd;
90
91   matrix  [N_i, rand_params]  p_1_rand;
92   vector  [rand_params] p1_rand_ef_mean;
93   vector <lower=0> [rand_params] p1_rand_ef_sd;
94
95   vector [(k_1 + 1) - rand_params] p_1_fixed;
96   vector [k_2 + 1] p_2;
97 }
98
99 transformed parameters {
100   matrix[N_i, q + 1] beta_final;
101   matrix[N_i, (2*k_1) + 1] gamma_1;
102   matrix[N_i, (2*k_2) + 2] gamma_2;
```

```
103   matrix[N_i, q] gamma;
104   vector[N_x] mu;
105
106   mu = rep_vector(0, N_x);
107
108   for(ii in 1:(N_i)) {
109     vector[k_1 + 1] temp_p1;
110     //vector[position_vector[ii + 1] - position_vector[ii]] temp_mu;
111     int lower_loop;
112     int upper_loop;
113
114     lower_loop = position_vector[ii];
115     upper_loop = position_vector[ii + 1] - 1;
116
117
118     // set entries in matrix to zero (they aren't initialised to be)
119     gamma_1[ii] = rep_row_vector(0, (2*k_1) + 1);
120     gamma_2[ii] = rep_row_vector(0, (2*k_2) + 2);
121
122     temp_p1 = append_row(p_1_rand[ii]', p_1_fixed);
123
124     // put in correct rows of 'temporary' matrix gamma_i
125     gamma_2[ii] = convolve(convolve(p_2, p_2), lower_bound_vector)';
126     gamma_1[ii] = convolve(temp_p1, temp_p1)';
127
128     // add the zero correctly
129     if (q % 2 == 0) {
130       gamma[ii] = (append_row(gamma_1[ii]',rep_row_vector(0,1)') +
            gamma_2[ii]')';
131     } else {
132         gamma[ii] = (append_row(gamma_2[ii]', rep_row_vector(0,1)') +
              gamma_1[ii]')';
133     }
134
135     beta_final[ii, 1] = beta_zero[ii];
136
```

```
137     for(qq in 1:q) {
138       beta_final[ii, qq + 1] = alpha *  (gamma[ii, qq] / qq);
139     }
140
141     mu[lower_loop:upper_loop] = horner(x_vector[lower_loop:upper_loop],
            beta_final[ii]');
142
143   }
144 }
145
146 model {
147   for (ii in 1:(N_i)) {
148     int loop_lower;
149     int loop_upper;
150     loop_lower = position_vector[ii];
151     loop_upper = position_vector[ii + 1] - 1;
152     y_vector[loop_lower:loop_upper] ~  normal(mu[loop_lower:loop_upper
           ], sd_y);
153
154     p_1_rand[ii] ~ normal(p1_rand_ef_mean, p1_rand_ef_sd);
155
156   }
157
158   beta_zero ~ normal(beta_zero_mean, beta_zero_sd);
159   p1_rand_ef_sd ~ normal(0, 0.1);
160   p1_rand_ef_mean ~ normal(0, 100);
161   beta_zero_mean ~ normal(0, 100);
162   sd_y ~ normal(0,1) T[0,];
163   beta_zero_sd ~ normal(0,1) T[0,];
164
165   p_1_fixed ~ normal(0, 100);
166   p_2 ~ normal(0, 100);
167
168 }
169
170 generated quantities {
```

```
171   matrix [N_i, N_x_new] mu_new;
172   matrix [N_i, N_x_new] y_indiv_new;
173
174   for (ii in 1:(N_i)) {
175     mu_new[ii] = horner(x_new_vector, beta_final[ii]')';
176     y_indiv_new[ii] = multi_normal_cholesky_rng(mu_new[ii]',
          diag_matrix(rep_vector(sd_y, N_x_new)))';
177   }
178
179 }
```

A Full Fit

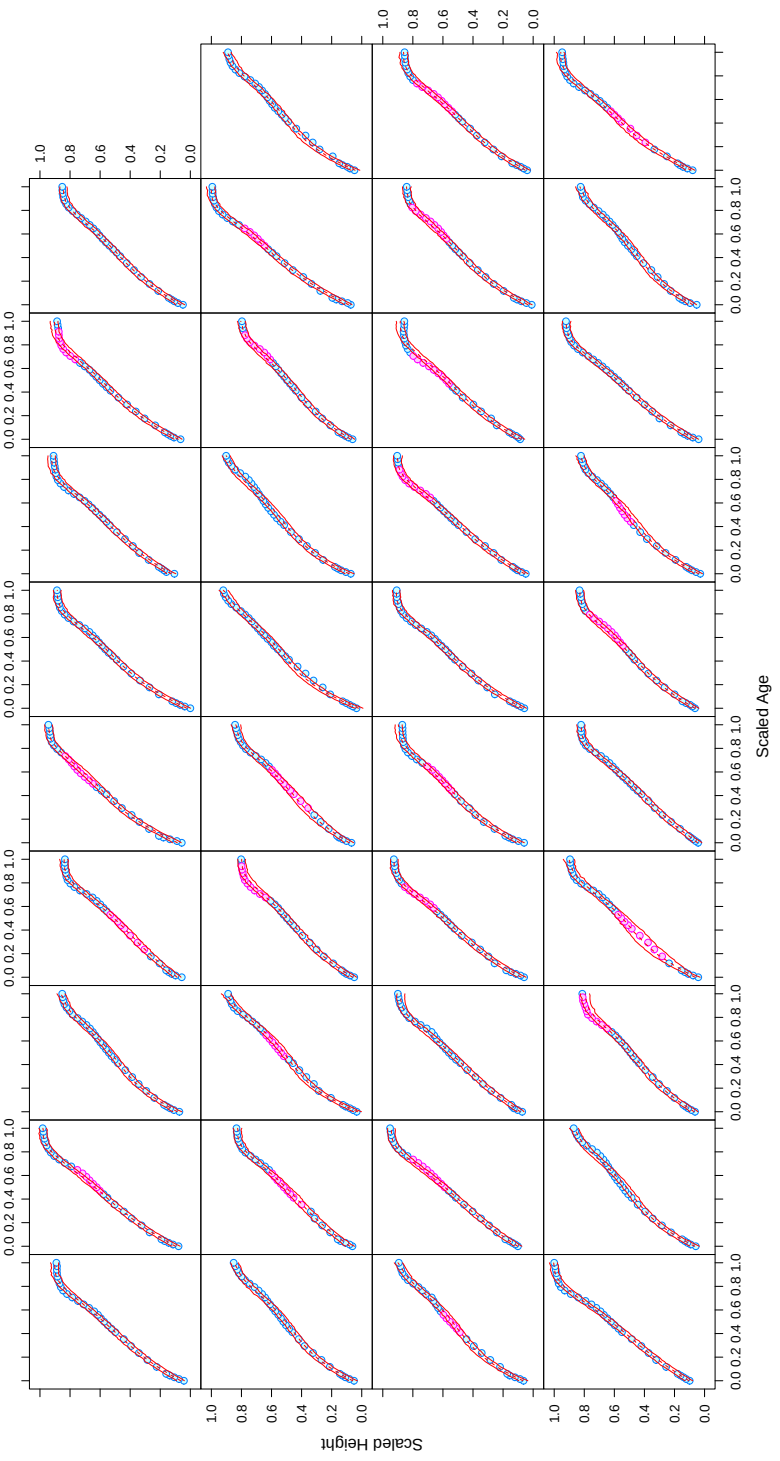


Figure C.1: An overview of the fitted model on the full population with missing data. Missing data points are in pink, observed data points are in blue, and the 95% prediction intervals and individual mean curves are in red.

Data set subsetting function for interpolation test

```
1  library(fda)
2
3  dataProducer <- function(p_yes_no, p_num_obs, max_to_remove) {
4  # This function produces two data frames (returns both in a list).
5  # The first is the dataframe with the observations removed according
       to
6  # the method specified in the thesis. The second is the full data set
        with an
7  # extra column indicating if that particular observation was removed
       or not.
8
9  y <- as.vector(growth$hgtm)
10 y <- (y - min(y)) / diff(range(y))
11 x <- growth$age
12 x <- (x - min(x)) / diff(range(x))
13 full_data <- data.frame(y = y, x = x)
14
15 n_indiv <- dim(growth$hgtm)[2]
16 n_obs_per_indiv <- length(growth$age)
17
18 yes_no <- rbinom(n = n_indiv, size = 1, prob = p_yes_no)
19 num_remove <- rbinom(n = n_indiv, size = max_to_remove, prob =
       p_num_obs)
20 num_obs_to_remove <- yes_no * num_remove
21
22 removal_to_start <- sample(x = 1:(n_obs_per_indiv - max_to_remove -
       2), size = n_indiv, replace = TRUE) + 2
23 pos_vec <- c(1, 1:(n_indiv) * n_obs_per_indiv + 1)
24 drop_vec <- 1:length(y)
25
26 person_vec <- c()
27 for (i in 1:n_indiv) {
28   person_vec <- c(person_vec, rep(i, 31))
29 }
```

```
30
31   full_data$person_vec <- person_vec
32
33   for (ii in 1:n_indiv) {
34     starting_index <- pos_vec[ii] + removal_to_start[ii] - 1
35     number_to_drop <- num_obs_to_remove[ii]
36
37     if (number_to_drop == 0) {
38       next
39     }
40
41     stopping_index <- starting_index + num_obs_to_remove[ii] - 1
42     drop_vec[starting_index:stopping_index] <- -drop_vec[starting_index
           :stopping_index]
43
44   }
45
46   full_data$drop_vector <- drop_vec
47   full_data$was_dropped <- drop_vec < 0
48   row_dropper <- as.numeric(full_data$was_dropped) * drop_vec
49   dropped_data <- full_data[row_dropper, ]
50   rownames(dropped_data) <- 1:nrow(dropped_data)
51
52   new_pos_vec <- pos_vec - c(0, cumsum(num_obs_to_remove))
53   res_list <- list(list(full_df = full_data), list(dropped_df =
         dropped_data, dropped_person_vec = new_pos_vec))
54
55   return(res_list)
56
57 }
```

A numerical issue akin to label switching

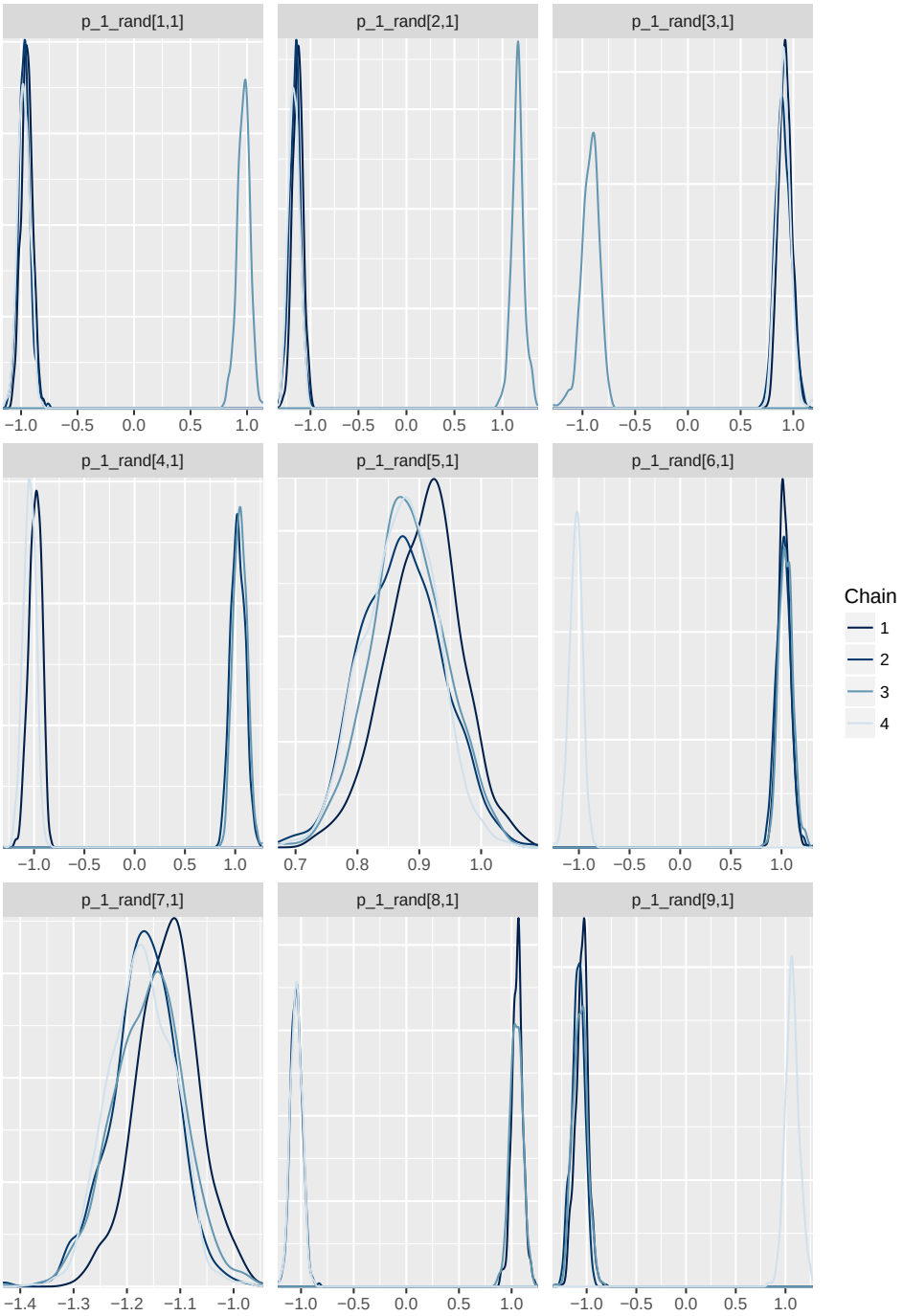


Figure E.1: Posterior density estimates for a subset of the subject-specific terms in the model presented in Chapter 3, split by MCMC chain. Several parameters in this plot display behaviour that is akin to "label switching" in mixture models.

QR decomposition

```
1   genAllMatrices <- function(x, d) {
2 # generate the q, r_inverse matricies from a (column) vector of x
      values,
3 # and a polynomial degree d.
4 x <- sort(x)
5
6 # set up matrix w/ correct dimensions
7 Q <- matrix(NA, nrow = length(x), ncol = d + 1)
8 R_inv <- matrix(0, nrow = d + 1, ncol = d + 1)
9
10 # renormalising constants
11 A <- B <- C <- 0
12 gamma <- rep(0, d + 1)
13 k <- rep(1, d + 1)
14
15 # setup first two cols, corresponding to the 1 and x cols in the X
      matrix
16 n <- length(unique(x))
17 Q[ ,1] <- 1 / sqrt(n)
18 k[1] <- 1 / sqrt(n)
19 R_inv[1, 1] <- k[1]
20
21 # fill  in all inital values
22 Q[, 2] <- x - mean(x)
23 k[2] <- 1 / (sqrt(sum(Q[, 2]^2)))
24 Q[, 2] <- Q[, 2] * k[2]
25 R_inv[1, 2] <- -mean(x)
26 R_inv[2, 2] <- 1
27 R_inv[, 2] <- R_inv[, 2] * k[2]
28
29 gamma[1] <- k[2] / k[1]
30
31 for (ii in seq_len(d - 1) + 2) {
32   gamma[ii - 1] <- k[ii] / k[ii - 1]
```

```
33   A <- -gamma[ii - 1] * sum(x*Q[, ii - 1]^2)
34   B <- gamma[ii - 1]
35   C <- (gamma[ii - 1] * sum(Q[, ii - 1]^2)) /
36        (gamma[ii - 2] * sum(Q[, ii - 2]^2))
37
38   Q[, ii] <- (A + B*x)*Q[,ii - 1] - C*Q[, ii - 2]
39   ix1 <- 1:(ii - 1)
40   ix <-  1:ii
41   R_inv[ix, ii] <- A*R_inv[ix, ii - 1] +
42                    B*c(0, R_inv[ix1, ii - 1]) -
43                    C*R_inv[ix, ii - 2]
44
45   ## reorthogonalisation
46   for (jj in 1:(ii - 1)) {
47     tmp <- as.vector(crossprod(Q[, jj], Q[, ii]))
48     Q[, ii] <- Q[, ii] - tmp*Q[, jj]
49     R_inv[,ii] <- R_inv[,ii] - tmp*R_inv[,jj]
50   }
51
52   k[ii] <- 1 / sqrt(sum(Q[, ii]^2))
53   gamma[ii - 1] <- k[ii] / k[ii - 1]
54   Q[, ii] <- Q[, ii] * k[ii]
55   R_inv[, ii] <- R_inv[, ii] * k[ii]
56
57 }
58 return(list(Q = Q, R_inv = R_inv))
59 }
```