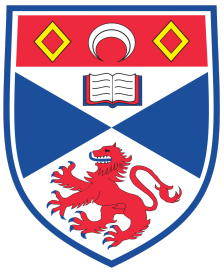


Sequential Monte Carlo in Population Dynamics

A THESIS PRESENTED
BY
STUART BURRELL
TO
THE SCHOOL OF MATHEMATICS AND STATISTICS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF MATHEMATICS

UNIVERSITY OF ST ANDREWS
FIFE, SCOTLAND

APRIL 2016

Supervisor: Dr. Len Thomas Stuart Burrell

Sequential Monte Carlo in Population Dynamics

Abstract

We aim to empower the undergraduate statistician to create effective SMC algorithms by demonstrating an application in ecological population dynamics. To begin, we introduce the notion of density dependance and an associated state-space model from [1]. Next, we fit this model for the North American Redhead (*Aythya americana*) with an MCMC approach from [1]. Next, by reconducting the analysis from [1] over an extended time-frame, we obtained an additional benchmark with which to evaluate the performance of SMC methods. The following three chapters progress from the theoretical foundations of SMC to the simplest of SMC algorithms, the bootstrap filter [2], and on to the more sophisticated auxiliary particle filter (APF) of West and Liu [3]. This choice is thematic, and centres around the challenge to SMC known as particle depletion. To conclude we outline a number of further enhancements for investigation, including a method for real-time adaptation for the APF based on a novel diagnostic derived from the concept of effective sample size.

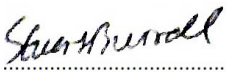
Contents

Listing of figures

Acknowledgments

First and foremost I'd like to thank my supervisor Dr. Len Thomas for all his generous help and support. In addition, the work of Jamieson and Brooks [1], Doucet, de Freitas and Gordon [2], and West and Liu [3] has been very instructive, for which I am enormously grateful. Finally I'd like to give my appreciation to everyone who has supported me throughout the year.

I certify that this project report has been written by me, is a record of work carried out by me, and is essentially different from work undertaken for any other purpose or assessment.



0

Introduction

Contemporary Bayesian analysis is highly computational with an ever-growing and diverse praxis. Of these, Sequential Monte Carlo (SMC) methods perform inference by constructing samples from the distributions of unknown quantities through a Darwinian evolutionary process. This is achieved, in part, by sequentially updating our estimations as we incrementally introduce the data. Thus, SMC represents an approach to Bayesian inference that lends itself, but is not restricted to, applications where data becomes available sequentially in time. In recent years their popularity has soared, with applications emerging in fields ranging from artificial intelligence and target-tracking, to ecological population

dynamics and forecasting.

We aim to empower the undergraduate statistician to create effective SMC algorithms by demonstrating an application in ecological population dynamics. In particular, we compliment the necessary theoretical machinery with a comprehensive example for the North American Redhead, intended to translate the abstract into a clear process. One of the themes we will see is the conflict between SMC methods and dimensionality. This is highlighted in our discussions on results, which primarily focus on algorithmic performance and efficacy.

Chapter 1 sets the scene for our example, starting with an introduction to the species we consider, the North American Redhead, before moving on to the notion of density dependant population growth and how to model it mathematically. This model forms the backbone of later chapters, since simulation type methods such as SMC are based around being able to generate possible future population states in a systematic way.

Next, chapter 2 aims to establish benchmarks with which to evaluate the performance of our algorithms. Typically, one might use a simulated data set for this purpose. However, to emphasize the real-world application of SMC, we opted to use real analysis derived from the hugely popular Markov Chain Monte Carlo (MCMC) methods. In particular, our first benchmark originates from the analysis of Jamieson and Brooks in [1]. Furthermore, we reconducted their analysis to test a similar MCMC implementation, which we then used to generate an additional benchmark for an extended time period.

The following three chapters represent a natural progression through SMC that gradually builds in sophistication. Chapter 3 develops the core theory through a literature review of [2]. In chapter 4 we describe and develop our first implementation, known as the

bootstrap filter (BF). Two modifications of the BF by West and Liu [3] yield the auxiliary particle filter (APF), which we discuss in chapter 5. Additionally, we comprehensively assess the performance of the APF over the two time frames, including a comparison with the corresponding bootstrap filter analysis.

Our demonstrations will show the advantages of the APF, but, due to the need to configure additional parameters for the algorithm itself, also bring to light the added complexity it entails. Addressing this problem is one of the main objectives of Chapter 6, where we develop diagnostic tools that enable the algorithm to automatically configure itself in real-time. The final parts of this chapter include a suggestion for how to utilise parallel computing, and some suggestions for further reading.

We conclude with a brief chronological walkthrough of the material covered to distill and condense the most important concepts we cover.

'So once you do know what the question actually is,
you'll know what the answer means.'

The Hitchhiker's Guide to the Galaxy

1

The North American Redhead

1.1 Setting the scene

The purpose of this chapter is to set the scene and introduce the mathematical model which will be used throughout, along with its underlying concepts. First, we'll introduce the North American Redhead and the source of our data. Secondly, we look at density dependance, a property which describes how the size of a population regulates its growth. This will lead us on to the final part of the chapter, where we present a model of population growth able to reflect this from [1], which is in part based on the work of Dennis and Taper in [4].

Figure 1.1: The Redhead.*

The Redhead (*Aythya americana*) is a diving duck native to North America, with the population distributed as far south as Mexico during the winter months and northern Canada throughout breeding season, as depicted in figure 1.2. In 2004, Jamieson and Brooks gave evidence supporting density dependance in the redhead population [1] and conjectured a causal relationship between strong density dependance and diving, as opposed to dabbling, ducks. As we shall see, this gave a suitably challenging testing environment for our algorithms, since these models contain multiple parameters to estimate.

The current population size stands at approximately 1.2 million. This far exceeds the figure of 760,000 proposed by the North American Waterfowl Management Plan [5]. However, should future population analysis indicate a collapse, future management will likely centre the conservation of the wetland where they breed and forage underwater [5].

In our analysis we use population estimates originating from the report *Trends in Duck Breeding Populations 1955-2015* [6] published by the U.S. Fish and Wildlife Service in 2015. The relevant data from this report has been included in appendix B.

This dataset was particularly appealing for a number of reasons. Firstly, the large quan-

*Modified, image from http://carolinabirds.org.

tity of complete data it contained. We were able to work with population point estimates and standard errors for the population of Redheads from 1955 through to 2015. Crucially, the historical extent of the data allowed us to consider the performance of SMC over two significantly different and substantial time periods. Secondly, given the source and the effort expended to produce the report we would expect a low coefficient of variation. Finally, it allowed us to benefit from [1], which analysed a portion of this data set between 1961-2002 using the Markov Chain Monte Carlo methods that we introduce in chapter 2.

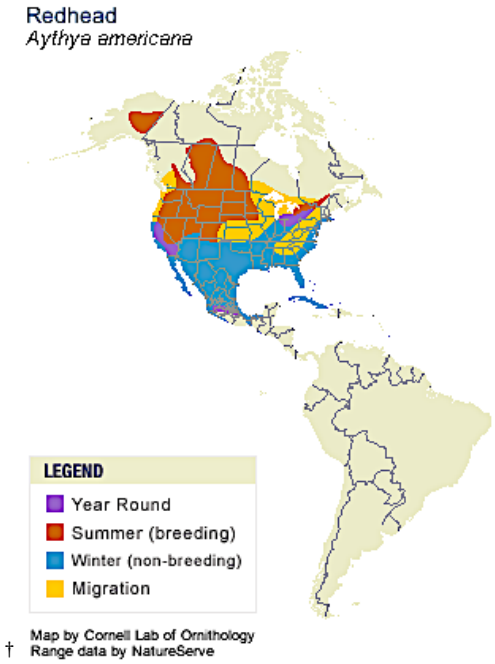


Figure 1.2: Geographical distribution of the Redhead.[‡]

[‡]Image from https://www.allaboutbirds.org

1.2 Density dependant population model

This section introduces the reader to the aforementioned ecological concept of density dependance in more detail, and then goes on to explore the ways in which this can be modelled in a formal mathematical framework. The choice of how to model density dependance is controversial [1], and so for comparability we adopt the choice of [1], which is a particular instance of the popular and very general state-space model.

1.2.1 Density dependence

Density dependance is a property that characterises the way the growth rate of a population is influenced by the the number of that species in the environment. More formally, we might consider how factors such as emigration, immigration, survival rate, or fecundity influence the growth rate [7].

Density dependance is complex and may either positively or negatively impact the growth rate. In addition, it may also be the case that the density dependance occurs over a larger time period. For example the population size one or two years prior may affect the current growth rate. This is referred to as first order and second order density dependance respectively.

There are intuitive reasons for a population exhibiting density dependant growth which can best be illustrated with an example. For a predatory animal, population increase may lead prey to become scarce causing a reduction in population growth due to scarcity of a primary resource: food. Competition for other valuable commodities can equally induce density dependance, for example: nesting materials or precious breeding habitat.

These examples of negative density dependance naturally describe the way populations

eventually limit themselves in an environment to something ecologists call a carrying capacity [8]. Positive density dependance is often less apparent. However, simple examples include how a dense patch of trees can help protect members from hazardous weather, or how fish gather together in a dense shoal for protection from predators.

Methods for detecting density dependance has evolved significantly over time. Early approaches used simple hypothesis testing, and considered whether there was sufficient evidence in an example to reject a null hypothesis of density dependance [7]. This approach received a number of criticisms, primarily due to a lack of statistical power, meaning it would often cause a rejection of the null hypothesis even when density dependance is present [7]. This has encouraged recent researches to embark on a new model selection approach [7]. Indeed, it is this approach that Jamieson and Brooks adopt to detect density dependance in [1].

1.2.2 The population model

Next, we consider the way in which we have mathematically modelled the Redhead population dynamics to incorporate the idea of density dependance discussed above. Our approach consists of a 'system process model', corresponding to the evolution of the true population size through time, and an 'observation process', which describes the relationship between the true value and the recorded data. We first specify the system process model, which originates from the work of Dennis and Taper [4] in 1994. This is a simple approach and can be described as follows.

Model 1.2.1 *Let N_t denote the size of the population at times $t = 0, 1, 2, ...,$, and define N_t*

recursively by

$$N_t = N_{t-1} \exp\left(b_0 + b_1 N_{t-1} + \sigma Z_t\right), \tag{1.1}$$

where $b_0, b_1 \in \mathbb{R}$, $\sigma \in \mathbb{R}^+$ *and* $Z_t \sim N(0,1)$.

However, notice that the above model allows the population at time t to depend only on the population at time $t-1$ and not previous time steps. In the analysis of Jamieson and Brooks, it was necessary to generalise this model with an extension which allowed the population to depend on an arbitrary specified number of previous years, as first suggested by Dennis and Taper in [4]. Recall that we refer to the specified time lag as the order of the density of dependance. This extended model, for which we are given strong evidence to use with $k = 2$ for the Redhead population by [1], is as follows:

Model 1.2.2 *Let* N_t *denote the size of the population at times* $t = 0, 1, 2, ...,$ *and denote the order of the density dependance by* $k \in \mathbb{N}$. *Then* N_t *is defined recursively by*

$$N_t = N_{t-1} \exp\left(b_0 + \sum_{i=1}^{k} b_i N_{t-i} + \sigma Z_t\right), \tag{1.2}$$

where $b_0, b_1, ..., b_k \in \mathbb{R}$, $\sigma \in \mathbb{R}^+$ *and* $Z_t \sim N(0,1)$.

Furthermore, drawing on an article by Bonner in [9], we give an interpretation of these parameters. First of all, the b_i terms describe the influence of N_{t-i} on the growth rate at N_{t-1}, with the exception of b_0, which determines an expected growth rate given a population of size very near zero. Secondly, σ scales the size of the 'process errors', given by Z_t, which account for random variation in the population growth.

Next, we recognise that the act of data collection entails its own inherent stochasticity. Thus, we must formulate a second model, the observation process, to relate the true population to the observed values. This is given by:

$$y_t = N_t + S_t Z_t, \tag{1.3}$$

where y_t denotes the observation at time t, S_t denotes the standard error of the data collection and $Z_t \sim N(0, 1)$.

As motivation for our next chapter, note an observation in [4], that the above models of 1.2.2 are Markov processes of order k. We now return to the Redhead to discuss the application of a set of related methods known as Markov Chain Monte Carlo analysis, one of the most popular Bayesian computational methods in modern usage.

'There's always a bigger fish.'

'Qui-Gon Jinn', The Phantom Menace

2

A Benchmark with MCMC

Markov Chain Monte Carlo (MCMC) is a powerful and widely applicable method for statistical inference. In this section we present and discuss an implementation of MCMC used to fit a density dependant population model for the Redhead, based on the work of Jamieson and Brooks in [1]. In the first instance, we attempt to reproduce their results in order to test our implementation. Secondly, we extend their analysis over a longer time-period, which provided us with additional benchmarks to those in [1]. These benchmarks are used to compare our SMC methods in chapters 4 and 5. However, preceding this we give a brief overview of the foundational theory.

2.1 The Bayesian paradigm

In order to understand MCMC, or indeed SMC, we first need to appreciate the Bayesian statistical paradigm they inhabit. In the world Bayesian statistics, all statistical inference arises from application of Bayes' theorem, originating from the paper [10] by Reverend Thomas Bayes in the 18$^{\text{th}}$ century. Bayes' theorem allows us to update our prior beliefs about a parameter of interest in light of new data, and represents a logical approach to learning. In the modern day, we typically state it in the following way for continuous parameters.

Theorem 2.1.1 *Let $\theta \in \Theta$ be a continuous parameter of interest with a prior distribution $p(\theta)$ which describes our current beliefs about θ. Furthermore, suppose we observe data $\boldsymbol{x} = (x_1, ..., x_n)$ from a probability density function $f(\boldsymbol{x}|\theta)$. Then Bayes' theorem states*

$$\pi(\theta|\boldsymbol{x}) = \frac{f(\boldsymbol{x}|\theta)p(\theta)}{f(x)} \tag{2.1}$$

where the posterior distribution $\pi(\theta|\boldsymbol{x})$ represents our updated beliefs about θ [11].

In plain English, Bayes' theorem tells us that the posterior distribution is proportional to the likelihood multiplied by the prior distribution. Importantly, this provides a framework for making inference based on observations that are made sequentially in time. Naturally, the ideal strategy would be to calculate the posterior distribution directly through analytic means. However, this is not always possible since the denominator in equation 2.1 is equivalent to the integral:

$$\int_{\Theta} f(\boldsymbol{x}|\theta)p(\theta)\,d\theta \tag{2.2}$$

which may be high dimensional, complex and analytically intractable. MCMC and Sequential Monte Carlo methods constitute some of the options for circumventing this problem. They avoid the need to compute 2.2 by obtaining a sample whose density approximates the posterior distribution.

2.2 The foundations of MCMC

We begin by looking at the basic definitions on which MCMC is built. Firstly, a Markov chain is a stochastically generated sequence of numbers $x_0, x_1, x_2, ...$ such that the distribution of x_{t+1} only depends on x_t [11]. In the context of state-space models, we refer to the elements of the sequence as states. If we wish to create a Markov chain, we can construct a series of stochastic maps $p(x_{t+1}|x_t)$, known as the 'transition kernel' or 'system process'. These define the evolution of the sequence. Thus, given some seed value x_0 at $t = 0$, the transition kernel enables us to construct a chain. For our applications, where the true value of the chain is considered unobservable or 'hidden', we are also required to specify an 'observation process' equation - as discussed in section 1.2.2. Together, these are known as a Hidden Markov model (HMM) [12].

In order to use this construction for statistical inference, we first need to describe a key property of Markov chains. In particular, it is possible to construct chains that converge to a 'stationary distribution' [13]. This is a distribution such that, for potential values of the chain i, j, the probability that $x_{n+1} = j$ given $x_n = i$ is independent of n - providing n is large enough. Importantly, the stationary distribution does not depend on the seed value of

the chain [11]. This is of particular value in scenarios where we have very little information on which to base a choice.

Recall that in the Bayesian paradigm we are seeking to gain information about a posterior distribution, either through analytic computation or by constructing a sample whose density approximates it. Using the above theory, it is clear that if we can create a Markov chain with a stationary distribution equal to the posterior, then we can construct a sample from the posterior once convergence has occurred. Unfortunately, the best theoretical tools to date cannot place adequate bounds on how many iterations are required to reach the stationary distribution [14]. However, from a practical point of view, we simply iterate the chain to some reasonably large depth, known as the 'burn in', and then utilise diagnostics which can offer sufficient indication of whether convergence has occurred.

2.3 MCMC analysis of the Redhead population

In this section we use MCMC to fit state-space model described in chapter 1 for the Redhead. The results of Jamieson and Brooks in [1] are based on data from 1961 to 2002. We first test our implementation by reproducing their parameter estimates and then update their analysis in light of the available data for 1955-2015. Importantly, this allowed us to test the performance of our SMC methods over two timeframes which provided valuable insight.

Furthermore, as in all Bayesian analysis we need to specify priors for the parameters we wish to estimate. We opted for the following choices given in table 2.1. Despite the prior sensitivity analysis performed in [1], it is convincingly argued in [15] that the following are superior choices, which we use both here and throughout. In addition, on the basis of [15],

we omit a prior sensitivity analysis of our own.

Parameter	Prior Distribution
b_0	$N(0,1)$
b_1	$N(0,1)$
b_2	$N(0,1)$
σ	$\Gamma^{-1}(0.001, 0.001)$

Table 2.1: Prior distributions for the model parameters.

We will now present an example of the code used to implement our MCMC analysis, before moving on to a brief discussion of the results.

2.3.1 Code

The analysis was performed with the programming language R [16] with the package 'rjags' which allows JAGS (Just Another Gibbs Sampler), a tool for MCMC implementation, to be used an R environment. We set $k = 2$ in model 1.2.2, corresponding to second order density dependance. This is the most probable model according to [1]. For consistency with [1], this example implementation considered the data between 1961 and 2002 and used MCMC simulations with 2,020,000 iterations, including a burn in of 20,000. The following code is based on code obtained from L. Thomas, which in turn is closely related to code found in [9].

```
library(rjags)
library(XLConnect)
in.data <- readWorksheetFromFile("duckdata.xlsx", sheet=1)
Nhat <- in.data[7:48, 16]/1000
SE <- in.data[7:48, 17]/1000
bigT <- length(Nhat)

modelstring <-
  'model{

  ### Priors

  # b_0,  b_1 and b_2
  b[1] ~ dnorm(0, 1)
  b[2] ~ dnorm(0, 1)
  b[3] ~ dnorm(0, 1)

  # Population size at times 1,2 with standard error, in millions
  N[1] ~ dnorm(Nhat[1], 1/(SE[1]*SE[1]))T(0,)
  p[1] <- log(N[1])
  N[2] ~ dnorm(Nhat[2], 1/(SE[2]*SE[2]))T(0,)
  p[2] <- log(N[2])
```

```
23
24    # Sigma determines size of random fluctuations in the process
        model
25    tau ~ dgamma(0.001, 0.001)
26    sigma2 <- 1/tau
27    sigma <- pow(sigma2, 0.5)
28
29    for(t in 3:bigT){
30      # Process model
31      Ep[t] <- p[t-1] + b[1] + b[2]*exp(p[t-1])+ b[3]*exp(p[t-2])
32
33      # Realized value with process error
34      p[t] ~ dnorm(Ep[t], tau)
35      N[t] <- exp(p[t])
36
37      # Observation model
38      prec[t] <- 1/(SE[t]*SE[t])
39      Nhat[t] ~ dnorm(N[t], prec[t])
40    }
41  }'
42
43  # Input data
44  dat = list('Nhat' = Nhat,'SE' = SE, 'bigT' = bigT)
```

```
# Load model
m <- jags.model(textConnection(modelstring), data = dat)
# Burn in, in addition to the default burn in of 1000
update(m, 19000)
# Sample for inference
n.iter <- 2000000
samples <- coda.samples(m, c("b[1]", "b[2]", "b[3]", "sigma"), n.
    iter=n.iter)
# Plot samples
plot(samples)
# Record and print summarized results
results <- summary(samples)
print(results)
```

1961-2002: MCMC implementation for parameter estimation.

2.3.2 Results

The first stage in assessing the output of an MCMC analysis is to determine whether it is likely the burn-in was sufficient for convergence to the stationary distribution. One diagnostic for this is to evaluate the 'trace plots', which tell us the realised value of the chain at each iteration. Figure 2.2, which is included at the end of this section, gives the trace plots, along with the posterior parameter distributions for the test 1961-2002 analysis (noting that parameter b_i is labelled $b[i+1]$). The important observation to make from figure 2.2 is that that the trace plots are consistently centred around a common value, thus creating a broad horizontal strip. From the discussion in [13], we can infer that this is a reasonable indication that convergence has occurred. For our purposes, when supported by [1], this is sufficient evidence that the stationary distribution has been reached. However, in other contexts, the reader may wish to make use of diagnostics detailed in [17] by Gelman and Brooks. Next, we consider our results for the initial analysis over 1961-2002 and include the point estimates from [1] for comparison.

Parameter	Point estimates [1]	Posterior Mean	2.5% Quantile	97.5% Quantile
b_0	0.194	0.191	0.0427	0.390
b_1	0.358	0.360	-0.617	1.199
b_2	-0.652	-0.660	-1.450	0.250
σ	0.078	0.0919	0.0393	0.160

Table 2.2: 1961-2002: MCMC parameter point estimates and quantiles with corresponding point estimates from [1].

Table 2.2 illustrates that our estimates closely match that of Jamieson and Brooks in [1]. The values deviate with errors in the order of 10^{-3} to 10^{-2}, which is well within the

bounds of expected discrepancy due to the randomness inherent in the procedure (known as Monte Carlo error). Our ability to reproduce these results gives us confidence in using this same algorithm to generate benchmark values using data from 1955 to 2015. We give the a summary of the results from this extended analysis in table 2.3.

Parameter	Posterior Mean	2.5% Quantile	97.5% Quantile
b_0	0.0738	-0.0338	0.185
b_1	0.234	-0.404	0.875
b_2	-0.328	-0.991	0.335
σ	0.117	0.0689	0.171

Table 2.3: MCMC 1955-2015: parameter point estimates and quantiles.

We omit the trace plots, since they were satisfactory and similar to figure 2.2. Figure 2.1 gives plots of the posterior densities (recall that $b_i = b[i+1]$).

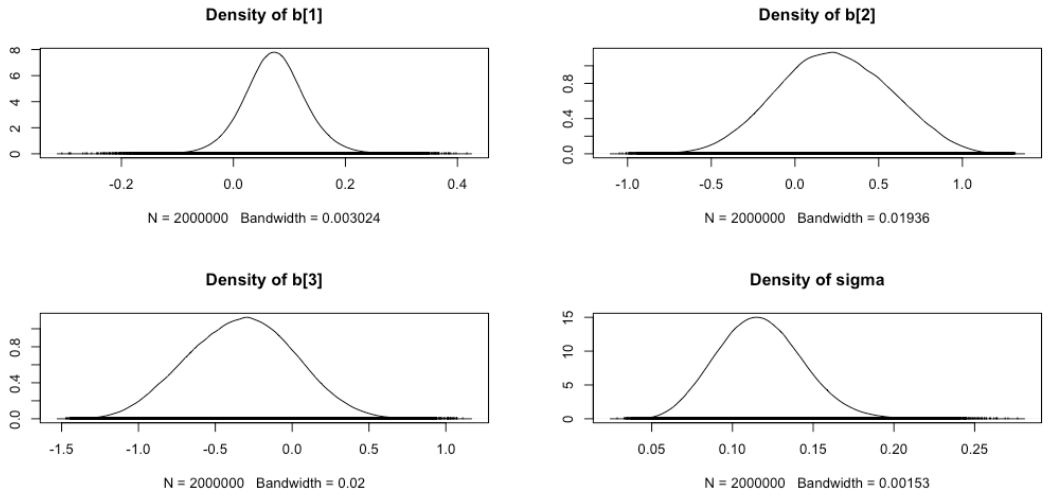


Figure 2.1: 1961-2002: MCMC parameter posterior densities.

As you would expect, the larger data set with the extended analysis produced narrower confidence intervals, which is visually reflected the density plots. In addition, the mean values of the posteriors have changed considerably in the case of $b_0,\ b_1$ and b_2. This is likely due to the over simplicity of our model, since the actual environment contains a huge number of additional covariates such as hunting quotas. For our purposes this is not an issue, since we are still able to compare the performance of SMC with MCMC. Despite the change in parameter estimates, it is interesting to note that there is a similar pattern. Specifically, over both time periods we observe $|\overline{b}_0| \leq |\overline{b}_1| \leq |\overline{b}_2|$ with $\overline{b}_0, \overline{b}_2 > 0$ and $\overline{b}_1 < 0$, where $\overline{b}_i$ denotes the mean of the posterior distribution for b_i.

We conclude this chapter with a comment on what this pattern might imply from an ecological perspective. It is suggested in [1] that diving ducks such as the Redhead may delay breeding due to challenging environmental conditions such as a lack of food or a loss of habitat. This yearly inconsistency could incur this form of density dependance where the population two years prior has a large negative impact on the current growth rate. For example, if the population two years prior was particularly large, there may have been a scarcity of food resulting in a change in breeding habits. This could have resulted in less ducks at breeding age in the current year, thus reducing the growth rate. On the other hand, if the current population was the same but the population two years prior was in less, then scarcity of food less likely to be an issue and impact breeding.

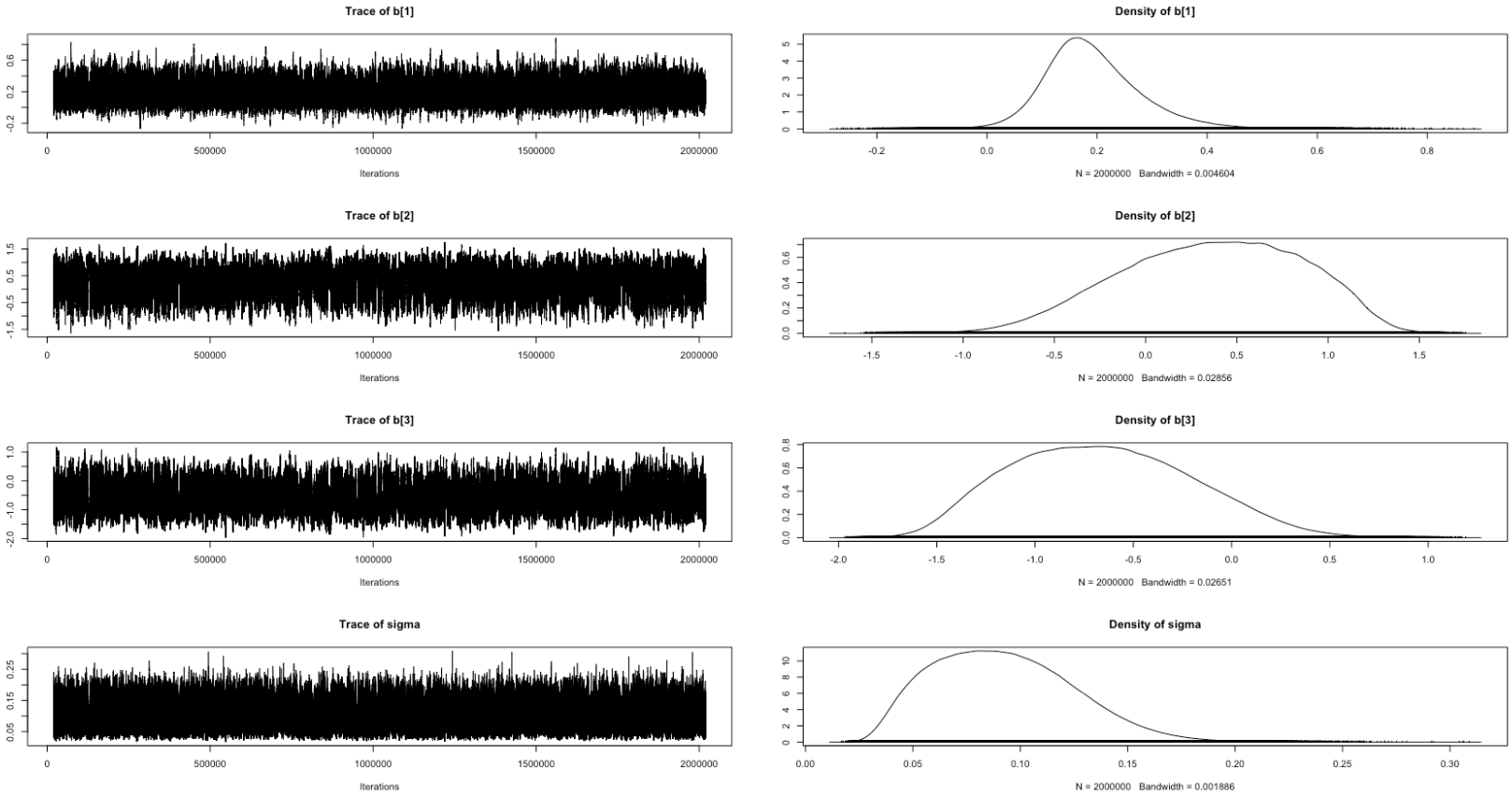


Figure 2.2: 1961-2002: MCMC trace plots and posterior densities.

'We demand rigidly defined areas of doubt and uncertainty!'

'Vroomfondel', The Hitchhiker's Guide to the Galaxy

3

Sequential Monte Carlo Methods

In this section we aim to build an appreciation for the theoretical foundations of SMC. We review the comprehensive introduction[2] by Doucet, de Freitas and Gordon. Our hope is that this will illuminate the code presented in later sections by allowing the reader to recognise how these methods function on a deeper level. Starting with the basic theory, we will progress to the core methods known as 'importance sampling' and 'sequential importance sampling'.

First, we introduce the class of model within which we can apply SMC and discuss a direct approach to Bayesian inference in this context. This motivates our sections by yielding

a rephrasing of the challenges presented by equation 2.2.

3.1 The Markovian model

The SMC methods we present require that we model the data in such a way that employs the idea of a Markov chain, as described in chapter 2. Working with discrete time, we consider a collection of hidden states $\{x_t : t \in \mathbb{N}\}$, which, for example, might represent the true population of the population size at time t. In addition, recall that the evolution of the Markov chain is governed by a system process defined by the density $p(x_t|x_{t-1})$. Note that, in our population analysis, we use second order Markov processes with densities $p(x_t|x_{t-1}, x_{t-1})$, which we briefly discuss later.

Our model must include a prior distribution $p(x_0)$, which allows us to simulate a potential trajectory of the population. Recall that the remaining step is to model the noise inherent in the observation of the hidden states, which is done by defining a new density $p(y_t|x_t)$. This 'observation process' describes the process of observing y_t at when the population is in the state x_t.

Our aim is to try and estimate the most likely sequence of states $x_{0:t}$ given a set of observations $y_{1:t}$. This is done by finding a sample whose density approximates the posterior distribution $p(x_{0:t}|y_{1:t})$. In practice, this is often done sequentially. Suppose we make a new observation y_{t+1}, then, we wish to find an approximation of the posterior $p(x_{t+1}|y_{1:t}, y_{t+1}) = p(x_{t+1}|y_{1:t+1})$, which represents our new beliefs about the state of the system given the new information.

The above process can be described mathematically and naturally fits inside the Bayesian paradigm due to the sequential nature of the way we update our beliefs. In order to clearly

see how the challenge of computing the integral 2.2 manifests itself in this setting, we look at the following form for the posterior distributions given by Bayes' theorem.

$$p(x_{0:t}|y_{1:t}) = \frac{p(y_{1:t}|x_{0:t})p(x_{0:t})}{\int p(y_{1:t}|x_{0:t})p(x_{0:t})\, d_{x_{0:t}}}. \tag{3.1}$$

[2]. Estimating $p(x_{0:t}|y_{1:t})$ is known as smoothed inference, by virtue of$p(x_{1:t}|y_{1:t})$ also being known as the smoothing distribution [18]. To construct the above, we consider the recursive formulae:

$$Prediction : p(x_t|y_{1:t-1}) = \int p(x_t|x_{t-1})p(x_{t-1}|y_{1:t-1})\, dx_{t-1} \tag{3.2}$$

$$Updating : p(x_t|y_{1:t}) = \frac{p(y_t|x_t)p(x_t|y_{1:t-1})}{\int p(y_t|x_t)p(x_t|y_{1:t-1})\, dx_t} \tag{3.3}$$

[2]. From this, we may also consider filtered inference. In this case we focus on the current state of the system, as opposed to the entire trajectory. Appropriately, $p(x_t|y_{1:t})$ is also known as the filtering distribution [18]. These formulae make clear how a direct approach gives rise to the problem of complex integration as discussed in section 2.1. Next, we look at how SMC methods address this issue. First of all, we introduce importance sampling, which forms the basis of all of our methods. A minor modification yields a second method, sequential importance sampling, which makes importance sampling computationally feasible for recursive estimation [2].

3.2 Importance sampling

In order to avoid calculating 2.2, SMC methods attempt to create a sample whose density approximates the posterior distribution. The elements of this sample are often referred to as particles from which the alternative term for SMC methods, *particle filters*, is derived. In an ideal world we would sample from $p(x_{0:t}|y_{1:t})$ directly, but it is rarely feasible to do this efficiently [2]. An alternative plan is to use 'importance sampling', where we introduce a function $\pi(x_{0:t}|y_{1:t})$, known as the importance function, which we can from sample instead. The only restriction we place on π is that it must contain the support of $p(x_{0:t}|y_{1:t})$, that is

$$\{x_{0:t}|p(x_{0:t}|y_{1:t}) > 0\} \subseteq \{x_{0:t}|\pi(x_{0:t}|y_{1:t}) > 0\} \tag{3.4}$$

for all feasible $y_{1:t}$. This condition is required to avoid degeneracy in the definition of the *importance weight* $w(x_{0:t})$:

$$w(x_{0:t}) = \frac{p(x_{0:t}|y_{1:t})}{\pi(x_{0:t}|y_{1:t})}. \tag{3.5}$$

The process of weighting is a central theme throughout all of the methods we discuss. In order to understand the motivation for this, suppose you wanted to compute the expected value of some function f_t of $x_{0:t}$ with respect to the posterior distribution. Elementary statistics tells us if you wished to do this analytically you would need to compute

$$\frac{\int f_t(x_{0:t})p(x_{0:t}|y_{1:t})\,dx_{0:t}}{\int p(x_{0:t}|y_{1:t})\,dx_{0:t}}, \tag{3.6}$$

which may be infeasible. However, by simple algebraic manipulation we see the substitution suggested by 3.5 yields:

$$\frac{\int f_t(x_{0:t})w(x_{0:t})\pi(x_{0:t}|y_{1:t})\,dx_{0:t}}{\int w(x_{0:t})\pi(x_{0:t}|y_{1:t})\,dx_{0:t}}, \tag{3.7}$$

which can be estimated numerically by sampling from $\pi(x_{0:t}|y_{1:t})$ and using standard methods [2]. This highlights the immense value of introducing an importance function to generate weights. However, it is not evident from the above of an efficient method to calculate $w(x_{0:t+1})$ from $w(x_{0:t})$. Thus, we are required to recompute weights for the entire trajectory at each time step. This is computationally undesirable and motivates our next method, sequential importance sampling, which proposes a modification to bypass the need for repeated calculation.

3.3 Sequential importance sampling

Sequential importance sampling (SIS) is a Monte Carlo method that is almost identical to importance sampling, but modified to drastically increase efficiency for recursive estimation. Firstly, recall the importance function $\pi(x_{0:t}|y_{1:t})$. In SIS we choose π by making the assumption that is can be calculated recursively using the formula:

$$\pi(x_{0:t}|y_{1:t}) = \pi(x_0)\prod_{k=1}^{t}\pi(x_k|x_{0:k-1}.y_{1:k}). \tag{3.8}$$

[2]. This small modification can be of great benefit. Specifically, we see that

$$w(x_{0:t}) \propto w(x_{0:t-1})\frac{p(y_t|x_t)p(x_t|x_{t-1})}{\pi(x_t|x_{0:t-1}, y_{1:t})}. \tag{3.9}$$

[2]. This recursive form minimises computation, and includes familiar terms such as the likelihood $p(y_t|x_t)$ and system process equation $p(x_t|x_{t-1})$ making it easy to implement.

This concludes our theoretical introduction of SIS, and now we move on to clarify the exact process the implied by the above.

3.4 The process

The theory we have presented gives a clear mathematical solution to the problem at hand. However, in this brief section we aim to translate it in such a way that the reader gains a concrete idea of the step by step inferential process it implies.

The process of sampling from the importance function π and assigning weights are the two main ideas steps these methods. The sample we obtain from π is known as the particle swarm. Thus, our aim is for the particle swarm at time t to approximate the posterior distribution. We do this by making use of the data to assign a weight to each particle in the swarm. To summarise, this can be described in three steps:

Step 1 At time t generate a particle swarm from $\pi(x_{0:t}|y_{1:t}, x_{0:t-1})$.

Step 2 Use the data y_t to calculate a weight for each particle in the swarm using equation 3.9 in the case of SIS, and 3.5 for basic Importance Sampling.

Step 3 Use the weighted sample to estimate summary statistics of the posterior distribution, such as the mean.

This weighted sample is sufficient for making inference about the posterior distribution. However, we usually choose to normalise the weights at each stage, by dividing through by their sum, which ensures they remain bounded as this process iterates.

To conclude, observe that after the second step one may resample with respect to the weights in order to create an unweighted approximation to the posterior, thus eliminating the need to record past weights. This idea brings us to our next chapter which deals a related method named the bootstrap filter (BF). This algorithm includes a resampling stage, which can be interpreted as introducing the idea of Darwinian evolution if we reimagine the particles as animals. In particular, the process of resampling simulates the process of 'survival of the fittest', where the measure of a particle's fitness is its weight. This gives rise to the terminology we use later, where particles from the swarm at $t = 0$ are considered the 'ancestors' of the particles they generate through the resampling process. In turn, each particle at time $t > 1$ is referred to as the 'descendant' of some ancestral particle. Importantly, this simulation of Darwinian evolution begins to address some of the problems SIS faces with high-dimensionality. Specifically, SIS methods can suffer from a phenomenon known as *particle depletion*, which is a central theme of later chapters.

'Even the smallest person can change the course of the future.'

'Galadriel', The Fellowship of the Ring

4

Bootstrap Filters

The bootstrap filter is a form of sequential importance sampling where we adopt the prior distribution as the importance function and include a resampling step. In this chapter we first describe the motivation for this algorithm and give a mathematical overview of the process. We initially describe the process with first order Markov processes, before going on to explain how to adapt it for the second order case relevant to our population model. Thereafter, we give our implementation of the algorithm and consider its performance. To conclude, we illustrate the limitations of the bootstrap filter with an example where we estimate both the population size and the parameter values.

4.1 Motivation

Sequential importance sampling incurs a problem which will arise as a recurrent theme throughout our study of SMC, particle depletion. What we see is that as t increases the weights become more and more skewed [2]. Eventually, one particle may accumulate all of the weight, giving us a very limited amount of information about the posterior distribution. One mathematical way to measure this is called the effective sample size (ESS), which we will meet in Chapter 6. This is a diagnostic which tells us how many observations we *essentially* have. For example, the case where one particle has accumulated all of the weight is roughly equivalent to one observation. In order to help obtain a larger effective sample size, statisticians have created the bootstrap filter.

The bootstrap filter achieves this through the process of Darwinian evolution we briefly described in Chaper 3. To recap, if think of the particles as animals, then resampling causes those with high weights to produce 'offspring'. On the other hand, particles with a low weight may become scarce or extinct. This creates less discrepancy in the weights by creating more particles which reflect the data to a similar degree. Thus, future particle samples will be diversified.

Finally, our motivation for using the prior distribution to define the importance function is that it simplifies the calculation of the weights. This will become apparent as we consider the the algorithm from a more mathematical perspective in the next section.

4.2 Mathematical overview

Mathematically, there are two stages which deserve our interest. First of all, we consider the computational ramifications of defining the importance function using the prior distribu-

tion, and secondly, we look closely at the resampling step.

First, recall equation 3.9 from Chapter 3:

$$w(x_{0:t}) \propto w(x_{0:t-1}) \frac{p(y_t|x_t)p(x_t|x_{t-1})}{\pi(x_t|x_{0:t-1}, y_{1:t})}. \tag{3.9}$$

Next, suppose we adopt the following importance function

$$\pi(x_{0:t}|y_{1:t}) = p(x_{0:t}) = p(x_0) \prod_{k=1}^{t} p(x_k|x_{k-1}), \tag{4.1}$$

which implies $\pi(x_t|x_{0:t-1}, y_{1:t}) = p(x_t|x_{t-1})$. It is then immediate that 3.9 simplifies to:

$$w(x_{0:t}) \propto w(x_{0:t-1})p(y_t|x_t). \tag{4.2}$$

Thus, in order to update the weights for the bootstrap filter, we only need to calculate the likelihood. This can be derived from the observation process.

Next, we discuss the resampling step. First, recall that the particle swarm at time t represents a weighted empirical estimate of the posterior [2]. This estimate of $p(x_{0:t}|y_{1:t})$ which we denote $\overline{P}_N(x_{0:t}|y_{1:t})$ can be expressed mathematically. For a particle swarm $x_{0:t}^{(i)}$, with normalised weights $w_t^{(i)}$, we have

$$\overline{P}_N(dx_{0:t}|y_{1:t}) = \frac{1}{N} \sum_{i=1}^{N} w_t^{(i)} \delta_{x_{0:t}^{(i)}}(dx_{0:t}), \tag{4.3}$$

where N is the number of particles and δ is the Dirac mass function given by

$$\delta_{x_{0:t}^{(i)}}(dx_{0:t}) = \begin{cases} 0 & x_{0:t} \notin dx_{0:t} \\ 1 & x_{0:t} \in dx_{0:t} \end{cases} \tag{4.4}$$

[2]. Though it may appear technical, equation 4.3 determines the distribution by simply calculating the proportion of particles that lie within each interval $dx_{0:t}$. In the bootstrap filter, we modify this approach so that we estimate the posterior by replacing the weights with a variable $N_t^{(i)}$, such that $\sum_{i=1}^{N} N_t^{(i)} = N$. These determine the number of offspring of particle $x_{0:t}^{(i)}$. Thus, we have

$$\overline{P}_N(dx_{0:t}|y_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} N_t^{(i)} \delta_{x_{0:t}^{(i)}}(dx_{0:t}) \tag{4.5}$$

[2]. Observe that this construction allows us to simulate particles becoming extinct, since if $N_t^{(i)} = 0$, then the term $x_{0:t}^{(i)}$ vanishes from the sum.

N_t can be defined in various ways, and the way we have chosen was first introduced by Gordon in 1993 [19] and described in [2]. They proposed to sample from $\{x_{0:t}^{(i)}|1, ..., N\}$ with respect to the multinomial distribution parameterised by the weights $w_t^{(i)}$. This gives a sample of surviving particles approximately distributed according to the weights. However, in our implementation we sample integers from $\{1, ..., N\}$ as opposed to particles. This improves computational efficiency and achieves the same effect.

On a final note, observe that resample according to the weights effectively 'stores' them in the sample, since a sufficiently large sample approximately reflects the distribution of the weights. This is computationally convenient because it alleviates the need commit the

weights to memory for use when we come to calculate the next batch.

We now proceed to the next section where we first present the algorithm in plain English. We hope this will serve as a reference point for the reader when trying to understand the more technical R code in section4.4.

4.3 The algorithm

The following is a general version of the algorithm based on that described in [2], which we will then adapt this for our analysis of the Redhead.

Step 1: Initialisation $(t = 0)$

- Set $N \in \mathbb{N}$ to be the desired number of particles.
- Set $T \in \mathbb{N}$ to be the desired runtime.
- For $i = 1, ..., N$, sample $\overline{x}_0^{(i)} \sim p(x_0)$.
- Set $t = 1$.

Step 2: Importance Sampling Step

- For $i = 1, ..., N$, sample $\overline{x}_t^{(i)} \sim p(x_t|x_{t-1}^{(i)})$.
- For $i = 1, ..., N$, assign $\overline{x}_t^{(i)}$ the importance weight

$$\overline{w}_t^{(i)} \leftarrow Pr(y_t|\overline{x}_t^{(i)}) \tag{4.6}$$

- For $i = 1, ..., N$, normalize the weights by setting

$$w_t^{(i)} \leftarrow \frac{\overline{w}_t^{(i)}}{\sum_{j=1}^{N} \overline{w}_t^{(j)}} \tag{4.7}$$

Step 3: Selection Step

- Construct a set S by resampling with replacement N integers from the set $\{1, ..., N\}$ such that

$$Pr(S[i] = j) = w_t^{(j)} \tag{4.8}$$

  for $i = 1, ..., N$.

- For $i = 1, ..., N$ set

$$x_t^{(i)} \leftarrow \overline{x}_t^{(S[i])} \tag{4.9}$$

- If $t = T$ then stop. Else, set $t \leftarrow t + 1$ and return to step 2.

In order to make this algorithm suitable for our application, we first let the population size N_t take on the role of x_t above. Furthermore, since we use a second order density dependant model, it is necessary to use priors to simulate the particles at both $t = 0$ and $t = 1$ during the initialisation phase. Similarly, we need the system process equation to reflect the fact we are now working with a second order Markov process, and so $p(x_t^{(i)}|x_{t-1}^{(i)})$ becomes $p(N_t|N_{t-1}^{(i)}, N_{t-2}^{(i)})$, where

$$p(N_t|N_{t-1}^{(i)}, N_{t-2}^{(i)}) = N_{t-1} \exp\left(b_0 + \sum_{i=1}^{2} b_i N_{t-i} + \sigma Z_t\right). \tag{4.10}$$

We must also consider the observation process, from which compute the likelihood for the weights. Recall from Chapter 1 that the observation process is given by

$$y_t = N_t + S_t Z_t. \tag{4.11}$$

This allows us to easily calculate the likelihoods in R since it implies $y_t \sim N(x_t, S_t^2)$, where S_t is the standard error at time t.

We are now prepared to move on to the next section and see how the above description translates in to a working implementation in R.

4.4 Implementation

The following gives example R code for a bootstrap filter which allows us to estimate the yearly population size of the Redhead between 1955 and 2015. Data along with standard errors are sourced from the USFWS report discussed in Chapter 1 [6], and the fixed parameter choices originate from [1]. The following was run in RStudio [20] on an intel dual core 2.7 GHz processor. Finally, the code for visualising the output is analogous to that in appendix A.0.2.

```
### Loading data
library(XLConnect)
data.input <- readWorksheetFromFile("duckdata.xlsx",sheet=1)
duck.N <- data.input[,16]/1000

### Model Parameters
k <- 2

### Filter Parameters
number <- 10000
endtime <- 61
```

```
12
13 ### Storage for particle swarm and correspondng parameters
14 swarm <- matrix(NA, number, endtime)
15 param <- matrix(NA, number, 2+k)
16
17 ### Population priors
18 for(t in 1:k){swarm[, t] <- rnorm(number, duck.N[t], duck.SE[t])}
19
20 ### Parameter priors
21 param[,1] <- 0.194
22 param[,2] <- 0.358
23 param[,3] <- -0.652
24 param[,4] <- 0.0775
25
26 ### The process model
27 mod.nextyear <- function(swarm2, params, num){
28   hidden <- swarm2[, 2]*exp(params[,1] + (params[,2]*swarm2[,2])+(
       params[,3]*swarm2[,1]) + params[, 4]*rnorm(num,0,1))
29   return(hidden)
30 }
31
32 ### Weights via likelihood from observation model
33 wts.L <- function(swarm1, time, dat.N, dat.SE){
```

```
  wts <- dnorm(dat.N[time], swarm1, dat.SE[time])
  wts <- wts/(sum(wts))
  return(wts)
}

### Filter loop
if(endtime > 2){
  for(t in 3:endtime){
    swarm[,t] <- mod.nextyear(swarm[, (t-2):(t-1)], param, number)
    wts <- wts.L(swarm[, t], t, duck.N, duck.SE)
    smp <- sample(1:number, number, replace = TRUE, prob=wts)
    swarm <- swarm[smp,]
  }
}
```

1955-2015: BF implementation for population estimation.

4.4.1 Results and performance

The primary focus of this section is to assess the output of our analysis and judge the performance. In particular, this will lead to further insight on the inner-workings of the BF, which will motivate some of our adaptations in chapter 5. We begin by presenting a plot of the data with error bars.

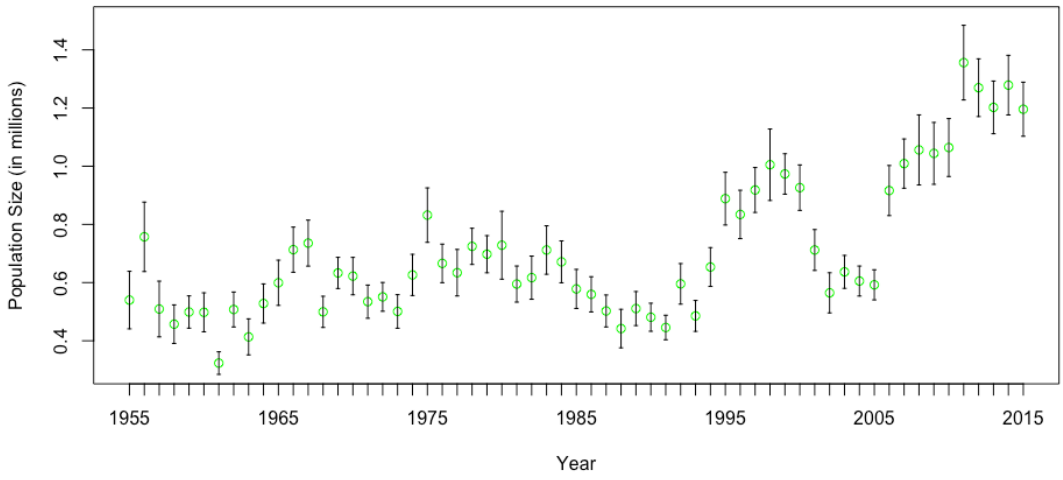


Figure 4.1: 1955-2015: data population estimates between 1955 and 2015 with error bars.

We wish to use the output of the BF to suggest the most probable trajectory of the population given the above data. In figure 4.1 we have plotted the mean of the posterior samples the BF generated, along with the corresponding 95% quantile intervals.

This is a promising output. In particular, the quantile intervals are healthily distributed about the mean and suggest a balanced and informative posterior at each stage. Importantly, we observe that this suggests significant particle depletion did not occur, and that our algorithm performed as intended. Furthermore, the trajectory suggested by our analysis appears to be a realistic interpretation of the noisy data.

To end this section, we briefly discuss runtime performance. Our simulation above used a particle swarm of $10,000$ particles, with a runtime of 1.07 seconds. One important performance diagnostic of an algorithm is how well it scales as instance size increases. Of particular importance for SMC is the scaling with respect to the number of particles used, since

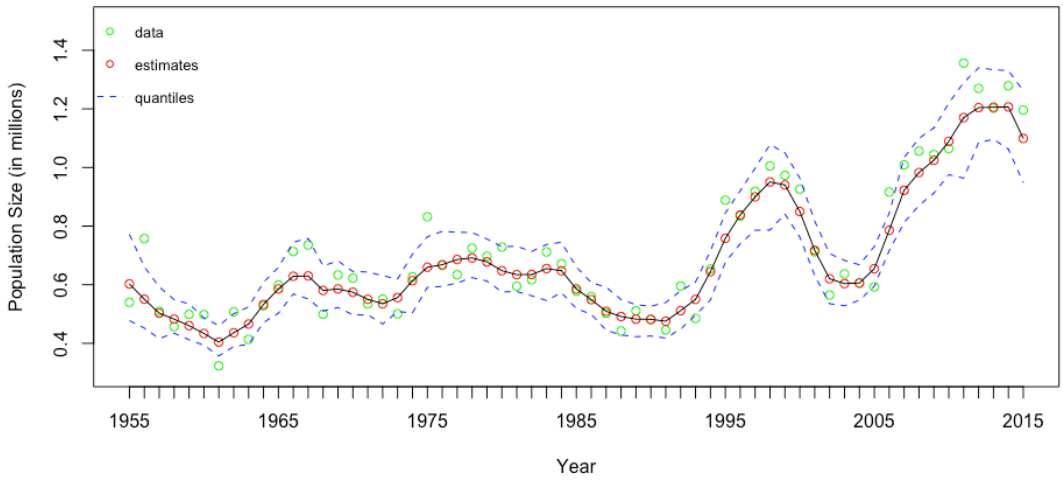


Figure 4.2: 1955-2015: bootstrap filter population estimates with 95 % quantile intervals.

for challenging high dimensional instances we may require a very large swarm. To test this, ran further simulations with 10^5, 10^6 and 2×10^6 particles.

Number of Particles	Runtime (seconds)
10^4	1
10^5	13
10^6	140
2×10^6	462

Table 4.1: Bootstrap filter runtimes as N varies.

Pleasingly, for the instances with 10^4, 10^5 and 10^6 particles the runtime increased by a factor of approximately 11 at each step, which suggest a roughly linear relationship. However, the instance with 2×10^6 particles contradicted this, with a factor increase of over 3, as

opposed to the expected 2. We conclude that within smaller instances the runtime behaves approximately linearly, however this relationship will eventually break down, perhaps due to memory shortages. To counter this, one might consider implementing elements of the algorithm in C, where there is a well known increase in speed.

4.5 Population and parameter estimation

In this section we will explore what happens when we expand the scope of the bootstrap filter and attempt to simultaneously estimate both the population size and the parameter values. This will bring us back to the central theme of particle depletion, motivating our further attempt to remedy this problem in Chaper 5.

Mathematically, we modify the bootstrap filter so that the particles $x_{0:t}^{(i)}$ are vectors, containing both the estimate of the population and a set of potential model parameters. Due to this adaptation, we must now specify prior distributions for the parameters, and the process equation must respect each particles associated parameters. The prior distributions are similar to those we used for our MCMC analysis given in table 2.1.

We omit our modifications to the code, since they were minor and analogous to aspects of the code found in Chapter 5. On running the new version of the algorithm, we obtained the population estimates given in Figure 4.3.

There are several important inferences to be made. First, notice how the quantiles have shrunk down to a point for the first few years. Looking at this from the perspective of the Darwinian evolution metaphor, then we deduce that very few particles must be the set of 'ancestors' (as defined in 3.4) to the 'descendant' particles in the final sample. We can verify this in a number of ways. One such way is to label each ancestor and attribute the same

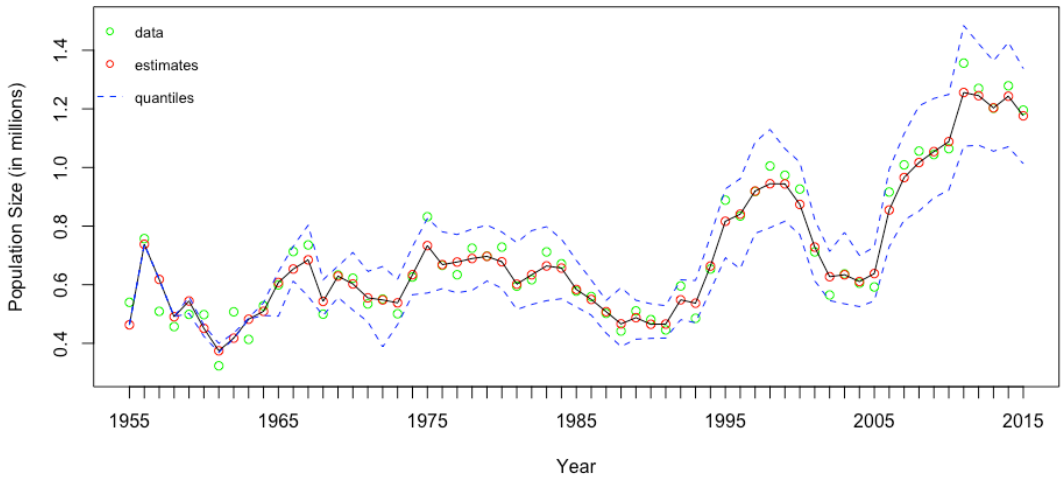


Figure 4.3: 1955-2015: bootstrap filter with population and parameter estimation.

label to all of its descendants. On implementing this, we found that the final samples from the parameter posterior distributions contained a single unique value. This is inadequate for reliable inference, and we give these parameter point estimates in Table 4.2.

Parameter	Point estimates	MCMC benchmark
b_0	-0.00179	0.0738
b_1	-0.309	0.234
b_2	0.236	-0.328
σ	0.149	0.117

Table 4.2: 1955-2015: Comparison of BF and MCMC parameter point estimates.

With the exception of σ, these values starkly contrast with the posterior estimates given by our MCMC analysis. On top of this, also note the gradual divergence of the quantiles in Figure 4.3. This implies an increasing level of uncertainty in our population estimates, which also testifies to the unsuitability of these parameters.

The most probable explanation for this outcome is quite intuitive. For some period the particle corresponding to these parameter point estimates may have described the data very well. However, this led to gaining a very high weight and quickly dominating the particle swarm. The effect of this was that later on, when it became unsuitable, no other particles were left surviving to take its place. Consequently, as time went on more and more population estimates grew further from the true value, creating the wider quantiles we see.

What we have seen here is the bootstrap filter struggling to cope with the added dimensionality inherent in estimating multiple quantities, resulting in particle depletion and poor inference. In order to overcome this problem we must adapt and use a more sophisticated approach, creating what is known in the literature as an auxillary particle filter [3].

'Your time will come. You will face the same Evil, and you will defeat it.'

'Arwen', The Fellowship of the Ring

5

Auxillary Particle Filters

The term Auxillary Particle filter (APF) describes a variety of SMC algorithms which represent novel solutions to the problem of particle depletion. In this chapter we focus on the work of J. Liu and M. West in [3]. To start, we draw from [3] to give a brief survey of the theoretical foundations. Next, to assess performance, we fit the Redhead population model over the two time frames for a variety of initial configurations of the APF.

The following APF is derived from two key adaptations of the bootstrap filter. First of all, we look at a process known as artificial evolution. This method attempts to combat particle depletion by manually diversifying the sample of parameters at each time step. This

is referred to as 'jittering' the parameter values. Naive attempts at this result in a loss of information and increased variability in our results. However, a more sophisticated approach found in [3] suggests we can carefully synthesise a method known as kernel smoothing with artificial evolution to prevent this.

Secondly, we look at an additional sampling step which we refer to as *deterministic projection resampling*. For this, we take each particle and look ahead to try and assess the degree to which it will 'agree' with future data, and then weight accordingly. This is achieved by evaluating each particle in a deterministic version of the process equation. In our model, this is created by fixing the $N(0, 1)$ random variable Z_t to be zero.

In the first section of this chapter we discuss the first of these adaptations.

5.1 Parameter diversification

5.1.1 Artificial evolution

Artificial evolution is the process of jittering parameter values in order to create diversity in the particle swarm, and thus avoid particle depletion. Recall that in this context the particles are a vector containing both population and parameter estimates. The process of jittering can be achieved by extending the process equation to act on the parameters. After which, we can run the algorithm as usual. For example, we might allow the process equation to act on the parameters by adding a zero mean normal random deviate to each. If we denote the portion of the particle vector corresponding to the parameters as $\boldsymbol{\theta}_t$, then we write this process as

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \boldsymbol{\psi}_{t+1} \tag{5.1}$$

where

$$\boldsymbol{\psi}_{t+1} \sim N(\mathbf{0}, \boldsymbol{W}_{t+1}) \tag{5.2}$$

for some variance matrix $\boldsymbol{W}_{t+1}$, the choice of which will be central in our later discussion. The inherent problem in this method is that the posterior estimates grow more and more diffuse, this is due to a 'loss of information' caused by treating fixed parameters as variable over time [3]. Of course, this is undesirable, and so now we'll take a look at kernel smoothing with shrinkage. We'll see this can be done in such a way to prevent over dispersion yet still incorporate a form of artificial evolution.

5.1.2 Kernel smoothing and shrinkage

Suppose at time t we have a particle swarm of vectors $\boldsymbol{x}_{\boldsymbol{t}}^{(\boldsymbol{i})}$. Further, denote the subvector of each particle corresponding to the parameters by $\boldsymbol{\theta}_t^{(i)}$. With kernel smoothing we wish to modify this sample to try and reduce noise and better approximate the posterior distribution. We do this by replacing the posterior sample with one drawn from a weighted summation of appropriately chosen normal kernels (known as a *mixture*). Much of our discussion will be on how to choose the mean and variance of these kernels, though first we describe the above formally.

Using notation similiar to before, let us denote the posterior distribution at time t by $p(\boldsymbol{\theta}_t|y_{0:t})$. Then, supposing we have N particles, we assume

$$p(\boldsymbol{\theta}_{t+1}|y_{0:t}) \approx \frac{1}{N}\sum_{i=1}^{N} N(\boldsymbol{\theta}_{t+1}|\boldsymbol{m}_t^{(i)}, h^2\boldsymbol{V}_t), \tag{5.3}$$

where $N(\cdot|\boldsymbol{m}, \boldsymbol{S})$ is a multivariate normal distribution with mean $\boldsymbol{m}$ and variance matrix

$\boldsymbol{S}$, and $h > 0$ is the 'smoothing parameter' [3]. Later on, we'll see how to specify these variables appropriately.

An appropiate choice of $\boldsymbol{V}_t$ is the sample variance, however, choosing $\boldsymbol{m}_t^{(i)}$ is more involved. A natural choice of $\boldsymbol{m}_t^{(i)}$ might be to take $\boldsymbol{m}_t^{(i)} = \boldsymbol{\theta}_t^{(i)}$ [3]. However, if we calculate the variance of the above mixture in this case we get $(1 + h^2)V_t$ [3]. Unfortunately, this is larger than V_t for all $h > 0$. Thus, the resulting mixture is will be too diffuse [3]. However, in [3] it is claimed that 'shrinkage' of the kernel locations $\boldsymbol{m}_t^{(i)}$ will fix this problem. In fact, this approach dates back to earlier work of West [21]. Simply put, we shift the kernel locations a small amount towards the sample mean. This has the effect of reducing the mixtures variance slightly so that is equal to $\boldsymbol{V}_t$. Mathematically, this is done by defining $\boldsymbol{m}_t$ to be

$$\boldsymbol{m}_t^{(i)} = a\boldsymbol{\theta}_t^{(i)} + (1 - a)\overline{\boldsymbol{\theta}}_t^{(i)}, \tag{5.4}$$

where $a = \sqrt{1 - h^2}$. Furthermore, $\boldsymbol{m}_t^{(i)}$ is referred to as the prior point estimate of $\boldsymbol{\theta}_t^{(i)}$ [3].

Next, we see how a better form of artificial evolution can naturally embed in this framework and is ultimately reflected in a form similar to the above. Observe that we can rewrite the process of artificial evolution in a kernel mixture form as

$$p(\boldsymbol{\theta}_{t+1}|y_{0:t}) \approx \frac{1}{N}\sum_{i=1}^{N} N(\boldsymbol{\theta}_{t+1}|\boldsymbol{\theta}_t^{(i)}, \boldsymbol{W}_{t+1}). \tag{5.5}$$

Hence, at time t, we would sample new parameter values from the multivariate normal mixture. It is clear that each individual component of the mixture simulates the process 5.1, and hence there is an equivalence between these methods. However, this equivalence sim-

ply leads to the same problem of over dispersion. Thus, we need to incorporate the changes related to our earlier discussion.

In additional to location shrinkage, it is proved in [3] that we must introduce negative correlations correlations between $\boldsymbol{\theta_t}$ and the added noise implied by the method. In plain english, this means that as t increases, we wish to decrease the added noise. This informs our choice of h for equation 5.1.2. West and Liu show that this can be achieved with a standard technique by setting

$$\boldsymbol{W_{t+1}} = \boldsymbol{V_{t+1}}(\frac{1}{\delta} - 1) \tag{5.6}$$

where δ is known as the discount factor [3].

In the context of SMC, we are interested in $p(\boldsymbol{\theta}_{t+1}^{(i)}|\boldsymbol{\theta}_t^{(i)})$, which tells us how to update the parameters for particles at each time step. West and Liu show in [3], by utilising location shrinkage and the negative correlation described above, that $p(\boldsymbol{\theta}_{t+1}^{(i)}|\boldsymbol{\theta}_t^{(i)})$ dramatically reduces to a familiar form given by 5.1.2. Specifically, we get

$$p(\boldsymbol{\theta_{t+1}}^{(i)}|\boldsymbol{\theta_t}^{(i)}) \sim N(\boldsymbol{\theta}_{t+1}^{(i)}|\boldsymbol{m}_t^{(i)}, h^2\boldsymbol{V}_t)), \tag{5.7}$$

with $\boldsymbol{m}_t^{(i)} = a\boldsymbol{\theta}_t^{(i)} + (1-a)\overline{\boldsymbol{\theta}}_t^{(i)}$, $a = \frac{3\delta-1}{2\delta}$ and $h = \sqrt{1-a^2}$. We emphasize that if each parameter is updated in this way, there is an immediate correspondence with our original kernel mixture

$$p(\boldsymbol{\theta}_{t+1}|y_{0:t}) \approx \frac{1}{N}\sum_{i=1}^{N} N(\boldsymbol{\theta}_{t+1}|\boldsymbol{m}_t^{(i)}, h^2\boldsymbol{V}_t),$$

for the particular case where h is specified as above. Thus, we have an effective synthesis of kernel smoothing and artificial evolution.

Notice that the choice of discount factor determines both a and h. In our analysis, we

typically use a discount factor of 0.994, which results in very little jittering. However in the next chapter we suggest a method for varying this discount factor according to the current state of the particle swarm.

We conclude this section by discussing the error inherent in this process. Despite mitigating the 'loss of information' described in [3], we must recognise that when we alter the parameter values they no longer truthfully correspond to the trajectory to which they are attached. Thus, the weight associated to them by their prevalence in the particle swarm is now but an approximate measure of their suitability. Furthermore, this error will accumulate each time step. However, due to the scale of jittering used, in most instances the effect will be minor. The reader should observe that for each application there will be a balance between the error incurred and the improved accuracy due to the prevention of particle depletion.

5.2 Deterministic projection resampling

Deterministic projection resampling is an additional resampling process devised by Pitt and Shephard in 1999 [22] and is the original defining step of an 'auxiliary particle filter'. Suppose we are trying to construct an approximation to the posterior at time $t + 1$ from our current particle swarm. With this mode of particle filtering we first sample particles with probability relating their predicted future fit to the data. To achieve this in our implementation, we first alter the stochastic process equation to be

$$N_t = N_{t-1} \exp\left(b_0 + \sum_{i=1}^{k} b_i N_{t-i}\right), \tag{5.8}$$

thus removing the random component and making it deterministic. Then, we weight the projected value (which known as a *prior point estimate*) in a way similar to before - we simply calculate the likelihood based on the observation process model as seen in our implementation in Chapter 4. It is important to see that in general, the above can be seen as setting the prior point estimate to be the expectation of $x_{t+1}^{(i)}$ given $x_t^{(i)}$ and $\theta_t^{(i)}$.

The aim of this step is to try and maintain a larger number of particles which will have a significant weight in the next stage. In particular, this makes single anomalous data point have a reduced impact, since such a particle will significantly deviate from subsequent observations by definition.

Furthermore, we require a slight modification to the original resampling step to account for this change. We take a similar approach to the bootstrap filter, but we use a normalised version of the weights

$$w_{t+1}^{(i)} = \frac{p(y_{t+1}|x_{t+1}^{(i)})}{p(y_{t+1}|\mu_{t+1}^{(i)})}, \tag{5.9}$$

where $\mu_{t+1}^{(i)}$ denotes the prior point estimate of $x_{t+1}^{(i)}$. This choice of weight considers the amount of improvement the stochastic element of the process equation provided. In short, in this second resampling stage we favour the particles which improved relative to their prior point estimate.

We've now introduced the necessary theory for the next section, where we present a full description of the APF.

5.3 The algorithm

First we present a general step by step version of the APF given in [3]. However, have modified and expanded their exposition throughout, particularly to clarify the initialisation

phase. Subsequently, we'll explain how to adapt this for the Redhead case, before moving on to the real implementation in R. In the following a particle contains a the trajectory and a set of parameters. We denote the population estimates of a particle vector by $x_t^{(i)}$ and the parameter subvector as $\boldsymbol{\theta_t}^{(i)}$.

Step 1: Initialisation

- Set $t = 0$.
- Set $N \in \mathbb{N}$ to be the desired number of particles.
- Set $T \in \mathbb{N}$ to be the desired number of time steps.
- Set $\delta \in (0, 1]$ to be the desired discount factor.
- Set $a = \frac{3\delta - 1}{2\delta}$
- Set $h = \sqrt{1 - a^2}$
- For $i = 1, ..., N$, sample $\overline{x}_0^{(i)} \sim p(x_0)$.

Step 2: Prior point estimates

- For $i = 1, ..., N$ calculate and store $\mu_{t+1}^{(i)} = E(x_{t+1}|x_t, \boldsymbol{\theta}_t^{(i)})$.
- Calculate and store the vector

$$\overline{\boldsymbol{\theta}}_t = \frac{1}{N} \sum_{j=1}^{N} \boldsymbol{\theta}_t^{(i)}$$

  corresponding to the sample mean of the parameter estimates.
- For $i = 1, ..., N$ calculate and store $\boldsymbol{m}_t^{(i)} = a\boldsymbol{\theta}_t^{(i)} + (1 - a)\overline{\boldsymbol{\theta}}_t$.

Step 3: Deterministic projection resampling

- For $i = 1, ..., N$, assign $x_{t+1}^{(i)}$ the importance weight

$$\overline{w}_t^{(i)} \leftarrow Pr(y_t | \mu_{t+1}^{(i)}, \boldsymbol{m}_t^{(i)}),$$

  equal to the probability of observing y_t given population estimate $\mu_{t+1}^{(i)}$ and parameters $\boldsymbol{m}_t^{(i)}$.

- For $i = 1, ..., N$, normalise the weights by setting

$$w_t^{(i)} \leftarrow \frac{\overline{w}_t^{(i)}}{\sum_{j=1}^{N} \overline{w}_t^{(j)}}$$

- Construct a set S by resampling with replacement N integers from the set $\{1, ..., N\}$ such that

$$Pr(S[i] = j) = w_t^{(j)}$$

  for $i = 1, ..., N$.

- For $i = 1, ..., N$ set

$$x_t^{(i)} \leftarrow \overline{x}_t^{(S[i])}$$

Step 3: Kernel smoothing and shrinkage

- Compute the variance matrix for the sample of parameters and store it as $\boldsymbol{V}_t$.

- For $i = 1, ..., N$ sample a vector $\boldsymbol{\phi}$ from $N(\cdot | \boldsymbol{m}_t^{(i)}, h^2 \boldsymbol{V}_t)$ and set

$$\boldsymbol{\theta}_{t+1}^{(i)} \leftarrow \boldsymbol{\phi}$$

Step 4: Importance sampling step

- For $i = 1, ..., N$, sample $\overline{x}_t^{(i)} \sim p(x_t|x_{t-1}^{(i)})$.
- For $i = 1, ..., N$, assign $\overline{x}_t^{(i)}$ the importance weight

$$\overline{w}_t^{(i)} \leftarrow \frac{Pr(y_t|\overline{x}_t^{(i)})}{Pr(y_t|\overline{\mu}_t^{(i)})}$$

- For $i = 1, ..., N$, normalize the weights by setting

$$w_t^{(i)} \leftarrow \frac{\overline{w}_t^{(i)}}{\sum_{j=1}^{N} \overline{w}_t^{(j)}}$$

Step 5: Selection step

- Construct a set S by resampling with replacement N integers from the set $\{1, ..., N\}$ such that

$$Pr(S[i] = j) = w_t^{(j)}$$

  for $i = 1, ..., N$.

- For $i = 1, ..., N$ set

$$x_t^{(i)} \leftarrow \overline{x}_t^{(S[i])}$$

- If $t = T$ then stop. Else, set $t \leftarrow t + 1$ and return to step 2.

In order to make this algorithm suitable for the Redhead, we make almost identical adaptations as to the bootstrap filter case. We recall the main points. First, we first let the population size N_t take on the role of x_t. Furthermore, due to our use of a second-order density

dependant model, it is necessary to use priors to simulate the particles at both $t = 0$ and $t = 1$ during the initialisation phase. Also, we need the system process equation above needs to reflect the fact we are now working with a second-order Markov process, and so $p(x_t^{(i)}|x_{t-1}^{(i)})$ becomes $p(N_t|N_{t-1}^{(i)}, N_{t-2}^{(i)})$. Of course, likelihoods are computed similarly via the observation process model as described in Chapter 4.

We're now ready to move on to the next section and present one of the major parts of this thesis; a translation of the above description in to a working implementation in R.

5.4 Implementation

The following gives an implementation of the auxiliary particle filter described above. In this example we aim to estimate the yearly population size of the Redhead population between 1961 and 2002, as well as obtaining parameter posterior distributions for the second-order density dependance model. This time period gives comparability with the parameter estimates of Jamieson and Brooks in [1], while later on we shall discuss the results of an implementation using the full data set available and compare with our corresponding MCMC analysis. As in Chapter 4, population data along with standard errors were provided by the USFWS report [6] discussed in Chapter 1. In addition, all parameters were given priors similar to previous analysis, as stated in table 2.1. The following was run in RStudio [20], on an intel dual core 2.7 GHz processor.

```
### Load data and packages
library(XLConnect)
library(MASS)
data.input <- readWorksheetFromFile("duckdata.xlsx",sheet=1)
```

```
duck.N <- data.input[1:61,16]/1000
duck.SE <- data.input[1:61,17]/1000

### Model parameters
k <- 2

### Filter parameters
number <- 10000
endtime <- 61
dfactor <- 0.994
a <- (3*dfactor - 1)/(2*dfactor)
h <- (1-a^2)^0.5
reps <- number

### Storage for particle swarm and model parameters
swarm <- matrix(NA, number, endtime)
par <- matrix(NA, number, 2+k)

### Population priors
for(t in 1:k){swarm[, t] <- rnorm(number, duck.N[t], duck.SE[t])}

### Parameter priors
par[,1] <- rnorm(number, 0, 1)
```

```
par[,2] <- rnorm(number, 0, 1)
par[,3] <- rnorm(number, 0, 1)

# Equivalent to IG(0.001,0.001) and truncated to plausible values
par[,4] <- sqrt(exp(runif(number, -10, 2)))

### The process model
model.nextyear <- function(swarm2, params, num){
  hidden <- swarm2[, 2]*exp(params[,1] + (params[,2]*swarm2[,2])+(
    params[,3]*swarm2[,1]) + params[, 4]*rnorm(num,0,1))
  return(hidden)
}

### Function to compute weights via likelihood
weights.L <- function(swarm1, time, data.N, data.SE){
  weights <- dnorm(data.N[time], swarm1, data.SE[time])
  weights <- weights/(sum(weights))
  return(weights)
}

weights.quo <- function(swarm1, mu.t, time, data.N, data.SE){
  w1 <- dnorm(data.N[time], swarm1, data.SE[time])
  w2 <- dnorm(data.N[time], mu.t, data.SE[time])
```

```
  weights <- w1/w2
  weights <- weights/sum(weights)
  return(weights)
}

### Prior point estimates
ppe.mu <- function(swarm2, params){
  mu <- swarm2[, 2]*exp(params[,1] + (params[,2]*swarm2[,2])+(
    params[,3]*swarm2[,1]))
  return(mu)
}

ppe.mt <- function(params, A){
  mt <- A*params
  for(i in 1:dim(params)[2]){
    mt[,i] <- mt[,i] + (1-A)*mean(params[,i])
  }
  return(mt)
}

### Parameter kernel smoothing with shrinkage
par.update <- function(params, MT, H, num){
  par.new <- matrix(NA, num, dim(params)[2])
```

```
  v <- H^2*var(params)
  for(i in 1:num){
    par.new[i,] <- mvrnorm(1, MT[i,], v)
  }
  return(par.new)
}

### Filter loop
if(endtime > 2){
  for(t in 3:endtime){
    # Step 1: Prior point estimates
    mu <- ppe.mu(swarm[,(t-2):(t-1)], par)
    mt <- ppe.mt(par, a)

    # Step 2: Sample via auxillary integers
    wt <- weights.L(mu, t, duck.N, duck.SE)
    g <- sample(1:number, reps, replace=TRUE, prob=wt)
    swarm <- swarm[g,]
    par <- par[g,]
    mu <- mu[g]
    mt <- mt[g,]

    # Step 3: Update parameters
```

```
    par <- par.update(par, mt, h, reps)

    # Step 4: Sample via system process
    swarm[, t] <- model.nextyear(swarm[, (t-2):(t-1)], par, number)

    # Step 5: Resample to simulate final weighting and adjust
    parameter order
    wt <- weights.quo(swarm[,t], mu, t, duck.N, duck.SE)
    smp <- sample(1:reps, reps, replace=TRUE, prob=wt)
    swarm <- swarm[smp,]
    par <- par[smp,]
  }
}
```

1955-2015: APF implementation for combined parameter and population estimation.

5.5 Results and Performance

The primary aim of this section is to judge and discuss the performance of the above implementation over the timeframes 1961-2002 and 1955-2015. For each time frame we will first discuss the output relative to the corresponding MCMC benchmark, and then compare the performance of the APF with the bootstrap filter. To conclude, we present a brief analysis of the runtimes for varying initial configurations of the bootstrap filter and APF.

5.5.1 1961-2002: Results

In the first instance we ran simulations with $N = 10^4$, which is a modest number of particles and allows for a relatively rapid analysis. Furthermore, through experimentation we found the discount factor $\delta = 0.994$ to be effective, and so it is used throughout unless otherwise stated. This represents a fairly small amount of jittering in the context of the range of values suggested in [3]. To begin, we consider the population estimates given by Figure 5.1.

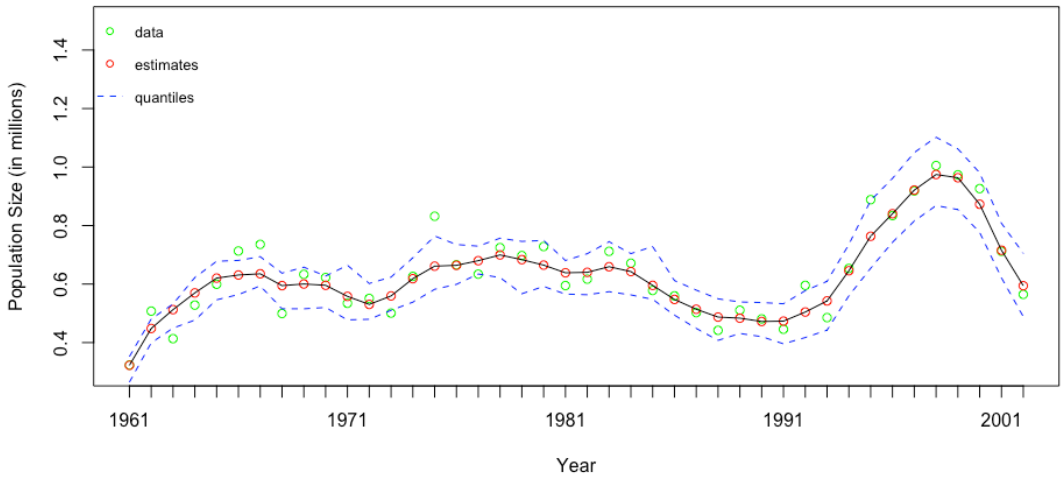


Figure 5.1: 1961-2002: APF population estimates for $N = 10^4$.

In figure 5.1 we witness a slight narrowing of the quantiles in the early years, which is suggestive of mild particle depletion. This is a less extreme example of what we saw in Chapter 4, where the quantiles had shrunk down to a point. However, in general the population estimates appear to be show a reasonable trajectory and there is no noticeable sign of a divergence in the quantiles as we saw before in the failed bootstrap case. The real test

of this first attempt will be to see how accurately we estimated the parameter values in comparison to the MCMC benchmark given in [1].

Parameter	Benchmark	Posterior mean	Posterior mode	2.5% quantile	97.5% quantile
b_0	0.194	0.179	0.164	0.0802	0.327
b_1	0.358	0.501	0.594	-0.685	1.06
b_2	-0.652	-0.790	-0.863	-1.327	0.137
σ	0.078	0.0840	0.069	0.0457	0.135

Table 5.1: 1961-2002: APF parameter posterior summaries for $N = 10,000$.

The estimates given by table 5.1 are surprisingly accurate, particularly b_0 and σ which approximately deviate by approximately 10^{-3} from the benchmark value. The other parameters deviate slightly more, though the overall shape, which we discuss in Chapter 2, is similar. We'll summarise the errors shortly, but before that we consider what can learnt from the posterior density plots.

The density plots given in figure 5.2 provide further evidence that the particle swarm is reasonably healthy - the quantiles are not too narrow to indicate particle depletion, and yet not so diffuse that precision suffers. Important qualities to observe are that each posterior is unimodal and does not show significant signs of skewness. Thus, in this case either the mean or mode given in table 5.1 would be reasonable point estimates.

In summary, given the size of the particle swarm the algorithm has performed surprisingly well, and it appears the amount of diversity generated with $\delta = 0.994$ was sufficient to prevent particle depletion. However, in order to see if it was possible to achieve significantly better accuracy, next we consider a longer runtime with $N = 2 \times 10^6$.

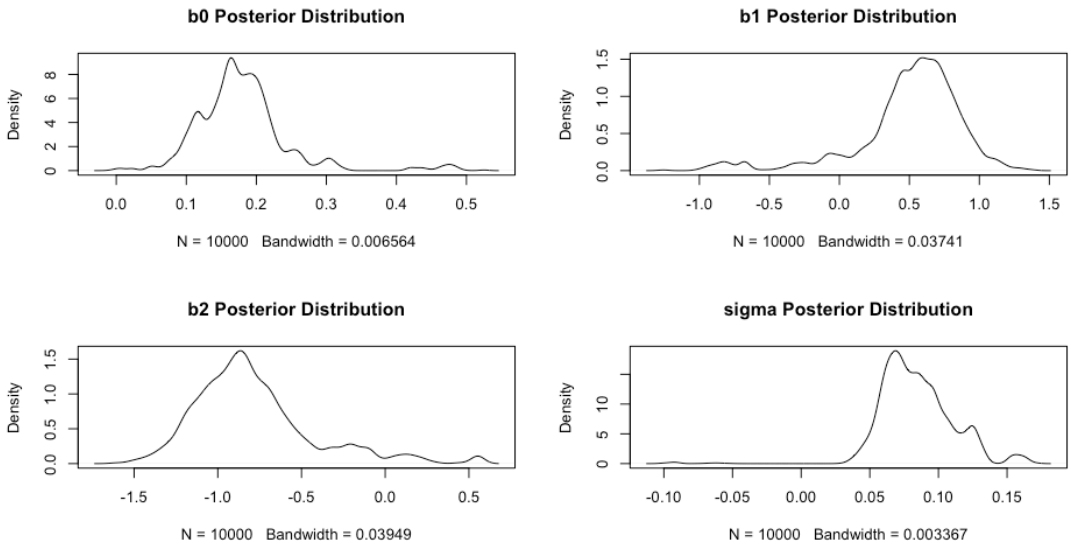


Figure 5.2: 1961-2002: APF paramater posterior densities for $N = 10^4$.

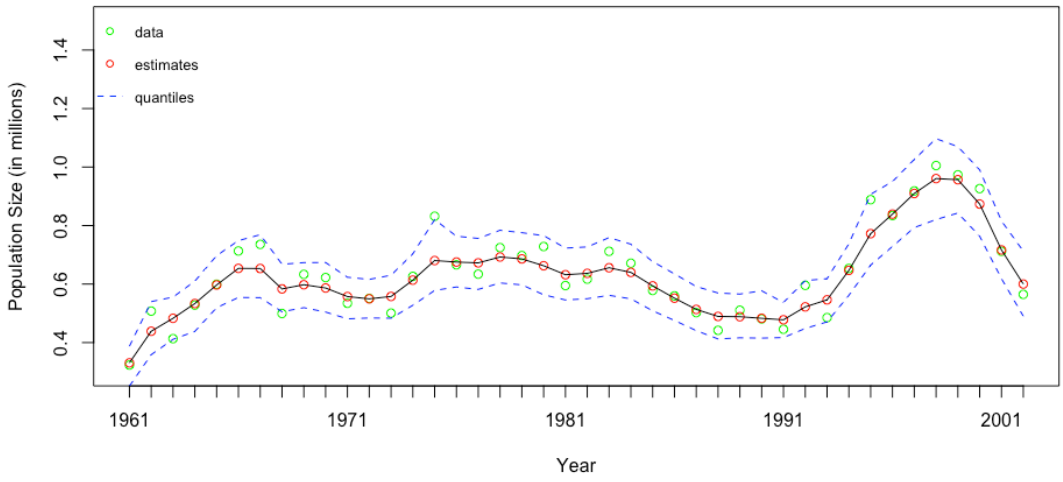


Figure 5.3: 1961-2002: APF population estimates for $N = 2 \times 10^6$.

As we would expect, the population estimates in figure 5.3 represent an improvement, with quantiles that are fairly uniformly distributed about the trajectory. However, our main interest is parameters estimation, and in the following table we compare the posterior summaries with the MCMC benchmark.

Parameter	Benchmark	Posterior mean	Posterior mode	2.5% quantile	97.5% quantile
b_0	0.194	0.196	0.185	0.043	0.376
b_1	0.358	0.343	0.304	-0.590	1.155
b_2	-0.652	-0.647	-0.673	-1.413	0.199
σ	0.078	0.044	0.095	-0.151	0.167

Table 5.2: 1961-2002: APF parameter posterior summaries for $N = 2 \times 10^6$ with the MCMC benchmark from [1] (3.s.f).

For b_0, b_1 and b_2 we have a noticeable improvement in the accuracy of the estimation compared with the case with $N = 10^4$. Interestingly, the posterior mean for σ diverged further from the benchmark of approximately 0.078. The reason for this is clear when we take a look at the density plots in figure 5.4.

In figure 5.4, we see that the posterior for σ is bimodal and contradicts the condition $\sigma \in \mathbb{R}^+$ from model 1.2.2. Given that σ has a strictly positive prior, we conclude kernel smoothing must be the cause of this phenomenon. This occurrence is logical when we consider the role of σ in the system process. Recall that σ is the coefficient of Z_t, a $N(0, 1)$ random variable. Thus, since Z_t is centred around zero, the sign of σ does not effect the expected value of the product. Indeed, the distribution we're seeing supports this reasoning, since the mode of each 'hill' has approximately the same absolute value. What remains to be explained is why the peak to the left has a significantly lower maximum density. This is

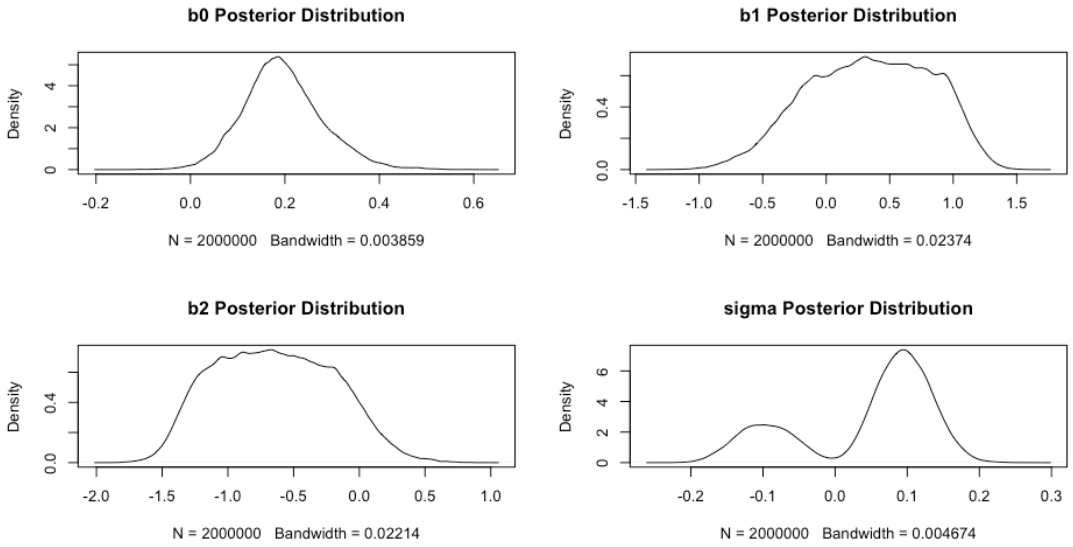


Figure 5.4: 1961-2002: APF parameter posterior densities for $N = 2 \times 10^6$.

due to the strictly positive prior for σ, and thus, given the small amount of jittering used there is a small probability of negative values being generated. Therefore, we would expect the majority of particles to be in the vicinity of the positive mode. It is likely that this phenomenon was not observed in the previous case because the small number of particles meant that is was less likely jittering would generate appropriate negative values.

By the above argument, and the definition of model 1.2.2, we believe that it is justified to discount the negative portion of the density for sigma when constructing point estimates, since this is distorting the calculation of the mean. Thus, we revise our point estimate of σ to be the mean of the positive portion of the density, which is equal to 0.0969 (3.s.f).

Before moving on, we give a summary of the above results and detail the error for each case.

Parameter	Benchmark	APF		Magnitude of error	
		$N = 10^4$	$N = 2 \times 10^6$	$N = 10^4$	$N = 2 \times 10^6$
b_0	0.194	0.179	0.196	0.0146	0.0174
b_1	0.358	0.501	0.343	0.143	0.0155
b_2	-0.652	-0.790	-0.647	0.138	0.00524
σ	0.0775	0.0840	0.0969	0.00650	0.0194

Table 5.3: Summary of APF point estimates and corresponding errors relative to MCMC benchmark over 1961-2002 (3 s.f.).

To conclude this section we wish to compare the bootstrap filter with the APF to see whether the additional complexity of the APF is justified. Similarly to before, we first consider the case with $N = 10^4$.

Parameter	Benchmark	BF	APF
b_0	0.194	0.357	0.179
b_1	0.358	-0.245	0.501
b_2	-0.652	-0.303	-0.790
σ	0.0775	0.110	-0.0840

Table 5.4: (1961-2002) Comparison of MCMC benchmark, APF and bootstrap filter point estimates with $N = 10^4$ (3.s.f).

In this instance we have illustrated the substantial benefit of using the APF over the bootstrap filter. The results from the APF preserve the shape of the parameters that we expect for the ecological reasons discussed in chapter 2. In addition, the overall error is sig-

nificantly reduced and within a reasonable range given the number of particles. Next, we make the comparison when $N = 2 \times 10^6$.

Parameter	Benchmark	BF	APF
b_0	0.194	0.208	0.196
b_1	0.358	0.206	0.343
b_2	-0.652	-0.528	-0.647
σ	0.0775	0.0986	0.0969

Table 5.5: (1961-2002) Comparison of MCMC, APF and bootstrap filter point estimates with $N = 2 \times 10^6$ (3.s.f).

This test witnesses a further improvement for both methods, particularly for the bootstrap filter. For the bootstrap filter, we now see the correct pattern emerging, and it appears that particle depletion has not occurred. Indeed, on checking the number of unique particles remaining we found over 1000 - a reasonable sample size despite the considerable reduction from the starting 2×10^6 particles. Given particle depletion did not occur, is is natural to question why APF yielded better results. We conjecture that the reason for this is that the small amount of jittering implied by $\delta = 0.994$ allowed the parameter values to converge to their 'true' value by creating just enough diversity for a small amount of Darwinian evolution to occur. We refer to this in subsequent discussions as the 'local optimisation' provided by kernel smoothing. However, there is a fine balance to be observed here, since using too much jittering results in an accumulation of error as described in 5.1.2.

5.5.2 1955-2015: Results

Our objective is to briefly highlight the key differences we found over this extended time frame. To conclude, we will lend the reader some insight on how to configure these algorithms. First of all, we look at at population estimates generated with $N = 10^4$.

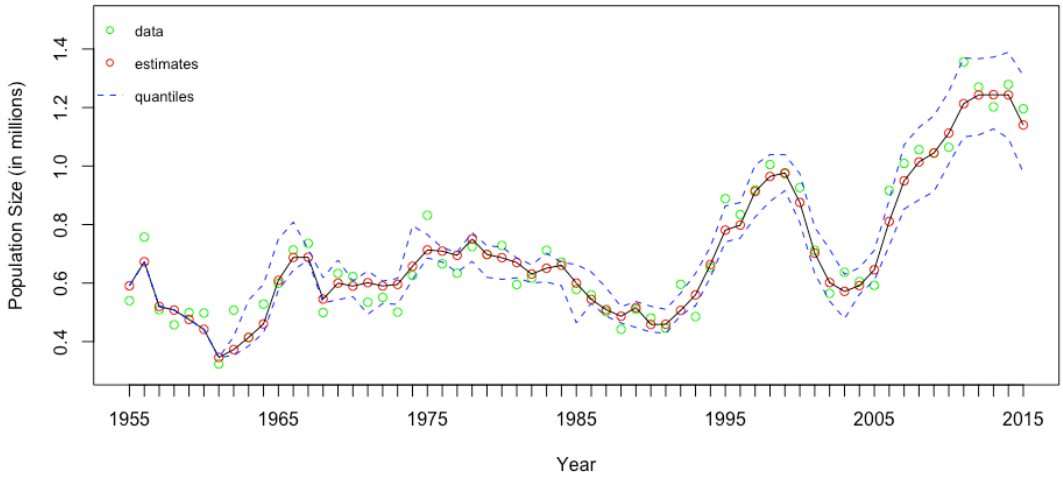


Figure 5.5: 1955-2015: APF population estimates with $N = 10^4$.

As you can see, this graph exhibits similar features to the initial bootstrap filter example. In particular, recall that a narrowing of the quantiles in the early years signals that a small number of particles are the ancestors of those that come after. It is important to see that this form of degeneracy, which we call *ancestral depletion*, is inevitable as time passes, since at each time-step there is a finite probability that one of original N lineages of particles will become extinct. However, recognising this clarifies the purpose of kernel smoothing - it exists to manually create diversity in spite of ancestral homogeneity. Moving on, we see evidence that, despite the advantages of this added diversity, there remain limitations to the

APF. The next table illustrates this with a comparison of the APF point estimates with the corresponding MCMC benchmark from chapter 2, table 2.1.

Parameter	Benchmark	Posterior mean	Posterior mode	2.5% quantile	97.5% quantile
b_0	0.0738	0.143	0.132	0.125	0.172
b_1	0.234	0.429	0.460	0.176	0.591
b_2	-0.328	-0.665	-0.716	-0.808	-0.449
σ	0.117	0.103	0.0962	0.0829	0.132

Table 5.6: 1955-2015: APF parameter posterior summaries for $N = 10^4$ with MCMC benchmark (3.s.f).

From this table we learn that while the MCMC algorithm managed to find a fit of the model radically different from previous time frame, the APF struggled to deviate significantly from the previous estimates. Under the assumptions that the MCMC results are valid and that the 5 year discrepancy in start time had a minor effect, then we offer is a simple explanation for this.

First of all, observe that the densities in figure 5.2 are quite dense, which suggests the posteriors in this case at the time step corresponding to 2002 would be similar. Then, the significant shift in posterior required to match the MCMC analysis would be challenging for this method, since convergence would be slow due to the small amount of jittering implied by $\delta = 0.994$ and the absence of many particles in the target area. However, should the few remaining particles in the area represent a huge increase in likelihood with respect to the new data then it is possible convergence would be more rapid - though it appears that this was not the case for this instance.

The above has served the purpose of giving us insight in to the workings and limitations of the algorithm, but should not be viewed as damning evidence for the APF in general.

Indeed, the shift in parameter estimates is largely due to the inadequacy of our model, due to the over simplicity that we discuss in section 4.4.1. This gives a somewhat unfair playing field for the APF, since our application here is optimised to estimate true and fixed parameters. Thus, this situation is not ideal since the MCMC analysis suggests that significantly different sets of parameters may provide the best fit for different time periods.

In the last part of this section, our focus will be to see to what extent available computational power must be increased in order to overcome the rigidity discussed above and match the estimates given by MCMC for the extended time frame. By starting with a greater number of particles, we hope we'll obtain increased variance at later stages and thus reduce the degree of ancestral depletion. This will allow for greater flexibility, since the population of particles in the vicinity of the new MCMC benchmark will likely be larger. In the following, we also consider alternative choices of δ to demonstrate the error caused by kernel smoothing.

For brevity and to aid comparison, the following table gives a collation of the MCMC benchmark and APF posterior means obtained with a variety of starting configurations.

Parameter	Benchmark	$N = 2 \times 10^6$		$N = 4 \times 10^6$	
		$\delta = 0.994$	$\delta = 0.9999$	$\delta = 0.994$	$\delta = 0.9999$
b_0	0.0738	0.0603	0.0922	0.0786	0.0601
b_1	0.234	0.440	0.177	0.331	0.236
b_2	-0.328	-0.523	-0.293	-0.438	-0.323
σ	0.117	0.0861	0.122	0.100	0.119

Table 5.7: 1955-2015: APF parameter posterior means with a variety of starting configurations.

In table 5.7 we witness a trend of increased accuracy as we progress through the configurations from left to right. The case with $N = 4 \times 10^6$ and $\delta = 0.9999$ is unique in giving results of similar precision to the best of the shorter time frame. This gives a concrete demonstration of discussion given in 5.1.2. That is, a positive correlation between an increased discount factor and reduced error is seen when particle swarm sizes are large enough to prevent significant particle depletion. In particular, it is interesting to observe that in the shorter time-frame the choice $\delta = 0.994$ yielded very accurate results whereas here we required a value significantly closer to 1. Finally, we observe these results demonstrate that the error caused by kernel smoothing accumulates over time.

Next, we discuss if, for instances with $N = 4 \times 10^6$, it is beneficial to use the APF over the simpler bootstrap filter. Our comparison is summarised in table 5.8.

Parameter	Benchmark	BF	APF
b_0	0.0783	0.0590	0.0601
b_1	0.234	0.231	0.236
b_2	-0.328	-0.303	-0.323
σ	0.117	0.118	0.119

Table 5.8: 1955-2015: APF ($\delta = 0.9999$) and bootstrap filter posterior means with $N = 4 \times 10^6$.

Table 5.8 illustrates an almost negligible improvement in all parameters except b_2. Thus, given by the added complexity of the APF, certainly is not sufficient to justify the added runtime (of approximately 4 hours) for most applications. Thus, the practitioner of these methods is presented with a challenging optimisation problem to determine the correct N, δ and method which minimises error while avoiding particle depletion. However, there is

an added dimension of complexity, since the naive approach which seeks to find the minimum value of δ which avoids particle depletion may not be best. This is due to the local optimisation that kernel smoothing provides, which we demonstrated above. This problem naturally motivates an enhancement we'll discuss in chapter in chapter 6, where we consider modifying δ in real-time in accordance to an appropriate diagnostic.

To conclude, the reader should recognise that the complexity of this situation demands at the current stage the choice of N and δ cannot be determined soley by general principles. However, the three key considerations of bias, local optimisation and particle depletion can inform an efficient experimentation to choose an appropiate configuration. To further accellerate this process for the reader, the next section briefly details the relationship between the runtimes for the different instances.

5.5.3 Runtime performance

The following table should serve as a reference point when the reader is attempting to configure her implementation with respect to her situational time constraints. Of course, the specifics of the model must also be taken in to account, so these serve only a very rough guideline. It is assumed δ has negligible impact on runtime, and thus we focus on the choice of N and method used. In addition, it is clear from the sequential structure of the algorithm that the runtime is approximately linear with respect to the number of time steps. Thus, in the interests of simplicity, we give runtimes for 1955-1966 which includes 10 projective steps (since the first two years of data act as priors). In addition, we include results for the bootstrap filter for comparison.

Method	$N = 10^4$	$N = 10^5$	$N = 10^6$	$N = 2 \times 10^6$	$N = 4 \times 10^6$
APF	7	64	706	1333	2772
BF	0.2	1	17	32	74

Table 5.9: Comparison of runtimes (in seconds) for the APF and BF over 10 projective steps.

To conclude this chapter we note the two main inferences to be made here. First, the runtime for each algorithm is approximately linear with respect to N, as would be expected. It is reasonable to take this as a general result on principle, though note that for exceptionally large N memory availability may need to be considered. Secondly, in this instance we see the APF runtimes to be increased by approximately a factor of 6 in comparison to the bootstrap filter. This will vary considerably in different contexts, but illustrates the significant increase in computation required by the APF. However, the reader should note that the runtime of the APF may significantly reduce if another programming language was used. For example, the multivariate sampling process in the APF would be dramatically faster if implemented in a language such as C.

'He's holding a thermal detonator!'

C-3PO, Return of the Jedi

6

Enhancements and Optimisation

There are a multitude of ways we might improve the APF presented in chapter 5. Recent years have witnessed continuous incremental progress and the author believes the scope for improvement likely remains significant. In this chapter we tentatively outline blueprints for a few untested improvements. We hope to inspire the reader to experiment and investigate these methods further.

To begin, we develop a natural diagnostic based on effective sample size. Our aim is to measure the degree of particle depletion at a given time. This will enable us to suggest an enhancement to address the optimisation problem inherent in configuring the parameters,

as discussed in section 5.5.2.

6.1 Swarm diagnostics

Our first diagnostic is derived from a standard statistical concept mentioned in chapter 4, the *effective sample size* (ESS). The use of ESS in importance sampling techniques at least dates back to a 1992 report by A. Kong [23]. However, in practice the implementation of ESS is not an exact science, and in subsequent years various methods of calculation have been used. [24] gives a discussion of different choices, and asserts that the most common approximation to the theoretical value described in [23] to be

$$ESS = \frac{1}{\sum_{i=1}^{N} \left(w_t^{(i)} \right)^2}, \tag{6.1}$$

where $w_t^{(i)}$ is a normalised weight at time t. With this approximation we see that if a single particle has weight approaching unity then the ESS will be close to 1, whereas a sequence of uniform weights result in an ESS close to N, as intended.

However, it is important to note the use of ESS originates as an indication of when to resample for SIS methods which resample parsimoniously. Indeed, the above does not immediately synthesise well with the intensive resampling methods presented here - since the resampling process results in a set of uniform weights. To elaborate, recall that particle depletion in methods which resample at each step creates a particle swarm consisting of only multiple copies of a single particle. In this case, the ESS would be found to be N, since we would expect each to have the same weight.

Despite this, the ESS can be of value. Essentially the above implies the ESS will be of lim-

ited value once particle depletion has *already* occurred. However, during the initial steps before too much resampling has occurred the ESS will approximately perform as intended. At later stages, we will be limited to only the can detection of rapid particle depletion over a single time-period.

However, we suggest here a more effective approach based on the ESS. For the bootstrap filter case we might be inclined be to compute the number of unique parameter values in the current swarm. Though, for the APF an enumeration of unique values will not be effective, since the mutation caused by kernel smoothing prevents a single value dominating the swarm. Instead, particle depletion in this context is characterised by an overly dense posterior sample.

We instead look at approach which exploits the idea of 'ancestors' and 'descendants' as first discussed in section 3.4. Recall from chapter 4, that the ancestor of a particle can easily be identified if we assign each particle at $t = 0$ a label from $1, ..., N$ which can be preserved throughout the resampling process. Our strategy is to then consider the ESS formula with weights that correspond to the proportion of particles from each ancestor in the swarm. It is important to note that identifying the abundance of each ancestor is a simple counting process, and is less computationally intensive than enumerating the unique values from an arbitrary set of numbers. In short, this is because *the set of possible label values is known in advance*.

Therefore, we present the following algorithm, which computes an analogue of the ESS at time t, with the assumption that at $t = 0$ we set

$$l(x_0^{(i)}) \leftarrow i \tag{6.2}$$

for $i = 1, ..., N$, where N is the number of particles and $l(x_0^{(i)})$ denotes the label of ancestor $x_0^{(i)}$.

Step 1: Initialise a zero vector C of length N.

Step 2: For $i = 1, ..., N$ do

$$C[l(x_t^{(i)})] \leftarrow C[l(x_t^{(i)})] + 1 \tag{6.3}$$

Step 3: For $i = 1, ..., N$ set

$$C[i] \rightarrow \frac{1}{N} C[i]. \tag{6.4}$$

Step 4: Return

$$\frac{1}{\sum_{i=1}^{N} (C[i])^2} \tag{6.5}$$

This algorithm assigns a weight to each ancestral particle corresponding to the proportion proportion of its 'descendants' in the current sample. We then normalise these weights in Step 3, which prepares us to apply the standard ESS formula 6.1 in Step 4. To the authors knowledge this method does not exist in current literature, and we refer to this diagnostic as the *ancestral effective sample size* (AESS). In general, at time t we can formally define the AESS of a particle sample X_t, denoted by $N_A(X_t)$, to be

$$N_A(X_t) = \frac{1}{\sum_{i=1}^{N} \left(\frac{|D_i|}{N}\right)^2} = \frac{N^2}{\sum_{i=1}^{N} |D_i|^2} \tag{6.6}$$

where D_i is the set of descendants of the particle at $t = 0$ labelled i.

In the next section we make use of this diagnostic to address the optimisation problem discussed in section 5.5.2.

6.2 Adaptive shrinkage

In chapter 5 we saw the considerable impact δ has on performance. The purpose of this section is to address the optimisation problem inherent in determining δ given N by proposing an automated procedure that determines δ automatically.

Our results were consistent with the argument that as N increases δ should approach, but not reach, one. However, it was also demonstrated that even for sample sizes sufficient for the bootstrap filter to function effectively, the local optimisation provided by a minute amount of 'jittering' may outweigh the corresponding error associated with kernel smoothing. On the basis of these facts we suggest modifying the method of kernel smoothing given in [2]. In particular, we propose two different approaches for varying the discount factor δ at each time step in accordance to the AESS diagnostic. The first is allows δ to depend on an explicit function of the AESS, while the second suggests δ depend on the change of AESS between time-steps.

The additional flexibility of this approach allows a single instance of the algorithm to adapt as necessary to the current state of the swarm, from which we derive the term *adaptive shrinkage*. Simply put, if the swarm becomes overly homogeneous we allow a greater degree of 'jittering' than if if the AESS diagnostic suggests a diverse swarm.

We now discuss the first of our two methods, whereby δ is determined as a function of the AESS. Further research is required to investigate an optimum choice of function, which may well depend on context. One key factor to take in to consideration when choosing an

appropriate function is that the AESS values form a decreasing sequence that tends to 1 as $t \to \infty$, as discussed in section 5.5.2. In light of this, we suggest the following as a starting point:

$$\delta = (1 - c)\left[\frac{N_A(X_t) - 1}{N_A(X_t) + 1}\right] + c\,. \tag{6.7}$$

where c is some value between $[0.95, 1)$. Before giving our justification for this choice, we present a plot for clarification.

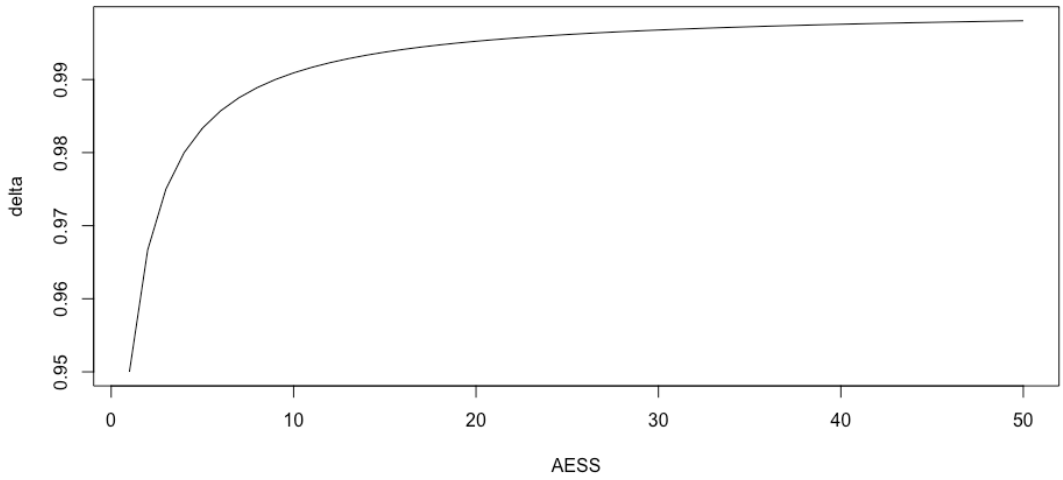


Figure 6.1: A plot of the proposed relationship between the AESS and δ with $c = 0.95$.

The above illustrates that equation 6.7 gives a range of δ values in $[c, 1)$ which can trivially be verified analytically. The lower bound for the range of values we suggest for c was motivated by [3], in which West and Liu suggest δ typically should lie in the region $[0.95, 0.99]$.

As previously noted, since the AESS tends to 1 then it immediately follows that under this function $\delta \to c$. Thus, we recommend c be equal to what the value of δ one would

use in the standard APF, for example, we would take $c = 0.994$ in the context of our earlier analysis. Such a choice causes this framework to behave asymptotically similar to the standard APF, but has an additional degree of optimisation since 'jittering', and thus bias, is reduced when it is needed less in the earlier stages.

Furthermore, notice that we opted for a function with a strictly negative second-derivative. This reflects the fact that the primary function of δ is to prevent particle depletion, and this gradient allows δ to respond quickly in the event that the AESS rapidly decreases to very low levels. On the other hand, it is less sensitive for higher values when a shift is less important.

Moving on, we'll now consider a framework where we allow δ to vary according to the change in AESS between time $t-1$ and t. In this case, we opt for a linear relationship and set

$$\delta = \frac{c_1 - c_2}{N} \left[N_A(X_{t-1}) - N_A(X_t) \right] + c_2, \tag{6.8}$$

with $c_1, c_2 \in [0.95, 1)$, $c_2 \geq c_1$ and where X_{t-1} and X_t denote the particle swarm at times $t-1$ and t respectively. Recalling that $N_A(X_{t-1}) \geq N_A(X_t)$ this relationship implies $\delta \in [c_1, c_2]$. In this case, when the AESS tends to 1 then $\delta \to c_2$ since it is clear that $N_A(X_{t-1}) - N_A(X_t) \to 0$. Furthermore, note that there are many options for c_1 and c_2 and it is suggested the reader experiment to see what best suits her application while keeping in mind the asymptotic behaviour of 6.8. To conclude, we suggest that while 6.8 only depends on the AESS for the current and previous time step, the reader may want to consider creating a more sophisticated function which makes use of a larger subset of previous AESS values.

6.3 A parallel approach with prior partitioning

In this section we tentatively suggest a new (to the author's knowledge) method for combined state and parameter estimation built on an easily parallelisable framework derived from a partition of the parameter space. The use of a partition in this way has vague conceptual links with the concept of of stratified sampling methods in SMC [25], which the reader may wish to investigate. Our approach estimates the posterior distribution of a parameter through a weighted combination of a samples. In particular, each sample has a density which approximates the geometry of the posterior density over a distinct region of the parameter space. On a cautionary note, we stress that the following is primarily intended as inspiration for future experimentation, rather than immediate usage.

The motivation for this method is that particle depletion is the result of over dispersed weights. This leads us to partition the parameter space and run separate instances of an SMC algorithm in each. Then, under the assumption that a small deviations in parameter values will cause a small deviations in projected states, we would expect less dispersion in the normalised weights. This is because in each instance of the algorithm we're only comparing the likelihood over a smaller range of state values. On a practical note, measures may need to be taken to prevent underflow for regions of the parameter space with very low likelihood.

In order to form the final posterior approximation we also include an additional weighting stage, which scales the posterior sample densities returned from the separate instances according to the relative likelihood of their regions. The advent of parallel computing dramatically increases the efficiency of this approach, since each instance can be run simultaneously.

We consider the case of estimating a single fixed parameter β of some arbitrary transition kernel. Further, we adopt the prior $\beta \sim U(0,1)$ for illustrative purposes, though the method should extend to a more general context with multiple parameters and more complex priors.

First of all, construct the auxiliary parameters $\beta_1, ..., \beta_m$ with prior distributions

$$\beta_1 \sim U\left(0, \frac{1}{m}\right)$$
$$\beta_2 \sim U\left(\frac{1}{m}, \frac{2}{m}\right)$$
$$.$$
$$.$$
$$\beta_m \sim U\left(\frac{m-1}{m}, 1\right)$$

Next, for each $i = 1, ..., m$ apply a bootstrap filter using N particles with β replaced by β_i. From each we obtain a sample $\hat{P}(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$ whose density approximates the truncated posterior distribution $p(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$.

To complete the process we need to combine the samples $\hat{P}(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$ in a meaningful way to approximate a sample from $p(\beta|y_{1:t})$. Theoretically, by the definition of a truncated distribution, we would like to weight each sample $\hat{P}(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$ according to the probability that β lies within $[\frac{i-1}{m}, \frac{i}{m}]$. However, it is likely this cannot be computed directly and so we use an alternative weighting approach. There may be a variety of ways to achieve this, but we suggest that for each $i = 1, ..., N$ and particle $(x)_i \in$

$\hat{P}(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$ we assign the weight

$$\frac{1}{N}\sum_{j=1}^{N} l(y_{1:t}|(x_{1:t})_i), \tag{6.9}$$

equal to the mean log-likelihood of the state trajectories where $(x_{1:t})_i$ is the state trajectory corresponding to particle $(x)_i$. If we denote the weighted sample of $\hat{P}(\beta|y_{1:t}, \beta \in [\frac{i-1}{m}, \frac{i}{m}])$ by $\hat{P}_w(i)$, then we claim a weighted sample which approximately corresponds to the posterior $p(\beta|y_{1:t})$ is

$$\hat{P}_w(\beta|y_{1:t}) = \bigoplus_{i=1}^{m} \hat{P}_w(i) \tag{6.10}$$

where $\bigoplus$ denotes the concatenation of samples. We may then resample according to the weights to produce an unweighted sample if desired.

In conclusion, we consider the drawbacks and benefits of this algorithm. The error incurred in this process is analogous to that incurred when approximating a function by piecewise linear maps. That is, the combination of weighted samples we use to approximate the posterior is potentially too rigid to capture the finer shape of the distribution. However, to balance this, we gain a dramatic increase in available computational power due to parallelisation. Furthermore, as previously explained it is the author's belief that the above process will help prevent particle depletion, and we encourage further theoretical and experimental analysis to test this. On a final note, observe that a natural next step would be to consider the above with the APF, and assess whether the added computational cost is justified. In the case of the APF, care must be taken to prevent kernel smoothing creating parameter values outside of the area of the parameter space each instance is working in. This can be achieved by truncating the multivariate normal distribution used to update

parameter values.

6.4 Suggestions for further reading

The few enhancements presented here are dwarfed by the abundance of innovations the reader will find in recent literature. Most notably we would encourage the reader to browse [26] for a description of how MCMC and SMC might be synthesised. Furthermore, articles on target-tracking such as [27] should help to expand the readers awareness for the scope of these methods. To explore the use of parallel computing with SMC, we recommend [28] and [29]. The first gives a description of how to exploit graphics cards to affordably perform parallel SMC on a large scale, and the second looks at efficient ways of resampling in a parallel framework. For a more comprehensive reading list, we direct the reader to a collection compiled by Doucet [30].

'Day is ended, dim my eyes,

but journey long before me lies...'

J.R.R Tolkien, Bilbo's Last Song

7

Summary

Numerous concepts have been introduced and discussed throughout, and, in summary, we present a continuous picture of our journey by recalling the key milestones.

In chapter one, recall that we were introduced to the concept of density dependance. This allowed us to appreciate the state-space model in [1] on which our later methods relied. Chapter 2 was brief but served an important purpose; we explored the widely popular MCMC methodology in order to establish the benchmarks necessary to evaluate the efficacy of our SMC implementations.

The subsequent three chapters constituted a comprehensive introduction to SMC, grad-

ually building in complexity as per the demands of our motivating example in population dynamics. In chapter 3, we began with a survey of material found in [2], introducing the theoretical basic of SMC methods. This included the approach known as importance sampling. However, by recognising that efficiency dictated the need for a recursive method to compute the weights, we introduced a special case of this known as sequential importance sampling.

Next, in chapter 4 we introduced the bootstrap filter and looked at our first demonstration of a technical implementation of SMC in R. Importantly, we saw how the generic bootstrap filter presented in [2] could be adapted for use with the Redhead population model. The bootstrap filter proved effective for state, or population, estimation, but we demonstrated its limitations in regards to parameter estimation. This moment highlighted perhaps the single most important theme throughout, *particle depletion*.

In order to address particle depletion and effectively fit the population model, Chapter 5 introduced the auxiliary particle filter by West and Liu [3], based on the algorithm by Pitt and Shepherd published in 1999 [22]. In particular, we saw how the methods of deterministic projection resampling and kernel smoothing with shrinkage help to prevent particle depletion. Thematically, kernel smoothing induced a further process of Darwinian evolution in the parameters, and we discussed mitigating a loss of information caused by treating fixed parameters as variable in time.

Our analysis of the performance of the APF was involved. Through a comparison of the APF with the bootstrap filter we justified the added complexity in some instances. However, our conclusions varied significantly depending on the initial configuration of the APF. In particular, we saw a complex relationship between N, δ and accuracy arise. This moti-

vated an enhancement given in chapter 6 which we call adaptive shrinkage. In order to implement adaptive shrinkage, it was necessary consider the effective sample size (ESS). From this, we derived a formula for the *ancestral* ESS (AESS). In chapter 6, we also suggest an ambitious new method employing parallel computing and a partition of the parameter space. Finally, to conclude this chapter, we gave recommendations for further reading.

In summation, we hope the reader is now well-equipped to appreciate the abundance of SMC literature, and to implement or improve upon these methods in her own projects and applications. Our lasting hope is that the reader finds their use fruitful, and receives a similar joy to the author in working with a remarkable set of algorithms which beautifully utilise the process of evolution which shapes our world.

A

Visualisation of Output

In this appendix we present the code which produces the visualisations and posterior summary statistics found throughout, to aid the reader who wishes to reproduce our results.

A.0.1 Data with standard errors

```
### Load data.
library(XLConnect)
data.input <- readWorksheetFromFile("duckdata.xlsx",sheet=1)
duck.N <- data.input[1:61,16]/1000
duck.SE <- data.input[1:61,17]/1000
```

```
endtime <- 61

### Initialise x-axis labels.
yrs <- rep(NA, endtime)
for(t in 1:endtime){
  if(t %% 10 == 1){
    yrs[t] <- 1954 + t
  }
}

### Original data with error bars.
plot(duck.N, col="green", xlab="Year",ylab="Population Size (in
   millions)", xaxt="n",ylim=c(0.3,1.5))
axis(1, at=1:endtime, labels=yrs)
t <- c(1:endtime)
arrows(t, duck.N-duck.SE, t, duck.N+duck.SE,length=0.02, angle=90,
   code=3)
```

A.0.2 Population estimates with quantiles

```
par(mfrow=c(1,1))

```

```
### Initialise x-axis labels.
yrs <- rep(NA, endtime)
for(t in 1:endtime){
  if(t %% 10 == 1){
    yrs[t] <- 1954 + t
  }
}

### Plot of APF population estimates with 95% quantile intervals.
lowbound <- numeric(endtime)
upbound <- numeric(endtime)
for(t in 1:endtime){
  lowbound[t] <- quantile(swarm[,t], c(0.025))
  upbound[t] <- quantile(swarm[,t], c(0.975))
}

est.pop <- numeric(endtime)
for(t in 1:endtime){
  est.pop[t] <- mean(swarm[,t])
}

plot(duck.N, col="green", xlab="Year", ylab="Population Size (in
    millions)", xaxt="n", ylim=c(0.3,1.5))
```

```
axis(1, at=1:endtime, labels=yrs)
legend("topleft", pch = c(1, 1, NA), lty=c(NA,NA,2),
       col = c("green", "red", "blue"), bty = "n", cex=0.8,
       y.intersp = 2,
       legend = c("data", "estimates", "quantiles"))
points(est.pop,col="red")
lines(est.pop, col="black")
lines(lowbound, col="blue", lty=2)
lines(upbound, col="blue",lty=2)
```

A.0.3 Parameter densities

```
### Function to compute mode of approximate density of sample.
estimate.mode <- function(x) {
  d <- density(x)
  return(d$x[which.max(d$y)])
}

### Compute parameter point estimates and quantiles.
est.max <- numeric(dim(par)[2])
est.mean <- numeric(dim(par)[2])
est.parlow <- numeric(dim(par)[2])
```

```
est.parhigh <- numeric(dim(par)[2])

for(i in 1:dim(par)[2]){
  est.max[i] <- estimate.mode(par[,i])
  est.mean[i] <- mean(par[,i])
  est.parlow[i] <- quantile(par[,i], c(0.025))
  est.parhigh[i] <- quantile(par[,i], c(0.975))
}

### Create matrix containing posterior summary statistics.
est.par <- rbind(est.max, est.mean, est.parlow, est.parhigh)
colnames(est.par) <- c("b0","b1","b2","sigma")
rownames(est.par) <- c("Posterior Maximum", "Posterior Mean", "2.5%
    Quantile", "97.5% Quantile")
print(est.par)
par(mfrow=c(2,2))

### Plot parameter posterior densities.
plot(density(par[,1]), main="b0 Posterior Distribution")
plot(density(par[,2]), main="b1 Posterior Distribution")
plot(density(par[,3]), main="b2 Posterior Distribution")
plot(density(par[,4]), main="sigma Posterior Distribution")
```

B

Data from *Trends in Duck Breeding Populations 1955-2015*

On the following page we present the data on the Redhead used in our analysis, extracted from *Trends in Duck Breeding Populations 1955-2015.*

Table B.1: *Redhead population estimates with standard errors 1955-2015 (in thousands).*

Year	Population estimate	Standard error	Year	Population estimate	Standard error
1955	539.9	98.9	1986	559.6	60.5
1956	757.3	119.3	1987	502.4	54.9
1957	509.1	95.7	1988	441.9	66.2
1958	457.1	66.2	1989	510.7	58.5
1959	498.8	55.5	1990	480.9	48.2
1960	497.8	67.0	1991	445.6	42.1
1961	323.3	38.8	1992	595.6	69.7
1962	507.5	60.0	1993	485.4	53.1
1963	413.4	61.9	1994	653.5	66.7
1964	528.1	67.3	1995	888.5	90.6
1965	599.3	77.7	1996	834.2	83.1
1966	713.1	77.6	1997	918.3	77.2
1967	735.7	79.0	1998	1005.1	122.9
1968	499.4	53.6	1999	973.4	69.5
1969	633.2	53.6	2000	926.3	78.1
1970	622.3	64.3	2001	712.0	70.2
1971	534.4	57.0	2002	564.8	69.0
1972	550.9	49.4	2003	636.8	56.6
1973	500.8	57.7	2004	605.3	51.5
1974	626.3	70.8	2005	592.3	51.7
1975	831.9	93.5	2006	916.3	86.1
1976	665.9	66.3	2007	1009.0	84.7
1977	634.0	79.9	2008	1056.0	120.4
1978	724.6	62.2	2009	1044.1	106.3
1979	697.5	63.8	2010	1064.2	99.5
1980	728.4	116.7	2011	1356.1	128.3
1981	594.9	62.0	2012	1269.9	99.2
1982	616.9	74.2	2013	1202.2	90.5
1983	711.9	83.3	2014	1278.7	102.5
1984	671.3	72.0	2015	1195.9	92.9
1985	578.2	67.1			

References

[1] L. E. Jamieson and S. P. Brooks. Density dependence in North American ducks. *Animal Biodiversity and Conservation*, 27.1:113–128, 2004.

[2] A. Doucet, N. De Freitas, and N. Gordon. An introduction to sequential monte carlo methods. In A. Doucet, N. De Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, chapter 1, pages 4–14. Springer, 2001.

[3] J. Liu and M. West. Combined parameter and state estimation in simulation-based filtering. In A. Doucet, N. De Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, chapter 10, pages 201–213. Springer, 2001.

[4] B. Dennis and M. L. Taper. Density Dependence in Time Series Observations of Natural Populations: Estimation and Testing. *Ecological Monographs*, 64:205–224, 1994.

[5] M.C. Custer. 13.1.11. Life History Traits and Habitat Needs of the Redhead. *Waterfowl Management Handbook*, 40, 1993.

[6] U.S. Fish and Wildlife Service. Trends in duck breeding populations 1955-2015. 2015. http://www.fws.gov/migratorybirds/pdf/surveys-and-data/Population-status/TrendsinDuckBreedingPopulations.pdf.

[7] M. A. Hixon and D. W. Johnson. Density dependence and independence. *elS*, 2009.

[8] C Hui. Carrying capacity, population equilibrium, and environment's maximal load. *Ecological Modelling*, 192:317–320, 2006.

[9] Bonner S. R. King R.A. Parker S. Brooks L.E. Jamieson V. Grosbois B.J.T. Morgan Giminez, O. and L. Thomas. WinBUGS for Population Ecologists: Bayesian Modeling Using Markov Chain Monte Carlo Methods. In *Modeling Demographic Processes in Marked Populations*, chapter 10, pages 885–918. Springer, 2009.

[10] T. Bayes and R. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, f. r. s. communicated by mr. price, in a letter to john canton, a. m. f. r. s. *Philosophical Transactions*, 53:370–418, 1763.

[11] P.M. Lee. *Bayesian Statistics: An Introduction*. Wiley, 2012.

[12] A. Doucet, N. De Freitas, and N. Gordon. *Sequential Monte Carlo Methods in Practice*. Springer, 2001.

[13] A Gelman, J.B. Carlin, H.S. Stern, D.B. Dunson, A. Vehtari, and D.B. Rubin. *Bayesian data analysis*. CRC press, 2013. Chapter 11.

[14] Bradley P. Carlin Mary Kathryn Cowles. Markov chain monte carlo convergence diagnostics: A comparative review. *Journal of the American Statistical Association*, 91(434):883–904, 1996.

[15] J. Lawrence, R. Gramacy, L. Thomas, and S. Buckland. The importance of prior choice in model selection: a density dependence example. *Methods in Ecology and Evolution*, 4:25–33, 2013.

[16] R Development Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, 2008. ISBN 3-900051-07-0.

[17] S.P Brooks and A. Gelman. General methods for monitoring convergence of iterative simulations. *Journal of Computational and Graphical Statistics*, 7.4:434–455, 1998.

[18] W. Fong, S. J. Godsill, A. Doucet, and M. West. Monte carlo smoothing with application to audio signal enhancement. *IEEE Transactions on Signal Processing*, 50(2):438–449, Feb 2002.

[19] N.J. Gordon, D.J. Salmond, and A.F.M. Smith. Novel approach to nonlinear, non-gaussian bayesian state estimation. *IEE Proceedings-F*, 140(2):104–113, 1993.

[20] RStudio Team. *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA, 2015.

[21] M. West. Modelling with mixtures (with discussion). In *Bayesian Statistics 4*, pages 503–524. Oxford University Press, 1992.

[22] M.K. Pitt and N. Shephard. Filtering via simulation: Auxiliary particle filters. *Journal of the American Statistical Association*, 94(446):590–599, 1999.

[23] A. Kong. A note on importance sampling using standardized weights (technical report 348). Department of Statistics, University of Chicago, 1992.

[24] L. Martino, V. Elvira, and F. Louzada. Effective sample size for importance sampling based on discrepancy measures.

[25] J. D. Hol, T.B. Schon, and F. Gustafsson. On resampling algorithms for particle filters. 2006.

[26] C. Andrieu, A. Doucet, and R. Holenstein. Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society*, 3:269–342, 2010.

[27] D. Salmond and N. Gordon. Particles and mixtures for tracking and guidance. In A. Doucet, N. De Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, chapter 25, pages 516–532. Springer, 2001.

[28] A. Lee et al. On the utility of graphics cards to perform massively parallel implementation of advanced monte carlo. *JCGS*, 2010.

[29] A. Lee and N.Whitely. Forest resampling for distributed SMC. *Statistical Analysis and Data Mining*, 2015.

[30] A. Doucet. Sequential Monte Carlo Methods and Particle Filters Resources. `http://www.stats.ox.ac.uk/~doucet/smc_resources.html`.