BAYESIAN MODEL COMPARISON via SEQUENTIAL MONTE CARLO

by

Yan Zhou

A thesis submitted to the University of Warwick
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

The University of Warwick

Department of Statistics

April 2014

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

First and foremost, I would like to sincerely express my deepest gratitude to my supervisors John Aston and Adam Johansen for their enthusiasm, encouragement and for introducing me to such exciting topics. The discussions and feedback during the course of this thesis were immensely appreciated. I am also grateful to them both for their trust in allowing me the freedom to follow my ideas and develop side projects.

No words can express my gratitude to my parents for all their support. Indeed this thesis would have never been possible without their constant encouragement.

To Zhou Yongjun & Han Ping

献给韩萍和周永均

DECLARATIONS

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. It has been composed by myself and has not been submitted in any previous application for any degree excepting some of the background material in Chapter 2, which was based on a dissertation previously submitted to the University of Warwick for the degree of Master of Science. This declaration confirms that this thesis is original and sole work of the author alone. Parts of this thesis have been published by the author:

Y. Zhou. *vSMC: Parallel sequential Monte Carlo in C++*. Mathematics e-print 1306.5583. ArXiv, 2013

Y. Zhou, A. M Johansen, and J. A. Aston. *Towards automatic model comparison: an adaptive sequential Monte Carlo approach*. Mathematics e-print 1303.3123. ArXiv, 2013

Y. Zhou, J. A. D. Aston, and A. M. Johansen. “Bayesian model comparison for compartmental models with applications in positron emission tomography”. *Journal of Applied Statistics* 40.5 (2013), pp. 993–1016

Yan Zhou, Adam M. Johansen, and John A. D. Aston. “Bayesian model selection via path-sampling sequential Monte Carlo”. In: *Proceedings of IEEE Statistical Signal Processing Workshop*. 2012

Yan Zhou

ABSTRACT

The sequential Monte Carlo (SMC) methods have been widely used for modern scientific computation. Bayesian model comparison has been successfully applied in many fields. Yet there have been few researches on the use of SMC for the purpose of Bayesian model comparison. This thesis studies different SMC strategies for Bayesian model computation. In addition, various extensions and refinements of existing SMC practices are proposed in this thesis. Through empirical examples, it will be shown that the SMC strategies can be applied for many realistic applications which might be difficult for Markov chain Monte Carlo (MCMC) algorithms. The extensions and refinements lead to an automatic and adaptive strategy. This strategy is able to produce accurate estimates of the Bayes factor with minimal manual tuning of algorithms.

Another advantage of SMC algorithms over MCMC algorithms is that it can be parallelized in a straightforward way. This allows the algorithms to better utilize modern computer resources. This thesis presents work on the parallel implementation of generic SMC algorithms. A C++ framework within which generic SMC algorithms can be implemented easily on parallel computers is introduced. We show that with little additional effort, the implementations using this framework can provide significant performance speedup.

1 INTRODUCTION

This thesis studies the use of sequential Monte Carlo (SMC) algorithms for the purpose of Bayesian model comparison. The main focus of the work is the performance of the Monte Carlo algorithms when they are used in the context of Bayesian model comparison. Contemporary methodologies on model selections and Monte Carlo methods for the purpose of Bayesian model comparison are reviewed. Methodologies on using SMC in this context are developed. Some extensions to as well as refinements of existing SMC practices are also presented in this work.

The performance of SMC algorithms for the purpose of Bayesian model comparison is studied empirically through various realistic models. Some theoretical results are also derived for non-standard methods. As this thesis covers a wide array of topics, one particular model, the position emission tomography (PET) compartmental model, is used as a running example for illustrating purpose throughout this thesis. This model is introduced in the next chapter. However, it shall be noted that, this thesis is not concerned with the analysis of the PET data in general. The particular model used in this thesis is chosen for a few reason. It provides a genuine model selection problem to which different methods can be applied and their performance can be compared. In the context of Bayesian model comparison, it is also considerably computationally challenging, in the sense that many widely used Monte Carlo methods might not perform well for practical use. The SMC algorithms are very well suited for this and many other realistic Bayesian model comparison problems. And the advantage of the SMC algorithm can be made more clear through such and other realistic models.

In the remainder of this chapter, the context that motivates the work of this thesis is first discussed. It is followed by a summary of notations used throughout the thesis. It is concluded with an outline of the structure of the following chapters.

1.1 CONTEXT

Model comparison and selection are problems found throughout the discipline of statistics. It can appear in different forms, such as the choice of regressors in regression analysis, or the determination of the number of components in mixture models. Often, there can be more than one model that can be potentially used to describe the data and to make predictions or for other purposes. However, some models might be better than others in the sense that the estimation and prediction based on them have smaller errors or variances, etc. Some models are simpler than others while providing comparable accuracy. In many application areas, model selection is also important for the purpose of identifying the underlying reasons of certain phenomena observed through the data. Many model selection and comparison methods have been developed throughout the history of statistics. Some of them are developed for particular classes of models while others make little assumptions of the candidate models. This thesis is more concerned with the later.

Bayesian model comparison has been studied and practiced for a long time. There are considerable computational difficulties when using this approach, as many high dimensional integrations are involved. The development of Monte Carlo algorithms has enabled the practice of Bayesian model comparison for a wide range of realistic applications. However, algorithms such as Markov chain Monte Carlo (MCMC) cannot efficiently simulate high dimensional multimodal distributions in many situations. In addition, estimators of quantities for the purpose of Bayesian model comparison, such as the Bayes factor, obtained through these algorithms are often unreliable in the sense that with manageable computational cost, the variances are often too large for practical use. In some cases, reliable and efficient estimators can be obtained, but they are often less generic as they require knowledge of the models not generally available. In this thesis, we aim to develop high performance algorithms that are both generic and reliable.

Population based algorithms have been developed in recent decades. They often prove to be more suitable than MCMC algorithms for simulating high dimensional multimodal distributions. Reliable estimators of quantities such as the Bayes factor can also be obtained through these algorithms. However, there is little literature on its application to Bayesian model comparison. This thesis presents a framework based on sequential Monte Carlo (SMC) algorithms, within which Bayesian model comparison can be carried out in a (semi-) automatic fashion while better accuracy compared to some other recent developments can be obtained. This is made possible through the use of various adaptive strategies.

This thesis also presents work on the practical implementations of SMC algorithms. Compared to MCMC, practical tools for SMC are relatively fewer. In addition, there is interest in the utilization of parallel computing for the implementation of SMC algorithms. The work presented in this thesis provides a toolbox with which researchers can implement generic SMC algorithms on parallel computers with relative ease.

1.2 NOTATIONS

Most notations used in this thesis are introduced and defined in context. A few conventions are followed throughout this thesis.

Capital Latin letters, such as X, are used to denote random variables and corresponding lower case letters, such as x, are used to denote their realizations. In the context of Markov chain, we use notations such as X^t to denote the random variable to indicate its dependency on time t. For various Monte Carlo estimators, we use notations such as $X^{(i)}$ to denote the random samples, including the case of MCMC algorithms. The difference between X^t and $X^{(i)}$ is to explicitly express that in some algorithms, not all samples from a Markov chain are used for estimation purpose. For SMC algorithms, we use $X_t^{(i)}$ to denote the particle value of the i^{th} particle at time t. For a sequence of variables, such as $X_1, \dots, X_n$, we use the notation $X_{1:n}$ to denote the sequence.

The letter $\boldsymbol{y}$ is used throughout this thesis to denote the data. The letter $\mathbb{E}$ is used to denote the expectation of random variables. And wherever appropriate, $\mathbb{E}_\pi$

is used to denote the expectation with respect to a distribution π. The letters Pr are used to denote probabilities of random events.

For a scalar function of an n-vector $\theta = (\theta_1, \dots, \theta_n)^T$, say $f(\theta)$, we use the notation, $\frac{\partial^2 f(\theta)}{\partial\theta\partial\theta^T}$ to denote the Hessian matrix, i.e., a matrix whose element at the i^{th} row and j^{th} column is $\partial f(\theta)/\partial\theta_i\theta_j$. We also use the notation $\frac{\partial f(\theta)}{\partial\theta}$ to denote the score vector whose i^{th} element is $\partial f(\theta)/\partial\theta_i$. For an m-vector function $f(\theta) = (f_1(\theta), \dots, f_m(\theta))$, we use the notation $J(f(\theta)) = \frac{\partial f(\theta)}{\partial\theta}$ to denote the Jacobian matrix whose element at the i^{th} row and j^{th} column is $\partial f_i(\theta)/\partial\theta_j$.

To avoid introducing too many notations, some notations might be reused if their meanings are clear in the context and their usage is limited to a particular section where they are relevant. These and other notations are defined when they are encountered the first time.

1.3 OUTLINE

This thesis is concerned with the methodologies of using SMC algorithms for the purpose of Bayesian model comparison and their practical implementations. It is structured as the following.

Chapter 2 introduces the positron emission tomography (PET) compartmental model. It is a realistic model that will be used as a running example throughout this thesis to demonstrate various methodologies. Work on the application of Bayesian model comparison to the PET compartmental model was published in [167].

Chapter 3 reviews some commonly used model selection methods. In particular some information-theoretic selection criteria and the Bayesian approach. By comparison, it will be shown that Bayesian model comparison is of interest for some realistic applications where its use was previously limited by the computational cost.

Chapter 4 reviews some Monte Carlo algorithms in the context of Bayesian model comparison. It will be shown that there are considerable difficulties for many problems of interest.

Chapter 5 presents a framework based on SMC that can be used for the purpose of Bayesian model comparison. In particular, various adaptive strategies will be discussed. This chapter is an extension to [168] and [169].

Chapter 6 presents a C++ library for the practical implementations of SMC algorithms. Parallel computing is of particular interest. Parts of this chapter is based on [166].

2 POSITRON EMISSION TOMOGRAPHY COMPARTMENTAL MODEL

Bayesian model comparison for the positron emission tomography (PET) compartmental model was studied before by the author. This thesis uses this realistic example for demonstration in Chapters 3 to 5. This chapter introduces the compartmental model and its application to PET. Later we will frequently refer to materials here for details of the model setting. This chapter is based on [167] by the author.

It shall be noted that, the application of Bayesian model comparison to the PET compartmental model is introduced here in a separate chapter only because it is used throughout the thesis as a demonstrating example. It is not unique to any of the following chapters. This thesis is not about the analysis of PET data or the compartmental model in general. Since this model is used for illustrating purpose only, only where demonstration and comparison of methods are appropriate it is used. Not all model selection and Monte Carlo methods reviewed in the next two chapters are applied to this model.

2.1 COMPARTMENTAL MODEL

Compartmental models are a class of models that describe systems in which some real or abstract quantity flows between different (physical or conceptual) compartments, each with its own characteristics. It is often of interest to infer both parameters that describe the dynamics of the system and the number of compartments that are required in order to adequately describe measured data within this framework. The choice of the number of compartments in the model presents a model selection problem of interest.

A compartmental system comprises a finite number of macroscopic subunits called *compartments*, each of which is assumed to contain homogeneous and well-mixed material. The compartments interact by material flowing from one compartment to another. There may be flows into one or more compartments from outside the system (inflows) and there may be flows from one or more compartments out of the system (outflows) [81]. In this thesis, linear compartmental models are considered. In these models, the rate of tracer flow from a compartment is proportional to the quantity of tracer in that compartment. In such models the flow may be parameterized by a pair of transfer coefficients, which are termed *rate constants* and may take the value zero, for each pair of compartments.

This class of models yields a set of ordinary differential equations (ODE) that describes the flow of tracer. Consider an r-compartments model. Let $\boldsymbol{f}(t)$ be the r-vector whose ith element corresponds to the concentration in the ith compartment at time t. Let $\boldsymbol{b}(t)$ be the r-vector that describes all flow into the system from outside. The ith element of $\boldsymbol{b}(t)$ is the rate of inflow into the ith compartment from the environment. The dynamics of such a model may be written as,

$$\begin{aligned}\dot{\boldsymbol{f}}(t) &= A\boldsymbol{f}(t) + \boldsymbol{b}(t),\\ \boldsymbol{f}(0) &= \boldsymbol{\xi},\end{aligned}$$

where $\boldsymbol{\xi}$ is the r-vector of initial concentrations and $\dot{\boldsymbol{f}}$ denotes the time derivative of $\boldsymbol{f}$. The matrix A is formed from the rate constants (see [67]). The solution [142, sec. 8.3.1] to this set of equations is,

$$\boldsymbol{f}(t) = e^{At}\boldsymbol{\xi} + \int_0^t e^{A(t-s)}\boldsymbol{b}(s)\ \mathrm{d}\,s,$$

where the matrix exponential $e^{At} = \sum_{k=0}^{\infty} \frac{(At)^k}{k!}$.

2.2 APPLICATION TO POSITRON EMISSION TOMOGRAPHY

Positron emission tomography (PET) is an analytical imaging technology that uses compounds labelled with positron emitting radionuclides as molecular tracers to image and measure biochemical process *in vivo*. It is one of the few methods available to neuroscientists to study biochemical processes within living brains, as methodology such as magnetic resonance imaging (MRI) is primarily only able to study effects via blood flow changes, while PET can study changes in the biochemical systems themselves. This is of considerable interest within research into diseases where biochemical changes are known to be responsible for symptomatic changes, such as in schizophrenia and other psychiatric diseases [47]. In a clinical setting, PET is now one of the most commonly used diagnostic procedures for cancer (both within and outside the brain), as fluoro-deoxyglucose ([^{18}F]-FDG, an radiotracer analogue of glucose) can be imaged. Cancer cells tend to be very metabolically active, thus requiring more glucose than surrounding cells, resulting in a greater uptake of [^{18}F]-FDG, leading to an indication of cancer location on an [^{18}F]-FDG scan [49].

In a typical molecular assay, usually a positron-labelled tracer is injected intravenously and the PET camera scans a record of positron emission as the tracer decays [127]. With all events detected by the PET camera, the time course of the tissue concentrations are reconstructed as three-dimension images [98]. The digital image so captured shows the signal integrated over small volume elements, termed *voxels*. Each voxel has a volume of the order of a few cubic millimeters. This data provides the tissue time-activity function, which is the total concentration of tracer in all tissue compartments. In the *plasma input compartmental model*, in addition to the PET data, a separate measurement of the concentration of tracer in the plasma is available. This measurement is generally assumed to be noise free (it can be measured with much greater accuracy than the signal of interest). This model is used in the current study. See [67] for the PET compartmental model in general.

There are many reasons that linear ODE models, of which the plasma input model is one, are the most commonly used in PET analysis. Perhaps most importantly, such systems have been shown to characterize PET experimental data well [103]. The amount of data available to fit the model for each voxel is relatively small (20-40 time points), and even with a three-compartments linear ODE model, the estimation of six parameters is non-trivial; it is clear that attempting to estimate

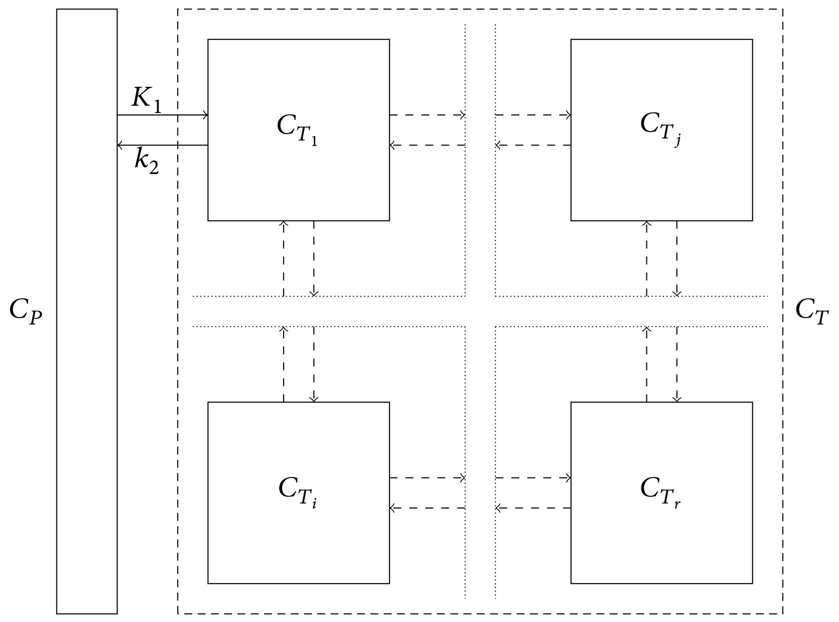


Figure 2.1 Illustration of the plasma input PET compartmental model.

the parameters of more general non-linear ODE systems robustly will be close to impossible in this setting. Furthermore, on a voxel level, which is the type of spatial analysis that is of interest here, the signal-to-noise ratio of the data is not high, making any parameter estimation difficult. Finally, as the models are estimated for every voxel in the brain (typically around a quarter of a million voxels per scan), computational consideration needs to be taken into account. Thus, linear ODE models are both experimentally useful and computationally efficient; and it is difficult to justify the additional complexity that would arise from considering more general models.

The model used in this thesis, the plasma input model as illustrated in Figure 2.1, with r tissue compartments can be written as a set of ODE,

$$\begin{aligned}\dot{\boldsymbol{C}}_{\boldsymbol{T}}(t) &= A\boldsymbol{C}_{\boldsymbol{T}}(t) + \boldsymbol{b}C_P(t)\\ C_T(t) &= \mathbf{1}^T\boldsymbol{C}_{\boldsymbol{T}}(t)\\ \boldsymbol{C}_{\boldsymbol{T}}(0) &= \mathbf{0},\end{aligned}$$

where $\boldsymbol{C_T}(t)$ is an r-vector of time-activity functions of each tissue compartment, $C_P(t)$ is the plasma time-activity function, i.e., the input function. A is the $r \times r$ state transition matrix with $A(i, j)$ being the rate constant of tracer flowing from the ith compartment into the jth compartment. $\boldsymbol{b} = (K_1, 0, \dots, 0)^T$ is an r-vector, where K_1 is the rate constant of input from the plasma into tissues. The r-vectors **1** and **0** correspond to the r-vectors of ones and zeros, respectively. The matrix A takes the form of a diagonally dominant matrix with non-positive diagonal elements and non-negative off-diagonal elements. Furthermore, A is negative semidefinite [67]. The solution to this set of ODE is,

$$C_T(t) = C_P(t) \otimes H_{TP}(t) = \int_0^t C_P(t-s) H_{TP}(s) \; \mathrm{d}s \tag{2.1}$$

$$H_{TP}(t) = \sum_{i=1}^{r} \phi_i e^{-\theta_i t}, \tag{2.2}$$

where $\otimes$ is the convolution operator and the $\phi_{1:r}$ and $\theta_{1:r}$ parameters are functions of the rate constants. There is a one-to-one mapping between the set of rate constants and the set of $\phi_{1:r}$ and $\theta_{1:r}$ parameters (see [67] for the explicit form of the mappings for various model configurations, including the ones later used in this thesis). The input function $C_P(t)$ is assumed to be nearly continuously measured. The tissue time-activity function $C_T(t)$ is measured discretely, leading to measured values of the integral of the signal over each of n consecutive, non-overlapping time intervals ending at time points $t_1, \dots, t_n$. The macro parameter of interest is the *volume of distribution*, V_D, defined by

$$V_D = \int_0^\infty H_{TP}(t) \; \mathrm{d}t = \sum_{i=1}^{r} \frac{\phi_i}{\theta_i}. \tag{2.3}$$

This corresponds to the steady state ratio of tissue concentration to plasma concentration in a constant plasma concentration regime. That is, if an injection of tracers into the plasma was made such that the plasma concentration remained constant over time, then the ratio of concentration in the tissues to the concentration in the plasma after an infinite time had passed would be exactly V_D.

It is assumed that the input is the same at all voxels of the reconstructed image. However, the model for each voxel is not assumed to be the same, and different number of compartments can be associated with each one. The model selection problem is to find the number of compartments given the data at each voxel. The compartments in the model typically can be identified with free tracer, specifically bound tracer (tracer bound to the system under investigation) and non-specifically bound tracer (tracer bound to different competing systems), indicating the role of certain chemicals within particular brain systems. In the model fitting, a "massive univariate" approach is taken with each voxel being analyzed separately. Spatial effects are neglected in this approach and voxels are assumed to be independent. This approach is common in the literature and makes the problem of dealing with a very large number of voxels feasible. However, it imposes very stringent computational requirements. About a quarter of a million voxels must be analyzed (i.e., the time series analysis must be repeated separately for each of these voxels), meaning that robustness is essential as complex model-specific characterizations and model/algorithm tuning cannot be performed on a voxel by voxel basis.

The changes of the biochemical systems are reflected in the different rates of the decay of the concentration of the compounds labelled with position emitting radionuclides. The compartments in the context of the PET compartmental model are conceptual instead of physical. In the situation where the tracers are not interacting with the brain tissues in any way, all tracers are free tracers. They are input into the brain and flows outside it without being bound to any tissues. And thus there will be only one compartment. When the biochemical process of interest does occur within the brain, some tracers will be bound to the tissues instead of flowing outside the brain. In this case, a second compartment may be observed. It is also possible that the tracers are bound to the brain tissues but not through the biochemical process of interest. In this case, a third compartment may be observed, too.

The model selection problem in this context is the choice of the number of compartments that can be best used to describe the data. As said above, identifying the existence of the second or higher order compartments is of interest. If the data support such a higher model, then it is at least possible that the biochemical process

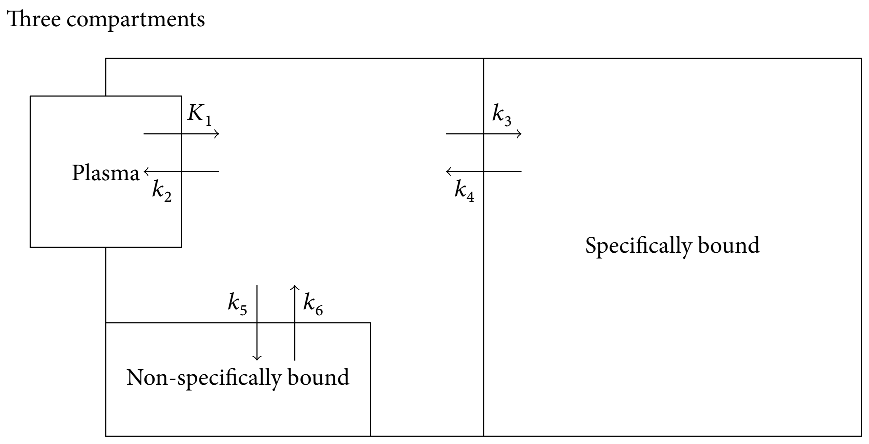


Figure 2.2 Illustration of the three-compartments plasma input PET model.

of interest is indeed happening within the brain. And thus further diagnoses through other techniques can be used to determine the cause the symptomatic changes.

2.3 SIMULATED AND REAL PET DATA

Two kinds of data are used in this thesis. The first is simulated from a three-compartments model as illustrated in Figure 2.2, with the matrix of rate constants,

$$A = \begin{bmatrix} -k_2 - k_3 - k_5 & k_4 & k_6 \\ k_3 & -k_4 & 0 \\ k_5 & 0 & -k_6 \end{bmatrix}, \tag{2.4}$$

where $k_2 = 3 \times 10^{-3}$, $k_3 = 5.5 \times 10^{-3}$, $k_4 = 1.5 \times 10^{-3}$, $k_5 = 10^{-3}$ and $k_6 = 3 \times 10^{-3}$. The rate constant of input $K_1 = 6 \times 10^{-3}$. All parameters have the unit s^{-1} except K_1, which has the unit ml s^{-1} cm^{-3} [78]. The simulated data has 32 time frames with lengths corresponding to the integration periods used in real experiments (27.5, 32.5, 2×10, 20, 6×30, 75, 11×120, 210, 5×300, 450, and 2×600, all in seconds). The plasma input function $C_P(t)$ is the same as the one obtained from real pet scans (see next paragraph). Noise is added to the synthetic data such that the noise is normally distributed with mean zero, and variance proportional to the time activities divided by the length of time frames (i.e., $C_T(t_i)/(t_i - t_{i-1})$). The noise is scaled such that the highest variance in the sequence is equal to a "noise level" variable (with the others scaled in proportion). This noise level ranges from 0.01 to 5.12, from lower than typical region of interest (ROI) analysis (in which the data is averaged over a biologically meaningful region in order to improve signal to noise ratio) to higher than the noise associated with voxel-level analysis [125]. For each noise level, 2,000 time series were simulated.

Data from a PET study using [^{11}C]diprenorphine are also used in this thesis, where [^{11}C] denotes the radioactive Carbon. The same data was previously analyzed in [125, 89]. In both studies, parameter estimation instead of model selection is of interest. The overall aim of the study was to quantify opioid receptor concentration in the brain of normal subjects allowing a baseline to be found for subsequent studies on diseases such as epilepsy. Diseases such as epilepsy tend to involve changes in brain receptor concentrations or occupancy levels either due to physical lesions within the brain or other chemically relevance differences from normal controls. Two dynamic scans from a measured [^{11}C]diprenorphine study of normal subjects, for which an plasma input function was available, were analyzed. One of them is used in this thesis. [^{11}C]diprenorphine is a tracer that binds to the opioid (pain) receptor system in the brain. The subject underwent 95 minutess dynamic [^{11}C]diprenorphine PET baseline scans on the same camera. The subjects were injected 185 MBq of [^{11}C]diprenorphine. PET scans were acquired in 3D mode on a Siemens/CTI ECAT EXACT3D PET camera, with a spatial resolution after image reconstruction of approximately 5mm. Data was reconstructed using the reprojection algorithm [98] with ramp and Colsher filters cutoff at Nyquist frequency. Reconstructed voxel size was 2.096mm$\times$2.096mm$\times$2.43mm. Acquisition was performed in listmode (event-by-event) and scans were rebinned into 32 time frames of increasing durations. The end time points of each frame is the same as in the simulated data. Frame-by-frame movement correction was performed on the PET images. Overall this resulted in images of size $128 \times 128 \times 95$ voxels, which when masked to include only brain regions, resulted in 233,054 separate time series to be analyzed. Figure 5.1 shows the estimates of V_D for this data obtained in a previous study [167]. It can be seen that the spatial structure of the data is heterogeneous. Robustness of algorithms is needed to obtain good performance for the large number of data sets.

2.4 MODELING ERROR STRUCTURES

In the scenarios considered in this thesis, linear one-, two-, and three-compartment models are considered possible; the methods could deal with other compartmental models straightforwardly, but we focus on these as they are the most interesting in the application of interest. Let $t_1, \ldots, t_n$ be the end points of the time frames at which the tissue concentrations are measured, and let $y_1, \ldots, y_n$ be the observed data, that is, the value of $C_T(t_i)$ in the ODE system. Measurement error is assumed to be white and additive with zero mean and variance proportional to activities divided by the length of time frames (i.e., $C_T(t_i)/(t_i - t_{i-1})$, the same as the one used in the simulated data). These assumptions arise from the physical characterization of the PET system of interest; alternative specifications would be possible and appropriate for other situations. Recall Equations (2.1) and (2.2) and rewrite $C_T(t)$ in terms of the parameters $\phi_{1:r}$ and $\theta_{1:r}$, for $i = 1, \ldots, n$

$$C_T(t_i; \phi_{1:r}, \theta_{1:r}) = \sum_{j=1}^{r} \phi_j \int_0^{t_i} C_P(s) e^{-\theta_j (t_j - s)} \, \mathrm{d}s$$

$$y_i = C_T(t_i; \phi_{1:r}, \theta_{1:r}) + \varepsilon_i \sqrt{\frac{C_T(t_i; \phi_{1:r}, \theta_{1:r})}{t_i - t_{i-1}}},$$

where $r = 1, 2$, or 3 is the number of tissue compartments, $t_0 = 0$, and ε_i are identically independently distributed (i.i.d.) random variables with mean zero. It is usually assumed that ε_i has a Normal distribution. It is demonstrated in [167] that there is evidence that a Student t distribution better fits the observed data.

Therefore, we consider two error structures,

$$\begin{aligned} \varepsilon_i &\sim \mathcal{N}(0, \lambda^{-1}) && \text{Normally-distributed errors} \\ \varepsilon_i &\sim \mathcal{T}(0, \tau, \nu) && t\text{-distributed errors,} \end{aligned}$$

where $\mathcal{N}(0, \lambda^{-1})$ is the Normal distribution with mean zero and precision λ, and $\mathcal{T}(0, \tau, \nu)$ is the Student t distribution with location zero, scale τ, and degrees of freedom ν. Unless stated otherwise, in the examples of this thesis, the Normally distributed error structure is used when modeling the simulated data. The Student t distributed error structure is used when modeling the real data.

3 MODEL SELECTION

Model selection is a problem found throughout statistics and related disciplines. A number of approaches has been developed through the history of statistics. We review some of the more widely used methods. We are mostly interested in methods that are generic in the sense that their usefulness is not limited to any particular class of models.

Section 3.1 reviews a few information-theoretic approaches. The most important one of them is perhaps the Akaike's information criterion (AIC; [3, 2]). A few other closely related methods are also reviewed in this section. Section 3.2 reviews the Bayesian approach to model comparison and selection. This chapter is concluded by discussions of the methods reviewed.

3.1 INFORMATION-THEORETIC APPROACH

Information theory is a discipline that covers a wide range of theories and methods that are fundamental to many scientific disciplines (see e.g., [34] for an overview). The most relevant one here is the Kullback-Leibler divergence (KLD) [102], which measures the discrepancy between two density functions. Many model selection methods are based on estimators of this measure of discrepancy.

3.1.1 *Kullback-Leibler divergence*

Assume that the distribution of data is continuous and has a density function g. Let $f(x) = f(x|\theta)$ be the density function of some continuous parametric distribution, where θ is the parameter vector. The KLD between g and f is defined by,

$$D_{\text{KL}}(g, f) = \int g(x) \log\left(\frac{g(x)}{f(x|\theta)}\right) \mathrm{d}\,x. \tag{3.1}$$

In [102] it was originally developed from information theory, as it relates the "information" lost when f is used to approximate g. The KLD is always nonnegative and equals to zero if and only if $g(x) = f(x)$ everywhere [23, sec. 6.8]. The concept can be generalized to discrete distributions and more general settings [23, sec. 2.1.3]. For the purpose of simplicity, in the remainder of this section, we will assume that distributions under discussion are continuous.

A procedure of model selection under this theme is thus finding models that have the minimum KLD between the true data generating distribution and the model distribution. There is often a set of candidate models. Each model is defined by a parametric distribution. Therefore model selection can be viewed as a two-step process. First, for each model a value of the parameter vector is found such that the KLD is minimized within this model across the parameter space. Second, models with the smallest KLD among all models are selected.

It is clear that the calculation of $D_{\text{KL}}(g, f)$ relies on the knowledge of both f and g which is unknown, as well as the value of the parameter vector θ. Rewrite Equation (3.1) as the following,

$$\begin{aligned} D_{\text{KL}}(g, f) &= \int g(x) \log g(x) \,\mathrm{d}\, x - \int g(x) \log f(x|\theta) \,\mathrm{d}\, x \\ &= \mathbb{E}_g[\log g(X)] - \mathbb{E}_g[\log f(X|\theta)]. \end{aligned} \tag{3.2}$$

The first term is a constant. Therefore, minimizing $D_{\text{KL}}(g, f)$ is equivalent to minimizing $(-\mathbb{E}_g[\log f(X|\theta)])$. The later is also called the *relative* Kullback-Leibler divergence. Let $\tilde{\theta}$ denote the value of the parameter vector that minimizes the relative KLD and $\hat{\theta}(\boldsymbol{y})$ denote an estimator of it, where $\boldsymbol{y}$ is the data generated from g. We have the minimum and estimated KLD,

$$\tilde{D}_{\text{KL}}(g, f) = \text{Constant} - \mathbb{E}_g[\log f(X|\tilde{\theta})], \tag{3.3}$$

$$\hat{D}_{\text{KL}}(g, f) = \text{Constant} - \mathbb{E}_g[\log f(X|\hat{\theta}(\boldsymbol{y}))], \tag{3.4}$$

respectively. Since $\hat{\theta}(\boldsymbol{y}) \neq \tilde{\theta}$ for (almost) all data $\boldsymbol{y}$, we have $\hat{D}_{\text{KL}}(g, f) > \tilde{D}_{\text{KL}}(g, f)$. An alternative criterion is the expected value of $\hat{D}_{\text{KL}}(g, f)$,

$$\bar{D}_{\text{KL}} = \text{Constant} - \mathbb{E}_{\hat{\theta}}\, \mathbb{E}_g[\log f(X|\hat{\theta}(\boldsymbol{y}))] \tag{3.5}$$

where the outer expectation is with respect to g and integrates out the estimated parameter $\hat{\theta}(\boldsymbol{y})$. It is again almost always larger than the minimum KLD. However, using the expected KLD as a model selection criterion allows us to select models that *on average* minimize the estimated KLD. Note that we cannot compute this term analytically since it depends on the true model g, which is assumed to be unknown. The model selection criteria discussed below rely on approximations of this quantity. These methods attempt to select the model that asymptotically minimizes, over a set of models, the expected KLD.

3.1.2 *Akaike's information criterion*

The AIC strategy is based on an observation of the relationship between the maximum likelihood estimator (MLE) and the KLD. Let $\boldsymbol{y} = (y_1, \dots, y_n)$ denotes identically independently distributed (i.i.d.) samples generated from g. Then by the Strong Law of Large Numbers (SLLN),

$$\frac{1}{n}\ell_n(\theta) \xrightarrow{\text{a.s.}} \mathbb{E}_g[\log f(Y|\theta)] \tag{3.6}$$

where $\ell_n(\theta) = \sum_{i=1}^n \log f(y_i|\theta)$ is the log-likelihood function. This suggests the use of the MLE, denoted by $\hat{\theta}$, which maximizes $\ell_n(\theta)$ as an estimator of $\tilde{\theta}$, which minimizes $D_{\text{KL}}(g, f)$. The expected KLD $\bar{D}_{\text{KL}}(g, f)$ can be approximated by empirical average $\ell_n(\hat{\theta})$, up to an additive constant that is the same for all models. However, as shown in [3], this approximation is systematically biased upward (also see [32, sec. 2.3] for some remarks on this bias). It can be shown that the bias is approximately k/n where k is the length of the parameter vector θ. This leads to the adjusted estimator of the expected relative KLD,

$$-\frac{1}{n}\ell_n(\hat{\theta}) + \frac{k}{n}. \tag{3.7}$$

In [3] it is rescaled to,

$$\text{AIC} = -2\ell_n(\hat{\theta}) + 2k \tag{3.8}$$

and the AIC strategy selects the model with the smallest value of AIC.

A more rigorous derivation of Equation (3.8) can be found in [32, sec. 2.3] and [23, sec. 6.2]. Here, we are more interested in the conditions under which this approximation is good enough for the purpose of model selection. Some remarks below are given without proof. For technical details, see the two references of the derivation of AIC.

First, the derivation of the bias term is based on a first order Taylor expansion of $\mathbb{E}_g[\ell_n(\hat{\theta})/n - \bar{D}_{\text{KL}}(g, f)]$. The accuracy is of order $o(n)$. The assumption about the parametric model f is quite minimal. Given more information about the structure of the models, more accurate estimator can be derived by using a second order expansion (discussed later in Section 3.1.3).

Second, more importantly, AIC assumes that the candidate models are close enough to the true model. When there is significant misspecification of the models, the results from using the AIC method can be misleading. Estimators of the expected KLD that are more model robust can be derived. Later, in Section 3.1.4 a more general estimator of the expected relative KLD is discussed, of which AIC is a special case.

Third, though earlier we assumed i.i.d. samples, which leads to the convergence (3.6) as a motivation of using the MLE for the estimation of expected relative KLD, this is not necessary for the application of the AIC strategy. The AIC model selection method has also been successfully used for dependent data. For example, [105] shows that AIC is efficient for selecting the order of an autoregressive process. However, AIC does assume that the model distribution is well behaved in the sense that the estimator used to evaluate the criterion is indeed close to the minimizer of the KLD.

Last but not least, AIC has the tendency of selecting more complex models in the sense that when there are multiple models with the minimum expected KLD, AIC is likely to choose the model with more parameters. Intuitively, the log-likelihood function increases linearly as the sample size grows, while the penalty term $2k$ is not affected. Therefore, the AIC strategy is likely going to select more complex models when more data becomes available. This is formally shown in [147]: When there are more than one model that minimizes the KLD, AIC does not necessarily choose the simplest model.

3.1.3 *A second order AIC*

As shown in [154], the first order approximation can perform poorly when the data size is small (compared to the number of parameters to be estimated). A second order variant is derived in the same paper and further studied by [77], which led to a criterion that is called AIC$_c$, the *corrected* AIC,

$$\text{AIC}_c = -2\ell_n(\hat{\theta}) + \frac{2nk}{n-k-1}. \tag{3.9}$$

It is clear that the additional bias correction is negligible if n is large when compared to k, as $\lim_{n\to\infty} 2nk/(n-k-1) = 2k$, which is exactly the penalty term in the original AIC formula. A rule of thumb, found in various source, is that AIC$_c$ should be used in place of AIC when $n/k \leq 40$; see e.g., [23, sec. 2.4].

AIC$_c$ is just one way to improve AIC for small sample size. In particular, it is derived in the case of a model with linear structure and Gaussian errors (see [77] and [23, sec. 6.4.1] for derivations of AIC$_c$). With other models, other forms of improved AIC can be derived. However, this form has also been used successfully in literature even in nonlinear non-Gaussian cases. For example see [160] for its application to the PET compartmental model.

Both AIC and AIC$_c$ assume the use of the MLE for the computation of the criteria. However, in many nonlinear applications, the estimator is obtained through optimization of criteria other than the likelihood function. For example, nonlinear least squares (NLS) estimation and other optimization procedures are widely used in the estimation of the PET compartmental model. Model selection criteria are computed with these estimators. These estimators are commonly used because of their ease of computation and other properties. However, the model selection results obtained this way may not be satisfactory.

For example, in [167] the model selection for the PET compartmental model using the AIC and other methods were studied. Table 3.1 shows the frequencies of

Table 3.1 Frequencies of models selected by AIC$_c$ (%) for 2,000 PET compartmental model data sets simulated from the three-compartments model.

	Noise level									
Model	0.01	0.02	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12
1	0	0.1	0.6	1.0	1.8	16.3	48.8	78.3	91.6	98.5
2	91.6	94.0	95.0	96.3	96.6	83.1	50.7	21.5	8.3	2.5
3	8.4	5.9	4.4	2.7	1.6	0.6	0.5	0.2	0.1	0

models selected by AIC$_c$ for 2,000 data sets simulated from the three-compartments model (see Section 2.3) while using the NLS estimator. It can be seen that, for data sets with small noise levels (the highest variance of the Normally distributed error added to the simulated time series, with others scaled in proportion), the AIC$_c$ method is able to select the two-compartments model with a very high frequency. Though this is not the true model that generated the data, it is very close as the third compartment is difficult to identify (see discussions in [167] and references therein). However, when the noise level increases, the method is unable to identify the second compartment.

It is possible to derive more accurate second or even higher order approximations to the expected relative KLD for some models. However, this may not be feasible for some realistic applications. For example, the PET compartmental model does not have an explicit form of the likelihood function, which may create significant technical difficulty if we want to refine the AIC approximation. More importantly, such refinement of AIC relies on assumptions about the explicit form of the true model g or one that closely imitates it. When the form of the model is drastically different from the one used to derive criteria such as AIC$_c$, poor results of model selection are likely to be obtained as we have seen here.

3.1.4 *Takeuchi's information criterion*

As stated earlier, AIC (and some of its refinements such as AIC_c) depends on the assumption that the candidate models are close to the one that generated the data. However, this might not be the case in reality. In [155], a general derivation from KLD to AIC was developed. An intermediate result indicated a selection criterion useful when there is considerable model misspecification, formulated as TIC,

$$\text{TIC} = -2\ell_n(\theta) + 2\,\text{tr}(H(\theta)K(\theta)^{-1}) \tag{3.10}$$

where $-H(\theta)$ is the expectation of the Hessian matrix and $K(\theta)$ is the variance matrix of the score vector, respectively, that is,

$$H(\theta) = -\,\mathbb{E}_g\Big[\frac{\partial^2 \log f(X|\theta)}{\partial\theta\partial\theta^T}\Big] \quad \text{and} \quad K(\theta) = \text{var}_g\Big[\frac{\partial \log f(X|\theta)}{\partial\theta}\Big], \tag{3.11}$$

provided that all differentiations and integrations exist. The expectations are taken with respect to the true data generating distribution g. Ideally TIC should be evaluated at the minimizer of the KLD, $\tilde{\theta}$. In reality, the MLE is often used to evaluate the likelihood function and various estimator of the bias correction term, $\text{tr}(H(\tilde{\theta})K(\tilde{\theta})^{-1})$, has been developed (see e.g., [32]). If the MLE is well behaved [106], then we can substitute the MLE into Equation (3.10) and use empirical averages as estimates of $\text{tr}(H(\hat{\theta})K(\hat{\theta})^{-1})$. The explicit form of the bias correction term can also be derived for some models. For example, see [23, sec. 6.6]. This allows more accurate evaluation of TIC.

Note that, since $\tilde{\theta}$ minimizes the KLD, the expectation of the score vector is a zero vector when evaluated at $\tilde{\theta}$, under suitable continuity conditions. Further, if $g(x) = f(x|\tilde{\theta})$ everywhere, then it is obvious that the above two matrices are equal and become the *Fisher information matrix*, provided the ability to exchange the order of integrations and differentiations and other regularity conditions. In this case, $\text{tr}(H(\theta)K(\theta)^{-1}) = k$, and the TIC leads to the AIC formula. It becomes clear now that AIC is an approximation of TIC in the situation where the candidate models are close to the true data generating mechanism. The TIC method does not have such assumptions and may perform considerably better than AIC in the situation of model misspecification.

Unlike the refinements of AIC such as AIC$_c$, the TIC method relies heavily on the assumption of large sample size in order to obtain accurate estimation of the bias term. It is difficult to derive small sample correction for the TIC approximation. See also the discussions in [23, sec. 6.7.8]

3.1.5 *Cross-validation*

Cross-validation has a long history in applied and theoretical statistics. It has been formalized in [51] and [151] (also see the introduction in [151] for an overview of earlier development on this method). The basic idea is to split the data into two parts. One part of the data is used for model fitting and the resulting estimates of parameters are used to predict the other part of the data. By comparing the predictions based on part of the data and the observed other part, the usefulness of the model is determined.

Formally, following [51], let $\boldsymbol{y} = (y_1, \dots, y_n)$ be the data set and $\boldsymbol{y}^t \subset \boldsymbol{y}$ be a non-empty proper subset. The sub-sample $\boldsymbol{y}^t$ is called the *training set* and its complement $\boldsymbol{y}^v = \boldsymbol{y}\backslash\boldsymbol{y}^t$ is called the *validation set*. For each model, defined by a parametric distribution with density $f(\cdot|\theta)$, a loss function is defined, say $\gamma(\cdot|\theta)$. The choice of the loss function γ is formally arbitrary. It is taken as a measurement of the fitness of the model. A commonly used one is the log-density function, $\gamma(x|\theta) = -\log f(x|\theta)$ [149]. The risk estimator of $\mathbb{E}_g[\gamma(X|\hat{\theta}(\boldsymbol{y}))]$, where $\hat{\theta}(\boldsymbol{y})$ is the estimate obtained with all data and the expectation is taken with respect to the unknown distribution g that generates the data, is obtained through averaging over the left-out data,

$$\hat{R}^v_f(\boldsymbol{y}, \boldsymbol{y}^t) = \frac{1}{|\boldsymbol{y}\backslash\boldsymbol{y}^t|} \sum_{y \in \boldsymbol{y}\backslash\boldsymbol{y}^t} \gamma(y|\hat{\theta}(\boldsymbol{y}^t)) \tag{3.12}$$

where $\hat{\theta}(\boldsymbol{y}^t)$ is the parameter estimate obtained with only the training set $\boldsymbol{y}^t$. Further, let $\boldsymbol{y}^t_1, \dots, \boldsymbol{y}^t_m$ be a sequence of non-empty proper subsets of $\boldsymbol{y}$. The cross-validation estimator of the risk is defined as,

$$\hat{R}_f^{\text{cv}}(\boldsymbol{y}, \{\boldsymbol{y}_i^t\}_{i=1}^m) = \frac{1}{m}\sum_{i=1}^m \hat{R}_f^v(\boldsymbol{y}, \boldsymbol{y}_i^t). \tag{3.13}$$

The model selection proceeds to choose the model with the smallest value of the estimated risk $\hat{R}_f^{\text{cv}}(\boldsymbol{y}, \{\boldsymbol{y}_i^t\}_{i=1}^m)$.

An alternative, as seen in [164], is called cross-validation *with voting*. A model with density f_1 is chosen over a model with density f_2 if and only if $\hat{R}_{f_1}^v(\boldsymbol{y}, \boldsymbol{y}_i^t) < \hat{R}_{f_2}^v(\boldsymbol{y}, \boldsymbol{y}_i^t)$ for a majority of the partitions of the data $\boldsymbol{y}$. When there are multiple candidate models, the same paper proposed the following procedure: For each partition of the data $\boldsymbol{y}$, and the corresponding training set $\boldsymbol{y}_i^t$ and validation set $\boldsymbol{y}_i^v = \boldsymbol{y}\backslash\boldsymbol{y}_i^t$, a model with the smallest value of $\hat{R}_f^v(\boldsymbol{y}, \boldsymbol{y}_i^t)$ is selected. Then the model selected most frequently (the most voted) among all partitions is chosen as the best model.

There are different ways to split the sample. The most commonly used is perhaps the *leave-one-out* procedure [151, 51]. In this case, training sets $\boldsymbol{y}_i^t = \boldsymbol{y}\backslash\{y_i\}$ for $i = 1, \dots, n$ are used. A more general scheme is that k observations are left out for each training set and all possible combinations are considered [144]. It is clear that $k = 1$ yields the leave-one-out procedure and for large sample size, a modest k can lead to higher computational cost as the number of possible partitions is the binomial coefficient. This is also called the *k-fold* procedure. Other procedures are also possible. For more information we refer to [150] and [75].

There are also different choices of the loss function γ. The one mentioned earlier, $\gamma(x|\theta) = -\log f(x|\theta)$, when combined with the leave-one-out procedure, leads to the estimator,

$$-\frac{1}{n}\sum_{i=1}^n \log f(y_i|\hat{\theta}(\boldsymbol{y}_i^t)) \tag{3.14}$$

where $\hat{\theta}(\boldsymbol{y}_i^t)$ is the estimate obtained using the sub-sample $\boldsymbol{y}_i^t = \boldsymbol{y}\backslash\{y_i\}$. It was shown in [150] that this is asymptotically equivalent to the AIC strategy for model selection.

An alternative loss function for linear regression models was proposed in [5]. The squared difference between the observation and the predictor is used as the risk estimator. Given a model $y_i = \boldsymbol{\beta}^T \boldsymbol{x}_i + \varepsilon_i$, and let $\hat{y}_i$ be the predictor of y_i obtained with the model fitted with all but the i^{th} observation, i.e., using the leave-one-out procedure, this leads to the PRESS statistic,

$$\text{PRESS} = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{3.15}$$

The model with the smallest PRESS value is selected. This is one of the commonly used model selection methods for regression models.

The performance of cross-validation for model selection depends on both the choice of the loss function and the partition of the sample. There is a large amount of literature on cross-validation for various model selection problems. For some models, specific choice of the function γ were proposed, for example, the PRESS statistic shown earlier and its more robust variant such as replacing the squared error by the absolute error [32, sec. 2.9]. Also as argued in the same book, the use of $\gamma(x|\theta) = -\log f(x|\theta)$ is a sensible choice for many applications, as the resulting cross-validation estimator can be interpreted as an estimator of the expected relative KLD.

The partition of the sample can influence the performance more significantly. And the procedure that minimizes the bias and variance of the risk estimator is not necessarily the same as the one that produces the best model selection results. For example, for regression models with random covariates, [22] gave examples where the best risk estimator was obtained with the leave-one-out procedure while a 10-fold cross-validation could produce more accurate model selection results. More generally, the performance depends on the asymptotic behavior of n_t/n where n_t is the size of the training set. For instance, [143] showed that for linear model selection, cross-validation is more efficient when n_t is asymptotically equal to n while using a k-fold procedure.

3.2 BAYESIAN MODEL COMPARISON

Bayes' theorem, in its simplest form is stated as below,

$$\Pr(H|\boldsymbol{y}) = \frac{\Pr(\boldsymbol{y}|H)\Pr(H)}{\Pr(\boldsymbol{y})} \tag{3.16}$$

where $\boldsymbol{y}$ is the data and H is a hypothesis. Like many other probability theories, technically Bayes' theorem merely provides a method of accounting for the uncertainty. There are different interpretations, rooted in the views of probabilities. See [20, chap. 1] and references therein for discussions on this topic. In this thesis, we are more concerned with the practical applications of the Bayesian model comparison technique, its computational difficulties, and its implementation for realistic models. More philosophical issues will not be elaborated in this thesis.

A treatment of Bayesian modeling from a decision-theoretic perspective can be found in [134]. Formal mathematical representations can also be found in [20, sec. 5.1 and sec. 6.1]. Notions of rational decisions in the context of uncertainty were also made precise in the form of axioms in [35, 36]. It is assumed that a rational decision cannot be considered separately from rational beliefs. And rational beliefs should be built upon available information (the data) and any personal preference input (the prior information).

In the remainder of this section, we first introduce the formalization of the model choice problem within the Bayesian framework. It leads to the important Bayes factor, discussed in Section 3.2.2. In Section 3.2.3 we discuss the construction of priors and its particular relevance to the Bayesian model comparison problem.

3.2.1 *Model choice problems*

Consider a (possibly infinite) countable set of parametric models, denoted by $\mathcal{M} = \{\mathcal{M}_k\}_{k\in\mathcal{K}}$. Under each model, the data $\boldsymbol{y} = (y_1, \dots, y_n)$ is generated according to a likelihood function $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$ where θ_k is the parameter vector in the space $\Theta_k \subset \mathbb{R}^{d_k}$. Within the Bayesian framework, a prior distribution is chosen for the parameters conditional upon the model, say $\pi(\theta_k|\mathcal{M}_k)$. And each model itself has a prior distribution $\pi(\mathcal{M}_k)$. For the purpose of simplicity, all distributions are assumed to be continuous except $\pi(\mathcal{M}_k)$, which is assumed to be discrete. According to Bayes' theorem, the posterior distribution of the parameters and the model, conditional upon the data, is given by the following density, defined on the space $\bigcup_{k\in\mathcal{K}}\{\mathcal{M}_k\}\times\Theta_k$,

$$\pi(\theta_k, \mathcal{M}_k|\boldsymbol{y}) = \frac{p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)\pi(\mathcal{M}_k)}{p(\boldsymbol{y})}, \tag{3.17}$$

where

$$p(\boldsymbol{y}) = \sum_{k\in\mathcal{K}} p(\boldsymbol{y}|\mathcal{M}_k)\pi(\mathcal{M}_k), \tag{3.18}$$

$$p(\boldsymbol{y}|\mathcal{M}_k) = \int p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)\,\mathrm{d}\theta_k. \tag{3.19}$$

The distribution $\pi(\theta_k, \mathcal{M}_k|\boldsymbol{y})$ is termed the *full posterior*. The within model posterior distribution of the parameters is given by,

$$\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_k) = \frac{p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)}{p(\boldsymbol{y}|\mathcal{M}_k)}. \tag{3.20}$$

The term $p(\boldsymbol{y}|\mathcal{M}_k)$ is called the *marginal likelihood* or the *evidence* of the model. Note that the marginal likelihood is also the normalizing constant of the posterior $p(\theta_k|\boldsymbol{y}, \mathcal{M}_k)$.

From Equation (3.17), it is clear that the posterior model probability $\pi(\mathcal{M}_k|\boldsymbol{y})$ is a marginal of the full posterior, and can be calculated given the prior $\pi(\mathcal{M}_k)$,

$$\pi(\mathcal{M}_k|\boldsymbol{y}) = \frac{\pi(\mathcal{M}_k)\int p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)\,\mathrm{d}\theta_k}{\sum_{l\in\mathcal{K}}\pi(\mathcal{M}_l)\int p(\boldsymbol{y}|\theta_l, \mathcal{M}_l)\pi(\theta_l|\mathcal{M}_l)\,\mathrm{d}\theta_l}. \tag{3.21}$$

The Bayesian model choice problem mostly centers around the inference of this posterior model probability. Many methods for computing this probability are reviewed in Chapter 4. In the remainder of this section, we assume that the calculation of required quantities is possible and accurate.

Our aim is to choose the "best" model from the set $\mathcal{M}$. There will usually be actions taken after the model selection, for example, parameter estimation, or prediction of future events, etc. The consequences of these actions instead of the chosen model itself are of interest. Therefore, from a decision-theoretic perspective, the "best" model should maximize the utility for some quantity of interest. However, in practice it is common to ignore the actions following the model selection and the sole interest is the true model, say $\mathcal{M}_t$. This is because the Bayesian framework is often used to simultaneously provide parameter estimation, model selection, model averaging and other inferences. It is difficult to define a criterion that chooses models best for all these purposes. In the simplified setting, where only the true model is of interest, it is natural to define a zero-one utility function, say $u(\mathcal{M}_k, \mathcal{M}_t)$,

$$u(\mathcal{M}_k, \mathcal{M}_t) = \begin{cases} 0, & \text{if } \mathcal{M}_k = \mathcal{M}_t, \\ 1 & \text{otherwise.} \end{cases} \tag{3.22}$$

It is easy to see that the model $\mathcal{M}_k$ that maximizes the expected utility given data $\boldsymbol{y}$ is the model with the highest posterior probability $\pi(\mathcal{M}_k|\boldsymbol{y})$ [20, chap. 6]. Also see [134, sec. 7.2.1] for an in-depth discussion of the difficulties of the Bayesian formulation in the model choice problem and the reason why such a simplified maximum posterior probability approach.

It should be noted that the use of the zero-one utility is only valid if the true model $\mathcal{M}_t$ belongs to $\mathcal{M}$. Otherwise, the utility is always zero for all models. In what follows, we presume that our aim is to find the model with the highest posterior probability.

Bayesian model selection can be attractive for a few reasons. First, it provides a natural probabilistic interpretation of the results. It is very easy to account model uncertainty within this framework. When there are more than one models well supported by the data and it is uncertain which one should be chosen as the best model, the posterior model probabilities can be used as weights to construct weighted estimator. This leads to Bayesian model averaging. See, e.g., [130, 33, 41], for more discussions and examples.

Second, it is also consistent in the sense that if there is indeed a true model, given enough data, it is guaranteed to be selected. Later we will see some results for the PET compartmental model showing that Bayesian model selection indeed provides better results compared to methods such as AIC.

Third, and perhaps a more important factor, the Bayesian framework can be applied to a wider range of applications compared to methods based on asymptotic behaviors of the data. There are very minimal assumptions about the models under consideration. Model selection methods reviewed earlier often require the good behavior of an estimator, or a sufficient large sample size etc. In contrast, within Bayesian framework, the regularity of the likelihood function is not an issue as long as the integrations in Equation (3.21) are finite. In addition, though a large sample size can be beneficial in the sense that it can reduce the uncertainty of the model selection results, it is not necessary. The uncertainty of model selection is well accounted within the Bayesian framework and improvements can be obtained through model averaging as mentioned earlier. These advantages allow Bayesian model selection to be successfully applied to a wide range of applications.

3.2.2 *Bayes factor*

When the model set $\mathcal{M}$ is finite, we can find the model with the highest posterior probability by comparing models pairwise. To compare the posterior probabilities of two models, say $\mathcal{M}_{k_1}$ and $\mathcal{M}_{k_2}$, one only needs to compute their ratio. Recall Equation (3.21), the ratio can be written as,

$$\frac{\pi(\mathcal{M}_{k_1}|\boldsymbol{y})}{\pi(\mathcal{M}_{k_2}|\boldsymbol{y})} = \frac{\pi(\mathcal{M}_{k_1})}{\pi(\mathcal{M}_{k_2})}\frac{\int p(\boldsymbol{y}|\theta_{k_1},\mathcal{M}_{k_1})\pi(\theta_{k_1}|\mathcal{M}_{k_1})\,\mathrm{d}\theta_k}{\int p(\boldsymbol{y}|\theta_{k_2},\mathcal{M}_{k_2})\pi(\theta_{k_2}|\mathcal{M}_{k_2})\,\mathrm{d}\theta_k} = \frac{\pi(\mathcal{M}_{k_1})}{\pi(\mathcal{M}_{k_2})}B_{k_1k_2}, \tag{3.23}$$

where

$$B_{k_1k_2} = \frac{\int p(\boldsymbol{y}|\theta_{k_1},\mathcal{M}_{k_1})\pi(\theta_{k_1}|\mathcal{M}_{k_1})\,\mathrm{d}\theta_{k_1}}{\int p(\boldsymbol{y}|\theta_{k_2},\mathcal{M}_{k_2})\pi(\theta_{k_2}|\mathcal{M}_{k_2})\,\mathrm{d}\theta_{k_2}} = \frac{p(\boldsymbol{y}|\mathcal{M}_{k_1})}{p(\boldsymbol{y}|\mathcal{M}_{k_2})} \tag{3.24}$$

is called the *Bayes factor*. Equation (3.23) states how the prior odds ratio is transformed into the posterior odds ratio by the Bayes factor [96]. The Bayes factor is the principle tool for Bayesian model comparison and model selection. As it is made clear in the above equation, to compute the Bayes factor, all that needs to be done is

Table 3.2 Jeffreys' intepretation of the Bayes factor.

$\log_{10} B_{k_1k_2}$	$B_{k_1k_2}$	Evidence in favor of model M_{k_1}
0 to 1/2	1 to 3.2	Not worth more than a bare mention
1/2 to 1	3.2 to 10	Substantial
1 to 2	10 to 100	Strong
> 2	> 100	Decisive

the computation of the marginal likelihood for each model $\mathcal{M}_k \in \mathcal{M}$,

$$p(\boldsymbol{y}|\mathcal{M}_k) = \int p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|k) \,\mathrm{d}\,\theta_k.$$

It is obvious that $B_{ij} = B_{ik}B_{kj}$, and thus the Bayes factor approach is equivalent to choosing the model with the highest marginal likelihood provided that the model prior distribution $\pi(\mathcal{M}_k)$ is uniform, as long as the model set $\mathcal{M}$ is finite.

The Bayes factor, $B_{k_1k_2}$, can be interpreted as the evidence provided by the data in favor of model $\mathcal{M}_{k_1}$ against model $\mathcal{M}_{k_2}$. As noted earlier, the marginal likelihood $p(\boldsymbol{y}|\mathcal{M}_k)$ is also called the evidence supporting model $\mathcal{M}_k$. Jeffrey suggested that the Bayes factor can be interpreted on a $\log_{10}$ scale [88]. The interpretations are reproduced in Table 3.2. The interpretation of the Bayes factor can be application dependent. The Jeffreys' interpretation, and a similar scale based on $2\log B_{k_1k_2}$, which is on the same scale as the likelihood ratio test [96], are only general guidelines. Other interpretations can be more suitable for specific applications. For example, [96] mentioned that for forensic evidence to be conclusive in a criminal trial, the posterior odds of guilt against innocence needs to be at least 1,000.

A final remark about the Bayes factor is that, though obviously it can only be used when the set of candidate models is finite, it is not necessarily an issue for many interesting cases. As we will see later in the next chapter, evaluating the marginal likelihood by simulating samples from the model posterior can be relatively easy compared to evaluating the full posterior probabilities. The ease of computation might offset limitations.

The calculation of the Bayes factor can be made exact using analytical results only occasionally. In most applications of interest, approximations have to be used. Two approaches are widely used. One is to use Monte Carlo approximations. Another is based on the asymptotic behavior of the Bayes factor. Two of the later are reviewed here.

Bayesian information criterion

The Bayesian information criterion (BIC) was developed as a large sample approximation to the marginal likelihood $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$ [141]. The BIC is defined as,

$$\text{BIC} = -2\ell_n(\hat{\theta}_k) + k\log(n), \tag{3.25}$$

where $\ell_n(\hat{\theta}_k)$ is the log-likelihood function evaluated at the MLE, k is the number of parameters to be estimated and n is the number of observations. The BIC strategy chooses the model with the smallest value of BIC. A derivation of BIC can be found in [32, sec. 3.2].

Similar to AIC, BIC assumes that the sample size is large enough in order to approximate the marginal likelihood properly. In addition, BIC also assumes "good behavior" of the likelihood function in the sense that the MLE is in the high posterior probability region. These assumptions restrict the use of BIC in some situations. See [18] for examples where the irregularity of the likelihood function caused the BIC method unable to give reasonable results. There are other criticism of the BIC strategy. For example [134, sec. 7.2.3] argued that the BIC strategy eliminated the subjective input into the Bayes modeling since the value of BIC does not depend on the prior distribution. However this is equally argued as an advantage of this strategy in the case that priors, to which the Bayes factors can be very sensitive, are hard to specify.

Though BIC is not an estimator of the Kullback-Leibler divergence, in [147] it was shown that asymptotically BIC is able to choose the simplest model that minimizes the expected KLD between the true model and the candidate model. In

Table 3.3 Frequencies of models selected by BIC (%) for 2,000 PET compartmental model data sets simulated from the three-compartments model.

	Noise level									
Model	0.01	0.02	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12
1	0	0.1	0.8	1.3	3.5	27.1	64.9	87.8	95.7	98.6
2	94.6	96.2	96.1	96.8	95.5	72.7	35.0	12.2	4.3	1.4
3	5.4	3.7	3.1	1.9	1.0	0.2	0.1	0	0	0

contrast to AIC (see discussions in Section 3.1.2), BIC is less subject to overfitting. On the other hand, it may not be as efficient as AIC for some applications. For example, in [105] it was shown that for autoregressive process and some other time series applications, BIC is not efficient in the sense that BIC may not choose the model that minimizes the prediction error. In [32, sec. 4.7], it was also shown that BIC is not efficient for regression variable selection.

Table 3.3 shows the frequencies of models selected by the BIC for 2,000 PET data sets simulated from the three-compartments model (see Section 2.3) while using the NLS estimator. Compared to the use of AIC_c (Table 3.1), the results are quite similar though BIC does tend to select lower order models more frequently. Intuitively, BIC penalizes model complexity more than AIC for $n \geq e^2 \approx 7.4$ (and thus the penalty term $k \log(n) > 2k$).

Laplace approximation

An alternative large sample approximation of the marginal likelihood is given by [159],

$$(2\pi)^{d_k/2}\sqrt{|(-H(\tilde{\theta}))^{-1}|}\,p(\boldsymbol{y}|\tilde{\theta}_k, \mathcal{M}_k)\pi(\tilde{\theta}_k|\mathcal{M}_k) \tag{3.26}$$

where $\tilde{\theta}_k$ is the maximizer of the posterior density $\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_k)$ and $H(\tilde{\theta})$ denotes the Hessian matrix evaluated at $\tilde{\theta}$. The accuracy of the Laplace approximation was examined in [97] and other sources. In general, it is an adequate approximation if the likelihood function is close to Normal [96].

Often the maximizer of the posterior density is not easily obtained. A variant of the Laplace approximation is to use the MLE instead. For a sufficient large sample size, the posterior is likely to peak at the same region as the likelihood function.

Though less commonly used than the BIC approximation, the Laplace approximation does not eliminate the effects of priors. For some applications, it provides a low cost (compared to simulation techniques) alternative for evaluating the Bayes factor over a range of priors.

3.2.3 *Choice of priors*

The prior distribution $\pi(\theta_k|\mathcal{M}_k)$ is chosen by statisticians in the modeling process. The choice of the prior distribution is one of the most critical and criticized part of Bayesian modeling. In principle, the prior distribution should represent the prior beliefs. That is, it represents how much is already known about the data generating mechanism and what is believed about it before the observation of the data, no more or no less. It should not only describe all knowledge already known, but also more importantly preserve all ignorance. If it contains more information than what is actually known, the inference can be biased. In other words, one can always construct a prior distribution such that inference will be biased towards one's preference, which is not necessarily rational. Ideally, the priors need to be considered carefully on a per problem basis. It is often too difficult to elicit a precise distribution from prior information. Therefore it is necessary to make at least partially arbitrary choice of the prior distribution [134, chap. 3][96]. Nonetheless there are a few classes of prior distributions frequently used in practice.

Conjugate priors

A conjugate prior, say $\pi(\theta_k|\mathcal{M}_k)$, for a parametric model with a likelihood function $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$, is one such that the posterior $\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_K)$ belongs to the same family of distributions as the prior. In [20, sec. 5.2] it was argued that a conjugate prior reduces the input of prior information to only the choice of parameter values and thus cannot be fully justified from a subjective perspective. Though their mathematical simplicity makes them attractive, for many applications of interest it is difficult to find such priors.

Non-informative priors

In situations where no or little prior information is available, the so-called "non-informative" priors are often used. Many of them are derived from the data or the likelihood function of the models.

Flat tails The simplest form is a uniform distribution or some distribution with flat tails such as a Cauchy distribution. This choice can provide a robust prior in the sense that outliers and misspecification of priors will not affect the results significantly. For example, see [124] and [42] for analysis of the use of the Student *t* distribution as the prior of location parameters such as the mean of a Normal distribution.

Jeffreys priors Jeffreys priors [87] have the form,

$$\pi(\theta_k|\mathcal{M}_k) \propto \sqrt{|I(\theta_k)|} \tag{3.27}$$

where

$$I(\theta_k) = -\mathbb{E}_{\boldsymbol{y}}\left[\frac{\partial^2 \log p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)}{\partial\theta_k \partial\theta_k^T}\right] \tag{3.28}$$

where the expectation is taken with respect to the likelihood function $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$, is the *Fisher information*. The defining property of Jeffreys priors is their invariance in the sense that, for a one-to-one transformation $\phi_k = h(\theta_k)$, we have the Jacobian transformation,

$$I(\phi_k) = (J(h^{-1}(\phi_k)))^T I(h^{-1}(\phi_k))(J(h^{-1}(\phi_k)))$$

where $J(h^{-1}(\phi_k))$ is the Jacobian matrix . For $\theta_k = h^{-1}(\phi_k)$, it follows,

$$\pi(\phi_k) \propto \sqrt{|I(\theta_k)||J(h^{-1}(\phi_k))|} \propto \pi(\theta_k)\left|\frac{\partial\theta_k}{\partial\phi_k}\right|$$

where the last term is the determinant of the Jacobian transformation. The above expression states that the Jeffreys prior of the transformed parameter ϕ_k is the same as the one obtained by changing variable of the Jeffreys prior of θ_k. In other words, a change of variable does not change the prior under the Jeffreys rule. Another informal interpretation is that, the prior should contain no more information than the observed data do. The Fisher information is widely accepted as an indicator of the amount of information brought by the model about the parameter θ_k given the data [45]. Therefore, intuitively values of θ_k for which $I(\theta_k)$ are large are more likely than those for which $I(\theta_k)$ are small.

However, Jeffreys priors derived for many models may be problematic. It is often the case that the derived Jeffreys priors can be improper. For example, the Jeffreys prior for the mean parameter of a Normal likelihood is uniform on the real line. This can create technical difficulties if the improper priors also lead to improper posteriors since the marginal likelihood, as the normalizing constant of the posteriors, cannot be computed. This makes the Bayesian model choice problem difficult to formalize. See also the discussion in [96]. However, improper priors can also lead to proper posteriors, for example see [57, sec. 2.9], [134, sec. 1.5], [96] and references therein for successful applications using improper priors. In general, for more complex models using proper priors is recommended since it might be easier to verify that the posteriors are also proper. In the situation where the derived Jeffreys priors are improper, it is possible to use some proper distributions to approximate them. For example, an inverse Gamma distribution can be used to arbitrarily closely approximate $\pi(x) \propto 1/x$, which is often seen as the Jeffreys prior for scale parameters, such as the precision parameter of a Normal distribution.

Reference priors Another class of non-informative priors is called *reference priors*, introduced in [19]. Reference priors aim to derive priors such that the distance between the posterior and prior is maximized, usually measured in terms of the Kullback-Leibler divergence [102] (also see Section 3.1.1). In some sense, a reference prior is the least informative prior. See [14, 16, 15] and [20, sec. 5.4] for more information on this class of priors.

Another form of the reference priors is to partition the parameter vector $\theta_k = (\theta_k^{(1)}, \theta_k^{(2)})$ where $\theta_k^{(1)}$ is the parameter of interest and $\theta_k^{(2)}$ is the nuisance parameter. First $\pi(\theta_k^{(2)}|\theta_k^{(1)}, \mathcal{M}_k)$ is defined as the Jeffreys prior associated with $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$ when $\theta_k^{(1)}$ is fixed. Then define the marginal,

$$\tilde{p}(\boldsymbol{y}|\theta_k^{(1)}, \mathcal{M}_k) = \int p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k^{(2)}|\theta_k^{(1)}, \mathcal{M}_k)\, \mathrm{d}\theta_k^{(2)}, \tag{3.29}$$

and compute the Jeffreys prior $\pi(\theta_k^{(1)}|\mathcal{M}_k)$ associated with $\tilde{p}(\boldsymbol{y}|\theta_k^{(1)}, \mathcal{M}_k)$. By using the Jeffreys prior, given fixed parameter of interest, the effects of the nuisance parameter is eliminated.

Using non-informative priors for the PET compartmental model In [167] results of using non-informative priors for the Bayesian analysis of the PET compartmental model were obtained. Here we consider the simulated data (see Section 2.3) and using Normally distributed errors (see Section 2.4). An inverse Gamma distribution approximation to the Jeffreys prior was used for the precision parameter of the Normal distribution. Uniform distributions are used for the $\theta_{1:r}$ and $\phi_{1:r}$ parameters (see Section 2.2 for the parameterization). The interval of the uniform distributions are the same as considered feasible in the optimization procedures such as the NLS estimator. The results of model selection is shown in Table 3.4. Compared to the results of using AIC$_c$ and BIC (Tables 3.1 and 3.3), it is a significant improvement. For data with low level of noise there is a high frequency of selecting the true model.

Table 3.4 Frequencies of models selected by the Bayes factor (vague priors) (%) for 2,000 PET compartmental model data sets simulated from the three-compartments model

	Noise level									
Model	0.01	0.02	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12
1	0	0	0	0	6.3	7.0	24.3	30.7	41.6	54.8
2	12.5	20.1	35.2	49.4	55.3	67.5	62.6	59.1	52.2	43.0
3	87.5	79.9	64.8	50.6	38.4	25.5	13.1	10.2	6.2	2.2

For more noisy data, it is expected the second and third compartments are difficult to identify. Overall, the results are much more satisfactory than those of AIC$_c$ and BIC.

Informative priors

When the parameters bear real world meaning, it may be possible to construct informative priors. For some applications, calibrating an informative prior requires substantial expertise and some model specific information. Nonetheless, there are also some general methods.

When the parameter space is finite, it might be possible to obtain subjective evaluation of the probabilities of the different values of the parameter. When the space is uncountable, the problem is obviously more complicated. One simple approach is to partition the parameter space and determine the probabilities of the parameter falling into each of the partition [134, sec. 3.2.2].

A more systematic approach is the *maximum entropy priors* [86]. Assume that some characteristics of the parameter vector θ_k in the model $\mathcal{M}_k$ are known in the form of prior expectations,

$$\mathbb{E}_\pi[h_i(\theta_k)|\mathcal{M}_k] = \gamma_i, \qquad \text{for } i = 1, \dots, K \tag{3.30}$$

where the expectation is taken with respect to the prior distribution $\pi(\theta_k|\mathcal{M}_k)$ and

h_i is some function. For example, if the mean and variance of the parameter is known, then we might have $h_1(x) = x$ and $h_2(x) = x^2$ in the univariate case. In a finite setting, define the entropy

$$\mathcal{E}(\pi) = -\sum_{\theta_k \in \Theta_k} \pi(\theta_k|\mathcal{M}_k) \log \pi(\theta_k|\mathcal{M}_k) \tag{3.31}$$

where Θ_k is the parameter space.The maximum entropy prior is the π that maximizes $\mathcal{E}$ under the constraints of Equations (3.30).

This can be extended to the continuous case by introducing a reference distribution, say $\pi_0(\theta_k)$. Define the entropy as the negative KLD between $\pi_0(\theta_k)$ and $\pi(\theta_k|\mathcal{M}_k)$,

$$\mathcal{E}(\pi) = -\int \pi_0(\theta_k) \log\left(\frac{\pi_0(\theta_k)}{\pi(\theta_k|\mathcal{M}_k)}\right) \mathrm{d}\,\theta_k \tag{3.32}$$

The reference distribution π_0 can be chosen as one of the non-informative priors discussed earlier. Such a choice is justified in [134, chap. 9].

The maximum entropy procedure in essence selects a prior that preserves the prior information – the expectations of $\{h_i(\theta_k)\}_{i=1}^{K}$ while making it as close to the non-informative prior as possible. Therefore, the prior knowledge is presented while the ignorance is also preserved.

Using informative priors for the PET compartmental model Here we give an example of constructing a biologically informed prior for the PET compartmental model. It is based on both the prior knowledge of the underlying biochemical process and mathematical properties of the models.

In [6] some useful results about compartmental models in general were provided. Recall the parameterizations in Section 2.2. Let γ_{0j} denote the rate constant of the outflow from the j^{th} compartment into the environment. Without loss of generality, assume that the parameters $\theta_{1:r}$ are ordered, $\theta_1 \leq \ldots \leq \theta_r$. Then,

1. $0 \leq \theta_i \leq 2\max_j |A_{jj}|$ for all i.
2. $\min_j \gamma_{0j} \leq \theta_1 \leq \max_j \gamma_{0j}$.
3. when there is only one outflow into the environment, say the rate constant of this outflow is k_2, as in the plasma input model, then $0 \leq \theta_1 \leq k_2$.

In addition, $\sum_{i=1}^{r} \phi_i = K_1$, where K_1 is the rate constant of input from the plasma into the tissues [67]. Therefore $\phi_i < K_1$ for $i = 1, \ldots, r$. Given this information, more informative prior distributions can be constructed. For simplicity, we restrict discussion to imposing upper and lower bounds on the possible values of the parameters. As we subsequently find that inference is not overly sensitive to the prior specification we do not pursue more complicated approaches.

To demonstrate the idea, an informative prior distribution for parameters $\theta_{1:3}$ and $\phi_{1:3}$ in the three-compartments model is constructed. First note that the transition matrix A (see Section 2.3) is,

$$A = \begin{bmatrix} -k_2 - k_3 - k_5 & k_4 & k_6 \\ k_3 & -k_4 & 0 \\ k_5 & 0 & -k_6 \end{bmatrix}.$$

It is believed that all the rate constants take values in the range $[5 \times 10^{-4}, 10^{-2}]$ (for example see [167, 125]). Without loss of generality, we impose the identifiability constraint $\theta_1 \leq \theta_2 \leq \theta_3$, then,

$$0 < \theta_1 \leq k_2 \leq 10^{-2} \tag{3.33}$$

$$\theta_1 \leq \theta_2 \leq \theta_3 \leq \max\{2(k_2 + k_3 + k_5), 2k_4, 2k_6\} \leq 6 \times 10^{-2} \tag{3.34}$$

Under the imposed ordering, as θ_1 is the smallest exponent, the term $\phi_1 e^{-\theta_1 t}$ decays more slowly than any other term in the expansion. Consequently, ϕ_1/θ_1 is likely to make a relatively large contribution to $V_D = \sum_{i=1}^{r} \phi_i/\theta_i$. It is not well known how large the ratio $(\phi_1/\theta_1)/V_D$ will be. However, it is easy to conduct a numerical study here, given the small number of parameters. It is found that among all possible combination of rate constants, $\phi_1/\theta_1 \geq 0.5V_D$. If the combinations of the rate constants are restricted to those without excessively large differences among them, i.e., cases in which, say, $k_5 \gg k_6$ are not considered, then $\phi_1/\theta_1 \geq 0.7V_D$. The reason for not considering these cases is that such irreversible (trapped) models yield infinite V_D estimates and it is generally known in advance that the tracer employed will exhibit reversible dynamics. In addition, from results of previous

Table 3.5 Frequencies of models selected by the Bayes factor (informative priors) (%) for 2,000 PET compartmental model data sets simulated from the three-compartments model.

	Noise level									
Model	0.01	0.02	0.04	0.08	0.16	0.32	0.64	1.28	2.56	5.12
1	0	0	0	0	0	1.0	6.2	15.2	27.8	37.1
2	10.6	17.5	33.3	45.8	58.8	70.2	73.0	67.3	57.3	53.0
3	89.4	82.5	66.7	54.2	41.2	28.8	20.8	17.5	14.9	9.9

studies, it is reasonable to believe that V_D in the range from 10 to 30 for the majority of voxels. We might just set $V_D \approx 20$.

With this information we can then construct informative priors for the parameters of interest. The means the distributions of $\phi_{1:r}$ are chosen such that they are less than K_1 but sum up to it, without significant differences among than. Conditional on $\phi_{1:r}$, the distributions of $\theta_{1:r}$ can be specified. For example, conditional on ϕ_1, θ_1 has mean $\phi_1/(0.8V_D)$ (since $0.7V_D \leq \phi_1/\theta_1 \leq V_D$). Similarly the distributions of θ_i conditional on $\phi_{1:i}$ and $\theta_{1:i-1}$ for $i = 2$ and 3 can be constructed. More technical details can be found in [167]. In particular, truncated Normal distributions are used as the priors for the parameters given all the upper and lower bounds. Table 3.5 shows the model selection results when using these informative priors. It can be seen that, especially for data with higher noise level, the results are considerably improved compared to using vague priors (Table 3.4).

Sensitivity analysis

Intuitively the influence of priors can be eliminated given enough data. In the particular problem of Bayesian model comparison, it should be noted that the Bayes factor can be more sensitive to the choice of prior than the posterior means of parameters, in the sense that more data are needed to eliminate the influence of priors [95, 96].

Note that by "more data", we not only refer the situation where a larger sample size is obtained, but also when the data are measured more accurately. For example, compare the results of using informative priors (Table 3.5) and those of using vague priors (Table 3.4) for the PET compartmental model. For less noisy data (i.e., the data is more informative), the difference is minimal. For noisy data (i.e., the data is less informative), there are considerable differences in model selection results. In contrast, the estimates of V_D see much less difference even for very noisy data in terms of mean squared errors (MSE; results are not shown in this thesis, see [167]). Though in our example, such difference is not problematic, the sensitivity of the Bayes factor to the choice of priors should not be overlooked in realistic applications.

It is therefore of interest to evaluate the Bayes factor over a range of possible priors to assess the sensitivity issues. This is often computationally expensive since many high dimensional integrations are required. When there is enough information to construct parametric priors, it is possible to alter the values of parameters and recompute the Bayes factor [114]. In general situations, a less computational expensive method is to use the Laplace approximation to compare the Bayes factor using different priors [97]. It is also proposed in the literature to use the maximum of the Bayes factor (and thus the maximal evidence against a model) to evaluate the sensitivity problem [17].

3.3 DISCUSSIONS

We have reviewed a few model selection methods in this chapter. The information-theoretic approach is well understood and has been practiced for a long history. The Bayesian approach has gained substantial interest in the last few decades. The computation of the integrations required by Bayesian model comparison will be reviewed in the next chapter.

The Bayesian approach can be appealing in at least two situations. First, when there is substantial prior knowledge about the underlying data generating mechanisms, the Bayesian framework provides a way to incorporate these information to lead to rational decisions. Even when no or little prior information is available, the Bayesian approach can be useful when the underlying assumptions about the models or data that lead to the asymptotic results regarding the various information criteria are not met, as Bayesian model comparison imposes very few assumptions on the form of models. For example, through the running example of this thesis, the PET compartmental model, we have found that Bayesian model selection even with vague priors can provide significant improvement over methods such as AIC$_c$ or BIC which is only an approximation to the Bayes factor. The results can be further improved via informative priors.

Despite all the advantages, using Bayesian model comparison also has considerably higher computational cost since many high dimensional integrations are required. As we will see later, many widely used techniques may fail to evaluate these integrations accurately. The work of this thesis aims to provide a new framework within which the computational difficulty is further lowered than current practice.

This review is far from comprehensive. We have restricted ourselves to methods that have minimal assumptions about the form of the models. For particular models, many other methods have been developed. For example, in regression analysis, the Mallow's C_p is a popular model selection criterion. For more information on information-theoretic approaches we refer to [23, 32].

One generic approach to model selection that is not reviewed in this chapter but worth mentioning is the minimum description length (MDL) method. It is somehow different from the methods reviewed so far. It selects the model with the least complexity which is measured as the length of using a programming language to describe the data given the model and the model itself. The description length of data given the model can be measured with $-\log p(\boldsymbol{y}|\mathcal{M}_k)$ while the description length of the model can be more problematic since there is no universally agreed good form of this length. We refer to [66] for more information on this approach and its modern refinements.

The review of Bayesian model comparison in this chapter gives context for later chapters. In Chapter 4, some algorithms for evaluating the Bayes factor and posterior model probabilities are reviewed and in Chapter 5, novel algorithms are developed. Limited by the scope, there are many important topics within the Bayesian framework that were not discussed. We refer to [20, 134] for a more systematic treatment. In particular, [20] also has comprehensive bibliographies on many of the topics related to Bayesian statistics.

4 MONTE CARLO METHODS

As shown in the last chapter, Bayesian model comparison usually involves computation of some integrations with respect to complex posterior distributions. Only in very special situations, this can be resolved analytically. In most cases, these integrations are approximated using simulation techniques. In this chapter, we review some of the widely used Monte Carlo methods with an emphasis on their applications to Bayesian computation.

In Section 4.1 we introduce the basic idea of Monte Carlo integration. Section 4.2 discusses the importance sampling technique. Section 4.3 reviews a class of important Monte Carlo algorithms, Markov chain Monte Carlo. This chapter is concluded by Section 4.4, a discussion on the reviewed algorithms and some other development in this area that is not reviewed in detail.

4.1 CLASSICAL MONTE CARLO

Classical Monte Carlo integration approximates the expectation of a function φ with respect to a (continuous) distribution π,

$$\mathbb{E}_\pi[\varphi(X)] = \int \varphi(x)\pi(x)\,\mathrm{d}\,x$$

provided that the above expectation exists, by drawing identically independently distributed (i.i.d.) samples from π, say $\{X^{(i)}\}_{i=1}^N$, and approximating the expectation by the empirical average,

$$\hat{\varphi}_{\text{MC}}^N = \frac{1}{N}\sum_{i=1}^N \varphi(X^{(i)}). \tag{4.1}$$

This method is also called *vanilla* Monte Carlo or *naïve* Monte Carlo. The estimator $\hat{\varphi}_{\text{MC}}^N$ converges almost surely to $\mathbb{E}_\pi[\varphi(X)]$ when $N \to \infty$ by the Strong Law of Large Numbers (SLLN).

Clearly this method can only be applied when drawing samples directly from the target distribution π is possible. There are a few ways to draw random variates from a reasonably well behaved distribution. See [135, chap. 2] on this topic. In many cases, simulation from a distribution π efficiently requires the evaluations of its density function point-wise, or finding some easy to simulate distribution that closely imitates π. In the context of Bayesian computation, the target distributions are usually complex posteriors only known up to some normalizing constants. And thus point-wise evaluation is not possible. In addition, the high dimensional aspect of many models makes it near impossible to find a distribution that closely imitates the target. In addition, even when it is possible, the accuracy of the estimator $\hat{\varphi}^N_{\text{MC}}$ depends heavily on the function φ.

For example, consider the approximation of the marginal likelihood (see Section 3.2.2),

$$p(\boldsymbol{y}|\mathcal{M}_k) = \int f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)\,\mathrm{d}\,\theta_k,$$

or equivalently,

$$p(\boldsymbol{y}|\mathcal{M}_k) = \mathbb{E}_\pi[f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)], \tag{4.2}$$

where the expectation is taken with respect to the prior distribution $\pi(\theta_k|\mathcal{M}_k)$. It is possible to use samples from $\pi(\theta_k|\mathcal{M}_k)$, the prior distribution, to approximate the integration. With samples generated from $\pi(\theta_k|\mathcal{M}_k)$, say $\{\theta_k^{(i)}\}_{i=1}^N$, we can approximate $p(\boldsymbol{y}|\mathcal{M}_k)$ by,

$$\widehat{p(\boldsymbol{y}|\mathcal{M}_k)}^N_{\text{MC}} = \frac{1}{N}\sum_{i=1}^N f(\boldsymbol{y}|\theta_k^{(i)}, \mathcal{M}_k). \tag{4.3}$$

This estimator was studied in [114] and also mentioned in [96]. However, this approach often results in large variances. The likelihood function (and thus the posterior distribution) is often much more concentrated than the prior distribution. And there may be a large proportion of samples with small likelihood values and a few with high values. For instance, consider the one-compartment PET model (see Section 2.2) and informative priors (see Section 3.2.3). Using 100,000 samples from the prior distribution, the empirical mean and standard deviation of the estimates from 100 simulations is −40.9 and 2.1, respectively for the simulated data. However when the dimension of the model is increased by using a two-compartments model, the estimates have empirical mean and standard deviation −39.6 and 12.6, respectively. The variance is too large for practical use of evaluating the Bayes factor. Similar problems were also shown in [114].

4.2 IMPORTANCE SAMPLING

The *importance sampling* method is based on the observation of the following identity,

$$\mathbb{E}_\pi[\varphi(X)] = \int \varphi(x)\pi(x)\,\mathrm{d}\,x = \int \varphi(x)\frac{\pi(x)}{\eta(x)}\eta(x)\,\mathrm{d}\,x = \mathbb{E}_\eta\Big[\varphi(X)\frac{\pi(X)}{\eta(X)}\Big], \quad (4.4)$$

where η is a distribution with respect to which π is absolutely continuous and the two expectations are taken with respect to π and η, respectively. The above equation is termed *importance fundamental identity* in [135]. The distribution η is often called the *proposal* or *instrumental* distribution. Thus given i.i.d samples $\{X^{(i)}\}_{i=1}^N$ from distribution η, the expectation $\mathbb{E}_\pi[\varphi(X)]$ can be approximated by the following importance sampling estimator,

$$\hat{\varphi}_{\mathrm{IS}}^N = \frac{1}{N}\sum_{i=1}^N \varphi(X^{(i)})\frac{\pi(X^{(i)})}{\eta(X^{(i)})}. \quad (4.5)$$

The above estimator converges almost surely to $\mathbb{E}_\pi[\varphi(X)]$ when $N \to \infty$. However, its variance is not necessarily finite. In general, the variance is finite if and only if, [135, sec. 3.3.2],

$$\int (\varphi(x))^2\frac{(\pi(x))^2}{\eta(x)} < \infty. \quad (4.6)$$

To access the above inequality, evaluating a more complex integration than the original problem is required. In [59] two types of sufficient conditions were mentioned. One is that π/η is upper bounded and $\mathrm{var}_\pi[\varphi(X)]$ is finite. Another is that the support is compact, π is upper bounded and η is lower bounded by $\varepsilon > 0$.

Both π and η are often only known up to some normalizing constants, which can be approximated with the same samples. This leads to the estimator,

$$\hat{\varphi}^N_{\text{WIS}} = \frac{\sum_{i=1}^N w^{(i)}\varphi(X^{(i)})}{\sum_{i=1}^N w^{(i)}} \tag{4.7}$$

where $w^{(i)} \propto \pi(X^{(i)})/\eta(X^{(i)})$, and are termed the *importance weights*. This estimator also converges almost surely to $\mathbb{E}_\pi[\varphi(X)]$ when $N \to \infty$. This estimator has a bias since it is the ratio of two unbiased estimator. However, even when the normalizing constants of π and η are both known, this estimator can be preferable to $\hat{\varphi}_{\text{IS}}$ due to its possible smaller mean squared error. In fact, [26] showed an example of using the Cauchy distribution as the proposal distribution for the evaluation of expectations under the Student t distribution, where for some functions, such as $\varphi(x) = |x|$, $\hat{\varphi}^N_{\text{WIS}}$ can outperform $\hat{\varphi}^N_{\text{IS}}$ considerably.

In general, the performance of the importance sampling depends not only on the choice of the proposal distribution η, but also the function of interest φ. As suggested in [135, sec. 3.3.2], to minimize the variance of the estimator, the distribution η should be chosen such that $|\varphi(x)|\pi(x)/\eta(x)$ is almost constant with a finite variance. That is, it is preferable for the distribution η to be proportional to $|\varphi|\pi$ and to have heavier tails. Though heavier tails do not necessarily lead to the sufficient conditions such as those mentioned in [59], thinner tails are more likely to result in infinite variances as extreme large importance weights are more likely to occur in this situation. Often the same samples are used to evaluate the expectations of different functions. And the proposal distribution is chosen such that η is close to π with heavier tails.

In the context of Bayesian model comparison, we are interested in the evaluation of the marginal likelihood $p(\boldsymbol{y}|\mathcal{M}_k)$. We may use a proposal distribution, say η and samples drawn from it to approximate the expectation (4.2). This leads to the estimator,

$$\widehat{p(\boldsymbol{y}|\mathcal{M}_k)}^N_{\text{IS}} = \frac{\sum_{i=1}^N w^{(i)} f(\boldsymbol{y}|\theta_k^{(i)}, \mathcal{M}_k)}{\sum_{i=1}^N w^{(i)}} \tag{4.8}$$

where $w^{(i)} \propto \pi(\theta_k^{(i)}|\mathcal{M}_k)/\eta(\theta_k^{(i)})$ and $\{\theta_k^{(i)}\}_{i=1}^N$ are distributed with η. Good performance can be obtained when η is close to $f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)$. In other words, we need some knowledge of the posterior distribution $\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_k)$, which is often not available for complex models. Even when some characteristics of the posterior distribution are available, other techniques might be preferred. For example, with such information, algorithms reviewed in the next section may allow us to efficiently simulate dependent samples with the posterior distribution as the limiting distribution.

One possible solution to such problems is to use some form of adaptive schemes. For example, [123] proposed to use a family of parametric distributions as proposal distributions and the parameters are iteratively tuned. In their paper, a multivariate t distribution is used as an example. Compared to a multivariate Normal distribution, it has heavier tails and by changing the degree of freedom and other parameters, the distribution can be changed into different shapes to match the characteristics of the target distribution. This can be flexible for some applications. However, for many distributions of interest, especially that are high dimensional and multimodal, it is difficult to find an explicit parametric distribution that can sample those local modes efficiently. As we will see in the next chapter, sequential Monte Carlo algorithms can iteratively construct efficient proposals and are suitable for a wide range of applications, including Bayesian model comparison.

4.3 MARKOV CHAIN MONTE CARLO

Both the vanilla Monte Carlo and importance sampling methods require simulations directly from a distribution, which is often not feasible in realistic applications. Estimation techniques based on dependent samples were developed. The most important type, Markov chain Monte Carlo (MCMC), uses dependent samples generated by a Markov chain with the target π as a limiting distribution for the approximation of the desired integration. A limiting distribution, informally is one such that if X^t, the state of the Markov chain at step t is distributed with π, then X^{t+1}, the next state is also distributed with π.

The basic idea is that, given samples from a Markov chain with a limiting distribution π, say $(X^{(1)}, \dots, X^{(i)}, \dots)$, then under suitable conditions (outlined for each algorithm later), for a π-integrable function φ,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} \varphi(X^{(i)}) = \mathbb{E}_{\pi}[\varphi(X)]. \tag{4.9}$$

Therefore samples generated by this Markov chain can be used for estimation of various quantities in a similar fashion as with vanilla Monte Carlo or importance sampling. This leads to the estimator,

$$\hat{\varphi}^{N}_{\text{MCMC}} = \frac{1}{N} \sum_{i=1}^{N} \varphi(X^{(i)}) \tag{4.10}$$

where $\{X^{(i)}\}_{i=1}^{N}$ are N samples from the Markov chain.

The construction of such Markov chains leads to the development of various widely used MCMC algorithms. In this section, some of the more important ones are reviewed.

4.3.1 *Discrete time Markov chain*

This section briefly discusses some notions of discrete time Markov chains. Most concepts are introduced in a descriptive way and we restrict ourselves to the continuous case. For formal definitions in more general settings, see Appendix A.1. Also see [135, chap. 6] for a treatment of the topic in more detail in the context of MCMC algorithms.

A Markov chain can be defined in terms of *transition kernels*. For continuous random variables X and X' defined in space E, which are ordered in time in some sense, a transition kernel is the distribution of X' conditional on X. That is, $\Pr(X' \in A|x) = \int K(x, x') \,\mathrm{d}x'$ where $A \subset E$. We also call K just *kernel*.

A discrete time Markov chain, denoted by (X^t) is a sequence of random variables $X^0, X^1, \dots, X^t, \dots$ such that conditional on $(x^{t-1}, \dots, x^0)$, X^t has the same distribution as it has conditional on x^{t-1}. Clearly a transition kernel is such a conditional distribution. In the context of MCMC, we are mostly concerned with *time homogeneous* Markov chains. A Markov chain (X^t) is said to be time homogeneous if for every $t_0 \leq t_1 \leq \ldots \leq t_k$, the distribution of $(X^{t_1}, \ldots, X^{t_k})$ conditional on x^{t_0} is the same as $(X^{t_1 - t_0}, \ldots, X^{t_k - t_0})$ conditional on x^0. In other words, given the initial state x^0 or its distribution, the Markov chain is determined solely by its transition kernel.

We will be mostly concerned with the sensitivity of the Markov chain with respect to the initial value X^0 or its distribution and the existence and the speed at which the Markov chain converges to its limiting distribution. A few properties of a given Markov chain are discussed below. In short, *irreducibility* states that all states communicate. The Markov chain can reach any state $y \in E$ starting from any other state $x \in E$. A stronger version says that the chain can travel any distance in one step. Another property is *aperiodicity*, which says that for a chain leaving a group of states it does not need to take k or a multiple of k steps to return to it with $k > 1$. Informally, a sufficient but not necessary condition for an irreducible chain to be aperiodic is that the chain can stay in a neighborhood of a state (or at the state in the discrete case) for an arbitrary number of instances without being forced to leave it. Or it does not need to go through a cycle to reach back into the neighborhood of the current state. These two properties guarantee a Markov chain to explore a space freely. A third property we will discuss is *recurrence*, which states that the Markov chain will visit any state for infinite times. In other words, the Markov chain can explore a space throughout starting from almost anywhere. A stronger version, *Harris recurrence*, allows the chain to start from *everywhere*. A property fundamental to MCMC algorithms is the existence of *invariant* distribution, which states that the Markov chain can be stable under suitable conditions and converge to a desired distribution. A sufficient condition for the existence of invariant distribution, *detailed balance*, is perhaps the most useful tool in practice to check the validity of a given algorithm. Last, we will discuss the *ergodicity* of Markov chains, which measures the speed at which a Markov chain converges to its invariant (and thus its limiting) distribution. An ergodic Markov chain is one whose marginal distribution of X^t converges to the limiting distribution π when $t \to \infty$, in the sense that the total-variation norm converges to zero. Stronger forms of convergence also exists. A Markov chain is said to be *geometrically ergodic*, if the convergence speed, measured as the total-variation norm between the marginal distribution of X^t and the limiting distribution π, is bounded by a geometric sequence, for every initial value in the space the Markov chain is defined. Further, if this bound is uniform across the speed, that is there is a geometric sequence independent of the initial value by which the total variance is bounded, then the Markov chain is said to be *uniformly ergodic*.

4.3.2 *Metropolis-Hastings algorithm*

The Metropolis-Hastings algorithm, first introduced in [116] and then generalized in [71], produces a Markov chain with limiting distribution π with a conditional distribution $q(\cdot|x)$ called the *proposal* or *instrumental* distribution through the following transition. At time t, given sample X^t, first Y^t is drawn from $q(y^t|x^t)$. Then, set

$$X^{t+1} = \begin{cases} Y^t, & \text{with probability } \alpha(x^t, y^t), \\ X^t & \text{with probability } 1 - \alpha(x^t, y^t). \end{cases}$$

where

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}\frac{q(x|y)}{q(y|x)}, 1\right\}. \tag{4.11}$$

The probability $\alpha(x, y)$ is called the *Metropolis-Hastings acceptance probability*. This leads to Algorithm 4.1.

The conditions under which the Markov chain produced by this algorithm has π as its limiting distribution are quite minimal [135, sec. 7.3.2]. Intuitively, the generated Markov chain is aperiodic if the algorithm allows events such as $\{X^{t+1} = X^t\}$, that is, the acceptance probability is not equal to one almost surely. A sufficient condition for irreducibility is that the conditional distribution $q(\cdot|x)$ is positive. In other words, it allows that every subset of the state space with can be reached in a single step. It can be proved that with these two conditions, the

```
Draw $X^0 \sim \mu$ where $\mu$ is the initial condition.
Set $t \leftarrow 0$.
**repeat**
    Draw $Y^t \sim q(y^t|x^t)$.
    Compute $\alpha = \min\left\{\frac{\pi(y^t)}{\pi(x^t)}\frac{q(x^t|y^t)}{q(y^t|x^t)}, 1\right\}$
    Draw $U \sim \mathcal{U}[0,1]$.
    **if** $U \leq \alpha$ **then**
        Set $X^{t+1} \leftarrow Y^t$.
    **else**
        Set $X^{t+1} \leftarrow X^t$.
    **end if**
    Set $t \leftarrow t+1$.
**until** Sufficiently many samples have been produced.
```

Algorithm 4.1 The Metropolis-Hastings algorithm

convergence in Equation (4.9) holds [135, Theorem 7.4 and Corollary 7.5].

The Metropolis-Hastings algorithm is important not only because it has found many applications, but also because it is the foundation of many other algorithms. For example the reversible jump MCMC and population MCMC algorithms, reviewed later in Section 4.3.4 and 4.3.5, respectively, can both be viewed as extensions to this algorithm.

The design of the proposal distributions can greatly influence the performance of the estimators. It has been a difficult problem and has attracted substantial attention in the past. In the following, we discuss three commonly used designs.

Independent proposals

A proposal independent of the current state X^t leads to the independent Metropolis-Hastings algorithm. Let η denote this proposal. The acceptance probability becomes,

$$\alpha(x, y) = \min\left\{\frac{\pi(y)\eta(x)}{\pi(x)\eta(y)}, 1\right\}. \tag{4.12}$$

The resulting Markov chain is uniformly ergodic if the target π is bounded by the proposal η up to a multiplier. In other words, there exists a constant M such that $\pi(x) \le M\eta(x)$ for all x in the support of π.

Though uniform ergodicity is a much desired property for a given algorithm, without proper optimizing, the performance of an independent proposal is often far from ideal. The proposal η should be chosen such that it maximizes the *average acceptance rate* $\bar{\alpha} = \mathbb{E}[\alpha(x, y)]$. Given a stationary chain and thus the state X is distributed with π, and a proposed value Y which is distributed with η, it is defined as,

$$\bar{\alpha} = \mathbb{E}\left[\min\left\{\frac{\pi(Y)\eta(X)}{\pi(X)\eta(Y)}, 1\right\}\right] = 2\Pr\left(\frac{\pi(Y)}{\eta(Y)} \ge \frac{\pi(X)}{\eta(X)}\right), \tag{4.13}$$

provided that π/η is absolutely continuous and the expectation is taken with respect to $f(x, y) = \pi(x)\eta(y)$. The second equality is made clear by the following. Let

$$z(x, y) = \frac{\pi(y)\eta(x)}{\pi(x)\eta(y)}.$$

It is clear that $z(x, y) < 1$ is equivalent to $z(y, x) > 1$. For continuous distributions, expand the expectation,

$$\begin{aligned}\bar{\alpha} &= \int \min\{z(x, y), 1\} f(x, y)\, \mathrm{d}x\, \mathrm{d}y \\ &= \int_{z(x,y)<1} z(x, y) f(x, y)\, \mathrm{d}x\, \mathrm{d}y + \int_{z(x,y)\ge 1} f(x, y)\, \mathrm{d}x\, \mathrm{d}y.\end{aligned}$$

Note that,

$$z(x, y) f(x, y) = \frac{\pi(y)\eta(x)}{\pi(x)\eta(y)}\pi(x)\eta(y) = \pi(y)\eta(x) = f(y, x).$$

It follows,

$$\begin{aligned}\bar{\alpha} &= \int_{z(y,x)>1} f(y,x)\,\mathrm{d}x\,\mathrm{d}y + \int_{z(x,y)\geq 1} f(x,y)\,\mathrm{d}x\,\mathrm{d}y \\ &= 2\Pr(z(x,y) \geq 1).\end{aligned}$$

The average acceptance rate $\bar{\alpha}$ measures how often a new proposed value is accepted in the long run of the algorithm. This optimization is generic in the sense that the function of interest φ is not involved. In practice, η should be chosen such that it is close to π as much as possible. The requirement for π/η to be bounded also suggests that η at least should not have too thin tails compared to π. Ideally it should have slightly heavier tails than π but not much less concentrated. In this aspect, the choice of η is similar to the choice of the proposal distribution for the importance sampling. Hence it inherits the same difficulties as outlined in Section 4.2.

Random walks

The random walk Metropolis-Hastings algorithm, originally introduced in [116], uses proposals that are symmetric, often in the form $q(y|x) = q(|y-x|)$. This leads to the acceptance probability,

$$\alpha(x,y) = \min\left\{\frac{\pi(y)}{\pi(x)}, 1\right\}. \tag{4.14}$$

This algorithm does not satisfy conditions for the uniform ergodicity in general. However it is geometrically ergodic under certain conditions. In [115], a condition based on log-concavity of π in the tails was given. The Markov chain is geometrically ergodic if,

$$\log\pi(x_1) - \log\pi(x_2) \geq \alpha|x_1 - x_2| \tag{4.15}$$

for some $\alpha > 0$ and some x_0 such that $x_0 < x_1 < x_2$ or $x_2 < x_1 < -x_0$.

The random walk is one of the most widely used type of MCMC algorithms. It provides a generic working solution to many otherwise difficult problems. However, without optimization, its performance is often far from satisfactory. For example, multimodal distributions often have modes that are separated by extremely small probability areas. These areas limit the move of the random walk. If the chain proposes bigger steps, then it is possible that most proposed values fall in small probability areas and the probability of jumping from one mode to another is arbitrarily small. This leads to extremely small acceptance rates. On the other hand, if the chain proposes smaller steps, it will take many iterations for the chain to explore the whole space. In either case, if the scaling (such as the variance of a Normal distribution or some other measures of the dispersion of the proposal distribution) of the random walk is chosen poorly, it can take arbitrarily long time for the chain to move outside the neighborhood of one local mode of the target distribution. In this situation, the sampler is said to be in a *trapping state*.

For instance, consider the PET compartmental model (see Section 2.2), a Normally distributed error structure (see Section 2.4), and non-informative priors (see Section 3.2.3) for the simulated data. We construct a random walk algorithm with three blocks,

1. Update $\phi_{1:r}$ with a multivariate Normal random walk proposal.
2. Update $\theta_{1:r}$ with a multivariate Normal random walk proposal
3. Update λ with a Normal random walk proposal on the log scale, i.e., on $\log\lambda$.

Both Figure 4.1 and 4.2 show the trace of (ϕ_1, θ_1) from three samplers for the one-compartment model, initialized with different values. Each sampler is iterated 10,000 times. In the former, the proposal scales (the variance of the Normal distributions, which are univariate in the case of the one-compartment model) are well tuned while the later uses scales five times of the former. In Figure 4.1, each sampler is able to find the high probability region quickly and explore it efficiently. In contrast, in Figure 4.2, none of the samplers is able to find the high probability region and they are trapped around the initial values.

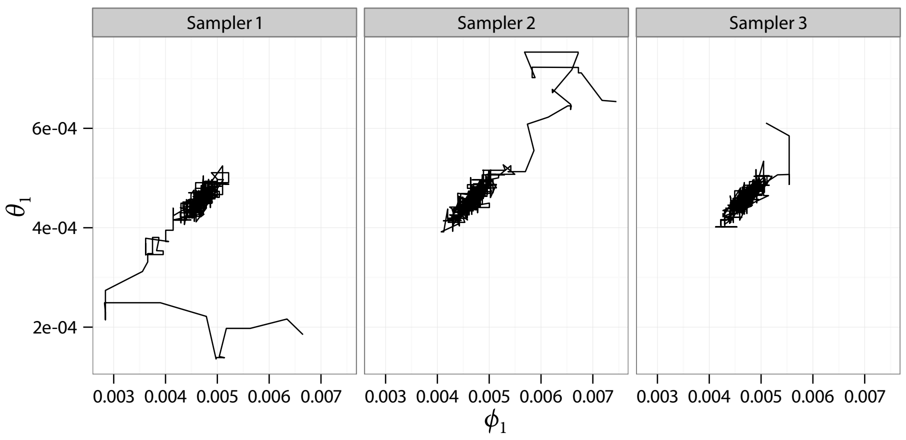


Figure 4.1 Trace of (ϕ_1, θ_1) from three random walk Metropolis-Hastings samplers for the one-compartments PET model with non-informative priors, using well tuned proposal scales. The first few values of each trace are not shown in the plots since they are far away from the high probability region with an order of magnitude difference in values.

Optimal proposal scales As seen in the above example, finding the optimal scales for a random walk algorithm can be important for realistic applications. One way to measure the optimality of the random walk is based on the asymptotic behavior of an *efficiency criterion* equal to the ratio of the variance of an estimator based on an i.i.d sample and the variance of the estimator $\hat{\varphi}^N_{\text{MCMC}}$ in Equation (4.10). In [136] it was recommended that the optimal proposal distribution should produce chains with acceptance rate close to 0.5 for models with dimension 1 or 2 and 0.25 for models with higher dimensions. One of the more widely used type of proposals is the Normal distribution or its multivariate variant. In [54] a form of optimal covariance is given as $(2.38^2/d)\Sigma_\pi$, where d is the dimension of the target distribution π and Σ_π is the true covariance matrix of the parameters under π. In [137] it was further established that when the dimension goes to infinity, and the covariance matrix is assumed to be diagonal, say $I_d\sigma_d^2$, the optimal scaling of σ_d has a corresponding acceptance rate 0.234. This optimal rate has been commonly used as a rule of thumb

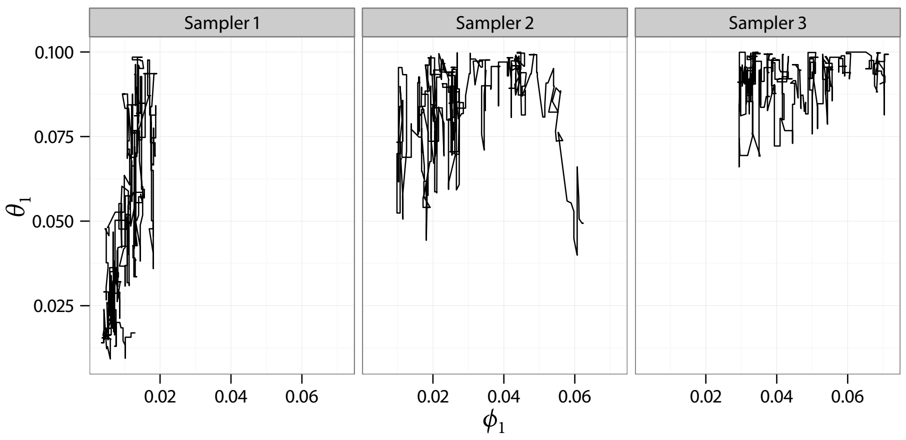


Figure 4.2 Trace of (ϕ_1, θ_1) from three random walk Metropolis-Hastings samplers for the one-compartments PET model with non-informative priors, using proposal scales five times of those tuned.

in practice. Some more recent results for other proposal distributions including non-Gaussian cases can be found in, e.g., [145, 119]

Adaptive proposals

It often requires substantial efforts to tune an algorithm's proposals towards optimality. Alternatively, many adaptive strategies have been developed. See [11] for a recent review. The basic theme is that a family of proposal distributions, indexed by some parameter, say $q(\cdot|x) = q(\cdot|x, \theta)$ with $\theta \in \Theta$, is considered. And the value of the parameter is updated along with the state. This leads to Algorithm 4.2.

There are many methods of updating the parameters. See [11] for some common algorithms. One of the more widely used is based on the Normal random walk or its multivariate variant. In [68, 69], it is proposed to use the past samples to approximate the optimal covariance matrix $(2.38^2/d)\Sigma_\pi$ [54]. The algorithm first initializes μ^0, a d-vector and Σ^0, a covariance matrix. At time t, they are updated

```
Draw $X^0 \sim \mu$ where $\mu$ is the initial condition.
Set $\theta^0$ to an arbitrary valid value.
Set $t \leftarrow 0$.
**repeat**
    Compute $\theta^t = \gamma^t(\theta^0, X^0, \dots, X^{t-1})$ where $\gamma^t$ is a transformation that update
    the parameters based on past samples.
    Draw $X^{t+1}$ using the proposal $q(\cdot|x^t, \theta^t)$ with the Metropolis-Hastings rule.
    Set $t \leftarrow t + 1$.
**until** The accept rate of the last $K$ iterations are close to 0.234.
Set $\theta \leftarrow \theta^t$.
Perform Algorithm 4.1 with proposal $q(\cdot|x) = q(\cdot|x, \theta)$.
```

Algorithm 4.2 Adaptive Metropolis-Hastings algorithm

with,

$$\mu^{t+1} = \mu^t + \gamma^{t+1}(X^{t+1} - \mu^t) \tag{4.16}$$

$$\Sigma^{t+1} = \Sigma^t + \gamma^{t+1}((X^{t+1} - \mu^t)(X^{t+1} - \mu^t)^T - \Sigma^t) \tag{4.17}$$

where $\{\gamma^t\}_{t>0}$ is a sequence of small numbers, which are formally arbitrary but could influence the performance. As noted by [11], though it is possible to set the sequence to a constant γ, it is more common to set it to a deterministic decreasing sequence such that $\sum_{t\geq1} \gamma^t = \infty$ and $\sum_{t\geq1}(\gamma^t)^2 < \infty$ to allow the effect of adaptation becomes smaller and smaller as the algorithm progresses. This algorithm has also been studied by [9] and others.

One obvious problem with such adaptive scheme is that the resulting chains are no longer Markovian. And the limiting distribution, if it exists, may no longer be π as shown by examples in [11]. It is a common practice to adapt the algorithm up to some point t, and stop the adaptation and use parameter θ^t for all iterations onwards, as seen in Algorithm 4.2. The first part of the generated chain is called the *burn-in period* and is usually discarded afterwards. Estimations are based on iterations after the burn-in period. Some rules for when to stop adaptation was discussed in [11] and references therein.

There are more advanced techniques where the adaptive scheme attains a vanishing adaptation property. Informally, after a long enough period, the adaptive algorithm will only change the parameter θ slightly and eventually it becomes stable. These algorithms require more careful design to assure convergence. The algorithm can be particularly problematic when the adaptation parameter θ approaches the boundaries of its space Θ. For example, in the Normal random walk example above, it is possible that the covariance matrix becomes too large or too small. And in either case, ergodicity of the algorithm may be lost [11]. One possible solution is to construct the algorithm such that the adaptation parameter is bounded. Some general methodologies of ensuring the boundedness and convergence of adaptive MCMC can be seen in literature, such as [8] and [10].

Adaptive schemes are often necessary for realistic applications. For example, consider the PET compartmental model. As seen earlier, the scaling of the Normal random walk influences the performance greatly. However, in a single PET scan, there are about a quarter of a million data sets, each results in a different posterior surface. Manual tuning for each of them is a difficult task. When the random walk algorithm is applied for the real data, we used the adaptive Normal random walk for the parameters $(\phi_{1:r}, \theta_{1:r})$. Using 10,000 iterations as the burn-in period for adaptation, we were able to obtain satisfactory acceptance rates (in the range from 0.2 to 0.4) for the majority of the vast amount of data sets.

4.3.3 *Gibbs sampling*

The Metropolis-Hastings algorithm is generic in the sense that it requires minimal knowledge of the target distribution to construct a valid sampler (though not necessarily an efficient one). There also exists a class of MCMC algorithms that are more model dependent and they can use the (conditional) features of the target distribution to construct potentially more efficient samplers. One more impor-

```
Draw $X^0 \sim \mu$ where $\mu$ is the initial condition.
Set $t \leftarrow 0$.
repeat
  for $i = 1, \dots, p$ do
    Draw $X_i^{t+1} \sim \pi_i(x_i^{t+1} | x_1^{t+1}, \dots, x_{i-1}^{t+1}, x_{i+1}^t, \dots, x_p^t)$.
  end for
  Set $t \leftarrow t + 1$.
until Sufficient many samples have been produced.
```

Algorithm 4.3 Gibbs sampling (deterministic scan)

tant of them is the *Gibbs sampling*. As we will see later, it is a special case of the Metropolis-Hastings algorithm.

A Gibbs sampler, as first introduced by [58], assumes that the random variable X can be written as $X = (X_1, \dots, X_p)$, where X_i's are either unidimensional or multidimensional. Let $\pi_1, \dots, \pi_p$ denote the *full conditionals*, defined by

$$X_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_p \sim \pi_i(x_i | x_1, \dots, x_{i-1}, x_{i+1}, \dots, x_p). \tag{4.18}$$

If sampling from each of these distributions is possible, the associated Gibbs sampler is given by the following algorithm that transits X^t to X^{t+1} in p steps. At each step i (within one iteration), X_i^{t+1} is generated from π_i,

$$X_i^{t+1} \sim \pi_i(x_i^{t+1} | x_1^{t+1}, \dots, x_{i-1}^{t+1}, x_{i+1}^t, \dots, x_p^t). \tag{4.19}$$

This leads to Algorithm 4.3. See [135, chap. 9 and 10] for a full theoretical treatment of the Gibbs sampling.

The Markov chain produced by a Gibbs sampler is irreducible if π satisfies the so-called *positivity condition*: All π_i are positive implies that π is also positive [135, Theorem 10.8]. An easier to verify condition for Harris recurrent is that the transition kernel associated with Algorithm 4.3 is absolutely continuous with respect to π [158]. Some simpler conditions can be found in [76]. Stronger convergence results such as geometrically ergodicity are more difficult to be established for the Gibbs sampling in general.

Intuitively, the decomposition of the joint distribution gives a particular coordinate system with each step only exploring one of the coordinates. It may take many cycles for the sampler to move around the surface of the joint distribution. As shown in [135, note 9.7.1], poor parameterizations or decomposition can lead to slow convergence to the extent of getting into a trapping state. This kind of situations most commonly occur when two highly correlated parameters, say X_{k_1} and X_{k_2}, are updated in separate Gibbs moves. When X_{k_1} is updated, because of its dependency on X_{k_2}, its move is limited, and *vice versa*.

For a particular parameterization of the target distribution, one can use better decompositions to speedup convergence in a Gibbs sampler. There are few general methodologies to solve this problem. The practical rule is to create decompositions as independent as possible. For instance, in the special case that $(X_1, \dots, X_p)$ are mutually independent, then a Gibbs sampler is equivalent to sampling directly from the target distribution. Admittedly, such decompositions, though they exist, can hardly be found in interesting cases, otherwise one would not need to consider an MCMC algorithm in the first place.

Another approach is to reparameterize the target distribution with the same principle of decomposition. For example, in [135, sec. 10.4.1], an example is given for a bivariate Normal distribution. To sample (X_1, X_2) from $\mathcal{N}_2(0, \Sigma)$, where Σ is such that its eigenvalues satisfy $\lambda_{\text{MIN}} \ll \lambda_{\text{MAX}}$ and its eigenvectors correspond to the first and second diagonals of $\mathbb{R}^2$, using a Gibbs sampler operating on $(X_1 + X_2, X_1 - X_2)$ is much faster than that on (X_1, X_2). The reparameterization here is based on the eigenbasis in this example. Note that, this kind of techniques is not unique to the Gibbs sampling. They can also be used for other MCMC algorithms. See, e.g., [74, 61] for more discussions on this topic.

Relation with the Metropolis-Hastings algorithm The Gibbs sampling can be viewed as a special case of the Metropolis-Hastings algorithm. It is equivalent to a composition of Metropolis-Hastings samplers in which at each step a single component is updated using its full conditional as the proposal distribution. It is easy to verify that the acceptance probability is uniformly equal to one [135, Theorem 10.13]. Let $Y = (X_{1:i-1}, Y_i, X_{i+1:p})$ where Y_i is the value proposed with π_i at step i. The acceptance probability is then,

$$\alpha(x, y) = \min\left\{\frac{\pi(y)}{\pi(x)}\frac{\pi_i(x|x_{1:i-1}, x_{i+1:p})}{\pi_i(y|x_{1:i-1}, x_{i+1:p})}, 1\right\}.$$

Rewrite $\pi(x)$ and $\pi(y)$ in the form of conditional densities, it follows

$$\begin{aligned}\alpha(x, y) &= \min\left\{\frac{\pi_i(y_i|x_{1:i-1}, x_{i+1:p})\pi(x_{1:i-1}, x_{i+1:p})}{\pi_i(x_i|x_{1:i-1}, x_{i+1:p})\pi(x_{1:i-1}, x_{i+1:p})}\frac{\pi_i(x_i|x_{1:i-1}, x_{i+1:p})}{\pi_i(y_i|x_{1:i-1}, x_{i+1:p})}, 1\right\} \\ &= \min\{1, 1\} = 1\end{aligned}$$

Random scan Algorithm 4.3 is also called the *deterministic scan* Gibbs sampling, in the sense that the components are updated in a deterministic order $(X_1, \dots, X_p)$. Consequentially, this resulting chain is not reversible. Intuitively, to construct the same Markov chain backward in time, the components need to be updated in the reverse order of the original Gibbs sampler. Another way of doing Gibbs sampling is to use a *random scan* [109], where at each time t, a sequence of integers $(k_1, \dots, k_p)$ is generated, usually uniformly across all permutations of $(1, \dots, p)$, and the components are updated in the order of $(X_{k_1}, \dots, X_{k_p})$. This leads to Algorithm 4.4. The resulting Markov chain is reversible [109]. This property can be useful when applying the Central Limit Theorem [135, sec. 10.1.2].

Completion A common difficulty of the Gibbs sampling is that some of the full conditionals may not be easily sampled from. In some situations, the full conditionals of the target are not explicit at all. For example, missing data models are often in the form,

$$f(\boldsymbol{y}|\theta) = \int f(\boldsymbol{y}, \boldsymbol{z}|\theta)\,\mathrm{d}\boldsymbol{z}.$$

```
Draw $X^0 \sim \mu$ where $\mu$ is the initial condition.
Set $t \leftarrow 0$.
**repeat**
  Draw $(k_1, \dots, k_p) \sim \sigma(1, \dots, p)$ where $\sigma$ is typically a distribution uniform over all permutations of $(1, \dots, p)$.
  **for** $i = 1, \dots, p$ **do**
    Draw $X_{k_i}^{t+1} \sim \pi_{k_i}(x_{k_i}^{t+1} | x_{k_1}^{t+1}, \dots, x_{k_{i-1}}^{t+1}, x_{k_{i+1}}^{t}, \dots, x_{k_p}^{t})$.
  **end for**
  Set $t \leftarrow t + 1$.
**until** Sufficient many samples have been produced.
```

Algorithm 4.4 Gibbs sampling (random scan)

where $\boldsymbol{z}$ is the unobserved data and $f(\boldsymbol{y}, \boldsymbol{z}|\theta)$ is the likelihood function given the complete data $(\boldsymbol{y}, z)$. In these situations, it is possible to use *completion* to construct a Gibbs sampler. A distribution, say η with the following property is chosen,

$$\int \eta(x, y) \, \mathrm{d}\, y = \pi(x). \tag{4.20}$$

In other words, π is a marginal of η. Then the Gibbs sampler is constructed with the full conditionals of η instead of π. The sub-chain of the resulting Markov chain that corresponds to the marginal π is then π-invariant. When such a technique is used, there are many possible choices of η to complete π. Some applications provide natural choices, such as the aforementioned missing data models. Simple examples are mixture models. Instead of direct sampling from the distribution that is proportional to the summation of some distributions, an auxiliary location variable can be introduced to complete the distribution. In particular, given the target distribution of the form,

$$\pi(x) \propto \sum_{j=1}^{r} w_i \pi_j(x)$$

The location variable, say Z can be introduced such that $X|z = j \sim \pi_j$ and $\Pr(Z = j) = w_j$. In some cases, by introducing such an auxiliary variable, it is easier to construct a Gibbs sampler.

4.3.4 *Reversible jump MCMC*

The reversible jump MCMC (RJMCMC) algorithm, introduced by [63], is a technique widely used for simulations where the dimension of the parameter space is not fixed. In the context of Bayesian model selection, it can be used for inference of the full posterior $\pi(\theta_k, \mathcal{M}_k|\boldsymbol{y})$, which is defined on the space $\Theta = \bigcup_{k\in\mathcal{K}}(\{\mathcal{M}_k\} \times \Theta_k)$. In situations where this can be reduced to the estimation of the Bayes factor (Section 3.2.2), techniques reviewed so far in this chapter can be used. However, when $\mathcal{K}$ is (infinite) countable, or for other reasons, direct inference on the full posterior distribution is desired, RJMCMC is the most widely used technique. The RJMCMC and other algorithms that are capable of simulating the full posterior distribution are not only conceptually appealing, but also sometime necessary. In the scenarios where a large number of models are possible and it is difficult to narrow it down to a manageable set of candidate models, using RJMCMC can be potentially more efficient than performing simulations for each model when model selection is of interest.

The RJMCMC algorithm adapts the Metropolis-Hastings algorithm to construct transition kernels to simulate the full posterior. Instead of a single type of moves defined by a proposal distribution, a countable set of moves is considered, say $m \in \mathcal{M}$. Each type of moves is capable of moving the current state of the Markov chain between, say Θ_k and $\Theta_{k'}$, the parameter space of model $\mathcal{M}_k$ and $\mathcal{M}_{k'}$ (where in the case of $k = k'$, the move is similar to those in an MCMC algorithm on a fixed dimension space). At state $\theta_k \in \Theta_k$, a move of type m together with a new state $\theta_{k'} \in \Theta_{k'}$ are proposed according to $q_m(\theta_{k'}|\theta_k)r_m(\theta_k)$, where $r_m(\theta_k)$ is the probability of choosing a type m move when at state θ_k; and $q_m(\theta_{k'}|\theta_k)$ is the proposal kernel for the new state when a move of type m is made. Usually, these moves are designed in pairs. For type of moves m, there is an inverse type, say m', that can move the state $\theta_{k'} \in \Theta_{k'}$ to $\theta_k \in \Theta_k$. The move is accepted with probability,

$$\alpha(\theta_k, \theta_{k'}) = \min\left\{1, \frac{\pi(M_{k'})\pi(\theta_{k'}|M_{k'})p(\boldsymbol{y}|\theta_{k'}, M_{k'})}{\pi(\mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)} \frac{q_{m'}(\theta_k|\theta_{k'})r_{m'}(\theta_{k'})}{q_m(\theta_{k'}|\theta_k)r_m(\theta_k)}\right\}. \quad (4.21)$$

In practice, the proposed new state $\theta_{k'}$ is often implemented by drawing a vector of continuous random variables, say u, independent of θ_k and a deterministic bijection of vector (θ_k, u) to $\theta_{k'}$, say $\theta_{k'} = T(\theta_k, u)$. The inverse of the move from $\theta_{k'}$ back to θ_k, m', then uses the inverse of this transformation. Through a simple change of variable, the conditional density $q_m(\theta_{k'}|\theta_k)$ can be expressed in terms of the density of vector u, say $q(u)$. The acceptance probability becomes

$$\alpha(\theta_k, \theta_{k'}) = \min\left\{1, \frac{\pi(M_{k'})\pi(\theta_{k'}|M_{k'})p(\boldsymbol{y}|\theta_{k'}, M_{k'})}{\pi(\mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)} \frac{r_{m'}(\theta_{k'})}{r_m(\theta_k)} \frac{1}{q(u)} \left|\frac{\partial \theta_{k'}}{\partial(\theta_k, u)}\right|\right\}, \quad (4.22)$$

where the last term is the determinant of the Jacobian transformation. The design of efficient between-model moves is often difficult, and the mixing of these moves largely determines the performance of the algorithm.

It should be noted that, in the above we only described the move that transits the parameters from the space of one model into another. As mentioned earlier, in practice, RJMCMC moves are designed in pairs. In each pair, the two moves are capable of moving the parameters between two models. For each type of move that generates parameters $\theta_{k'}$ given θ_k, an inverse move can be constructed. At each iteration, there may be multiple steps. One step is to update the parameters without changing the model. Other steps may move the parameters between models. At each of the later step, a pair of moves is implemented, and with equal probabilities (i.e., $r_m(\theta_k) = r_{m'}(\theta_k{}') = 0.5$), one type of the move in the pair is performed.

The main difficulties lie in the choice of cross-model proposals and the bijection T. Though the mapping T theoretically is quite flexible, its creation and optimization can be quite difficult in practice. This is particularly true when the parameter space is complicated. In some extreme cases, creating a valid kernel is already difficult. For example, in multimodal models, where RJMCMC has gained substantial attention, information available in posterior distributions of any given model does not characterize modes that exist only in models of higher dimension; and thus a successful between-model move between these dimensions becomes difficult [83]. Inefficient proposals result in Markov chains that are slow to explore the whole parameter space. However, the natural ideas of neighborhood and others, which proved to be useful concepts for within model simulations, may no longer be intuitive in the variable dimension model settings. For instance, when a cross-model occurs, the previous state of the parameters, which may be in a high probability region of the model in the last iteration, when transformed might be in a low probability region of the model of the current iteration. In addition, RJMCMC does not characterize all models well as some may be visited by the chain only rarely. This may not be a problem when the model is indeed of low posterior probability and there is little interest in such models. However, in some cases it will be difficult to determine whether the low acceptance rates of between model moves results from actual characteristics of the posterior or from a poorly adapted proposal kernel.

Some discussion of the optimization of the cross-model moves can be found in [64]. Also the adaptive scheme for the Metropolis-Hastings algorithm has been extended for RJMCMC, for example [70]. However little other work is known for the actual performance of this kind of improvement to RJMCMC. In [65] a method called *delayed rejection* was discussed. In this method, a rejection of a proposal does not immediately lead to the acceptance of current state, instead a second proposal is attempted. Their numerical results showed efficiency improvement but with increased computation cost.

4.3.5 *Population MCMC*

Population-based methods have been considered in recent research. An entire family of such algorithms, sequential Monte Carlo, is considered in Chapter 5. Another algorithm, population MCMC, which has seen applications in the area of Bayesian model comparison, is reviewed in this section.

Population MCMC operates by constructing a sequence of distributions $\{\pi_t\}_{t=0}^{T}$ with at least one of them being the target distribution π. Parallel MCMC chains are simulated for each of these distributions. In addition, the chains interact with each other by swapping or crossover moves, which allows fast mixing chains to "lend" information to slow mixing chains. The outputs are therefore samples that approximate the product $\prod_{t=0}^{T} \pi_t$ with the target distribution being a marginal.

Different choices of the sequence of distributions are possible. One commonly used in practice is called tempering. For a target π, a sequence $\{\pi_t\}_{t=0}^{T}$ is constructed such that,

$$\pi_t(x) \propto [\pi(x)]^{\alpha(t/T)} \tag{4.23}$$

where the mapping $\alpha : [0,1] \to [0,1]$ is monotonically increasing with $\alpha(1) = 1$ (also see [111] for similar annealing schemes). Other similar schemes can be constructed. For example, in the context of Bayesian modeling where π is the posterior distribution $\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_k) \propto \pi(\theta_k|\mathcal{M}_k) f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$, one can construct a sequence

$$\pi_t(\theta_k) = \pi(\theta_k|\mathcal{M}_k)[f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)]^{\alpha(t/T)} \tag{4.24}$$

where the monotonically increasing mapping α satisfies $\alpha(0) = 0$ and $\alpha(1) = 1$. Therefore the sequence of distributions moves smoothly from the prior, which usually can be sampled from easily, into the posterior.

The algorithm targets the distribution $\prod_{t=0}^{T} \pi_t$. After initializing T Markov chains for each of the marginals $\{\pi_t\}_{t=0}^{T}$ with a common support E, at each iteration, two types of moves are performed. One is *local* moves, sometimes termed *mutation*, that advances each chain individually using an MCMC algorithms such as the Metropolis-Hastings algorithm or the Gibbs sampling. One may select one chain at random in each iteration to mutate or advance all chains in parallel. The other type is *global* moves. The purpose is to allow fast mixing chains to transfer information into slowly mixing chains. In each global move, two chains, say with indices k_1 and k_2, are selected. Let X_{k_1} and X_{k_2} denote their current states. Two new states, Y_{k_1} and Y_{k_2} are proposed according to conditional distributions

```
for $k = 0, \dots, T$ do
  Draw $X_k^0 \sim \mu_k$ where $\mu_k$ is the initial condition.
end for
Set $t \leftarrow 0$.
repeat
  for $k = 0, \dots, T$ do
    Draw $X_k^{t+1} \sim K_k(x_k^x, x_k^{t+1})$ where $K_k$ is a $\pi_k$-invariant Markov kernel.
  end for
  Draw $k_1$ and $k_2$, $k_1 \neq k_2$, from $\{0, \dots, T\}$ such that the distribution of $(k_1, k_2)$
  is uniform over all possible permutations.
  Draw $Y_{k_1}^{t+1} \sim q_{k_1}(y_{k_1}^{t+1}|x_{k_2}^{t+1})$ and $Y_{k_2}^{t+1} \sim q_{k_2}(y_{k_2}^{t+1}|x_{k_1}^{t+1})$.
  Compute $\alpha$, the acceptance probability according to the Metropolis-
  Hastings rule ($\alpha((x_{k_1}^t, x_{k_2}^t)$ in Equation (4.25) for an exchange move and
  $\alpha(x_{k_1}^t, x_{k_2}^t, y_{k_1}^{t+1}, y_{k_2}^{t+1})$ in Equation (4.26) for a crossover move). With proba-
  bility $\alpha$, set $X_{k_1}^{t+1} \leftarrow Y_{k_1}^{t+1}$, $X_{k_2}^{t+1} \leftarrow Y_{k_2}^{t+1}$.
  Set $t \leftarrow t + 1$.
until Sufficiently many samples have been produced.
```

Algorithm 4.5 Population MCMC with parallel updating.

$q_{k_1}(y_{k_1}|x_{k_2})$ and $q_{k_2}(y_{k_2}|x_{k_1})$, respectively. That is, the proposed new state of each chain depends on the current state of the other. The new states are accepted with the usual Metropolis-Hastings acceptance probability. This leads to Algorithm 4.5. There are several approaches of the global move [82]. Two more widely used are the following.

Exchange The exchange move selects two chains at random, say k_1 and k_2, and propose to exchange the states between them. That is, the proposal distribution $q_{k_1}(y_{k_1}|x_{k_2})$ is defined by $\Pr(Y_{k_1} = x_{k_2}|x_{k_2}) = 1$ (similarly for $q_{k_2}(y_{k_2}|x_{k_1})$). The proposed exchange is accepted or rejected according to the Metropolis-Hastings acceptance probability,

$$\alpha(x_{k_1}, x_{k_2}) = \min\left\{\frac{\pi_{k_1}(x_{k_2})\pi_{k_2}(x_{k_1})}{\pi_{k_1}(x_{k_1})\pi_{k_2}(x_{k_2})}, 1\right\}. \tag{4.25}$$

For this to work, usually the two chains are chosen such that they are adjacent to each other in the sense that one is chosen randomly and the other is selected to be the one most close to it. For example, in the tempering scheme, usually a chain with index $k_1 \in \{0, 1, \dots, T-1\}$ is chosen randomly and it is proposed to be exchanged with the chain with index $k_2 = k_1 + 1$. The delayed rejection approach in [65] (see Section 4.3.4) can also be used in the exchange moves. Thus two chains in some sense that are very different can also be chosen. In either case, the key is that the chains chosen to be exchanged are chosen uniformly over all chains.

Crossover Another type of global move, called crossover was mentioned in [107]. Instead of proposing to exchange the whole states X_{k_1} and X_{k_2}, after the two chains are chosen, only parts of the two states are proposed to be exchanged. Suppose the state X can be partitioned into $X = (X_1, \dots, X_p)$ in the same way for each chain. Then a random position, say l is chosen and the position l of X_{k_1} is proposed to be exchanged with its counter-part in X_{k_2}. The acceptance probability is the same as Equation (4.25) with suitable notation changes. Let $X_{k_i} = (X_{k_i,1}, \dots, X_{k_i,p})$ for $i = 1$ and 2 denote the current states and

$$\begin{aligned} X_{k_1}{}' &= (X_{k_1,1}, \dots, X_{k_1,l-1}, X_{k_2,l}, X_{k_1,l+1}, \dots, X_{k_1,p}) \\ X_{k_2}{}' &= (X_{k_2,1}, \dots, X_{k_2,l-1}, X_{k_1,l}, X_{k_2,l+1}, \dots, X_{k_2,p}) \end{aligned}$$

denote the proposed states. Then the acceptance probability is,

$$\alpha(x_{k_1}, x_{k_2}, x_{k_1}{}', x_{k_2}{}') = \min\left\{\frac{\pi_{k_1}(x_{k_1}{}')\pi_{k_2}(x_{k_2}{}')}{\pi_{k_1}(x_{k_1})\pi_{k_2}(x_{k_2})}, 1\right\} \tag{4.26}$$

In [82] it was found that the cross over move can be more efficient than the exchange move.

Population MCMC algorithm can be more efficient than simulating from a single chain. Consider the situation where the MCMC algorithm targeting distributions π_1 and π_2 might be trapped. If π_1 and π_2 are similar in the sense of the shape of the locations of local modes. And each of them are trapped within different modes. The global move, say the exchange move, proposes to exchange the values from one high probability region with those in another high probability region. It is more likely that such an exchange is accepted than the MCMC algorithm jumps into the other modes itself. Those chains that mix fast can explore the parameter space more efficiently than those mix slower and are more likely to visit all the high probability regions. Through the global moves they propose values in other high probability regions to slowly mixing chains to help them avoid trapping states.

Optimal placement of distributions As discussed earlier, population MCMC allows efficient simulation of previously difficult problem, though at a cost of increasing computational cost. However, the algorithm requires another layer of optimization in addition to the mixing speed of each local move – the placement of the sequence of distributions $\{\pi_t\}_{t=0}^T$. If too many chains are present, the information can take many global moves to transfer from fast mixing chains to slowly mixing ones. If there are too few chains and they are placed far apart from each other, the global moves are likely to have small acceptance rates. In [13], based on the idea of maximizing the average information exchanged at each iteration, it was recommended that an optimal placement of the distributions should have global acceptance rate of around 0.234. The optimal placement of the distributions can be obtained iteratively if $\{\pi_t\}_{t=0}^T$ belongs to a family of distributions, say $\pi_\alpha = \pi(\cdot|\alpha)$, indexed by α. The algorithm first finds α_0 and α_1 such that the population MCMC algorithm operating on $\{\pi_{\alpha_t}\}_{t=0}^1$ has a global acceptance rate close to 0.234. For example, if a tempering scheme is used, one can set $\alpha_0 = 0$ and use a binary search algorithm to find α_1 since the smaller α_1, the higher the acceptance rate. The algorithm proceeds in the same way to find α_t for $t > 1$.

4.3.6 *Convergence diagnostic*

One important issue of MCMC algorithms is their speed of convergence. It is well understood yet sometimes overlooked in practice. In the previous sections we demonstrated for many algorithms, under fairly general conditions, the chains produced are ergodic, or even geometrically ergodic (random walk). In some cases the chain can be uniformly ergodic (independent Metropolis-Hastings algorithm). However, such development provides little insight on how many iterations the algorithm should be run to produce accurate estimates.

Convergence of an MCMC algorithm is assessed by monitoring certain statistics of samples. This process is also called *convergence diagnostic*. There are two types of convergence [135, chap. 12] widely used in practice. As we will see later, a convergence diagnostic can at best determine that a chain has not converged yet. One cannot be certain that a chain does converge.

Convergence to the stationary distribution

It might seem that a minimal requirement for samples from an MCMC algorithm to be used to approximate a target distribution π, is that the chain converges to this stationary distribution. However, π is only the limiting distribution and the stationarity is at best achieved asymptotically. Nonetheless, one possible assessment of such convergence is to obtain bounds on the total variation norm,

$$\|K^n(x,\cdot) - \pi\|_{TV}$$

where $K^n(x,\cdot)$ is the distribution of samples at the n^{th} iteration. However, obtaining analytical bounds can be prohibitively difficult.

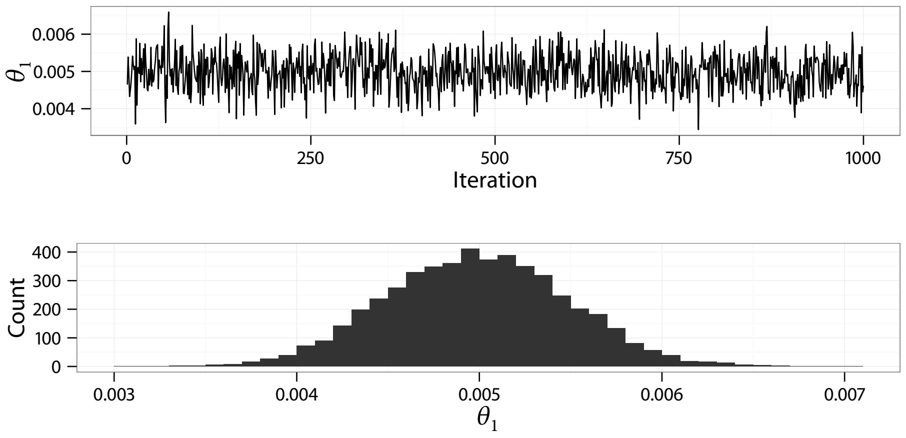


Figure 4.3 Trace and histogram plots of parameter θ_1 from a MCMC sampler for PET model with three components and non-informative priors without ordering. The trace plot has 1,000 samples and the histogram plot has 10,000 samples. The sampler is not well calibrated.

Graphical approach A natural empirical approach is to draw plots of simulated samples to detect non-stationary behaviors. For instance, [53] drew sequence of the samples $\{X^t\}_{t\geq 1}$ against the time t, which is a functionality now commonly seen in softwares that implement MCMC algorithms.

It should be emphasized that, even when the plots appears to show stationary behavior, it is still possible that the algorithm has not converged or explored the support of the target distribution surface efficiently. For example, consider the three-compartments PET model and non-informative priors without ordering (that is, parameters such as (ϕ_1, θ_1) and (ϕ_2, θ_2) are exchangeable, see Section 2.2). We use one of the real data sets and constructed a random walk algorithm (see Section 4.3.2). Figure 4.3 shows the trace and histogram plots of parameter θ_1. It appears that the MCMC chain has converged well. However, from the properties of the model, it is known that this parameter has at least three local modes. In fact, this sampler has been trapped into one of them. In Figure 4.4 the trace and histogram plots of the same sampler but with better calibrated proposal scales are shown.

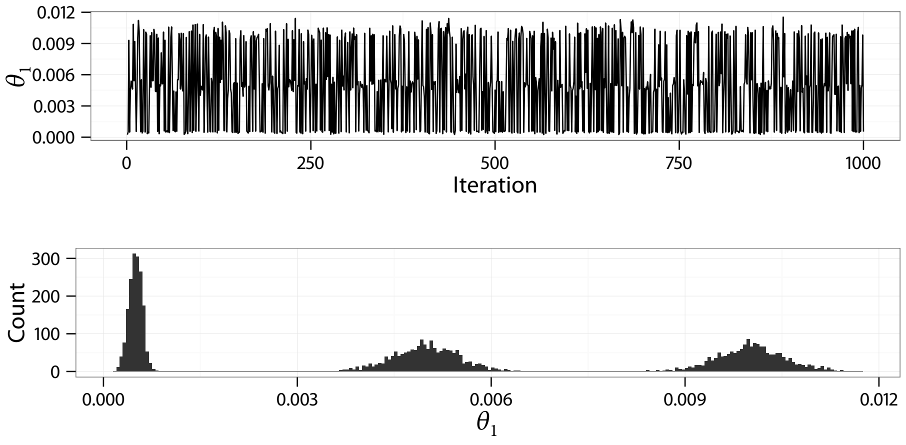


Figure 4.4 Trace and histogram plots of parameter θ_1 from a MCMC sampler for PET model with three components and non-informative priors without ordering. The trace plot has 1,000 samples and the histogram plot has 10,000 samples. The sampler is well calibrated.

Non-parametric test Standard non-parametric tests can be applied in stationarity assessment. This is based on the idea that, if the chain is stationary, then X^{t_1} and X^{t_2} have the same distribution for any two arbitrary time points t_1 and t_2. Therefore standard tests can be used to compare the distribution of samples $(X^t, \dots, X^{t+p-1})$ and $(X^{t+p}, \dots, X^{t+2p})$. It should be noted that the correlations between samples should be taken into consideration. One solution is to use sub-samples. A *batch size* G is introduced. Quasi-independent samples $(X^{t_1+G}, X^{t_1+2G}, \dots, X^{t_1+pG})$ and $(X^{t_2+G}, X^{t_2+2G}, \dots, X^{t_2+pG})$ are used to conduct the tests. See [135, sec. 12.2.2] for some examples of such tests.

A simpler statistic to use, as seen in [55], is the ratio of the variance of the last few samples to that of all samples. Formally, for some function φ, define the following ratio,

$$R_T^N = \frac{\mathrm{var}[\varphi(X^{N-T+1}, \dots, X^N]}{\mathrm{var}[\varphi(X^1, \dots, X^N]} \tag{4.27}$$

A value of R_T^N between 0.9 and 1.1 was recommended. For example, Figure 4.5

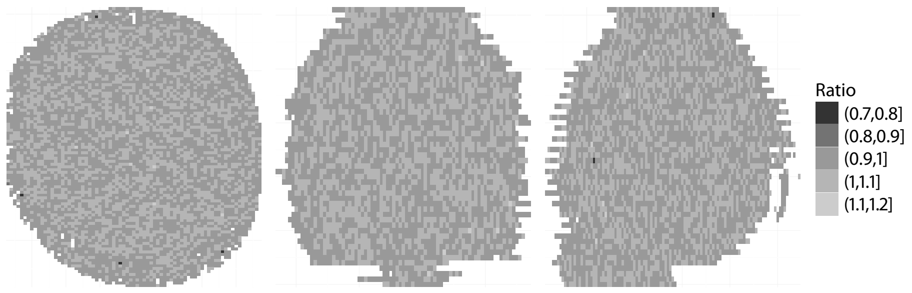


Figure 4.5 Convergence diagnostics for the three-compartments PET using ratio of variance of V_D estimates using final 1,000 samples to that of all 10,000 post burn-in samples. A value of the ratio close to one indicates that there are no apparent signs that the MCMC algorithms do not converge well. A value within the range of $[0.9, 1.1]$ is considered to be acceptable.

shows the ratios of the variance of V_D estimated using the final 1,000 samples to that of all 10,000 post burn-in (the iterations used to adapt the sampler to optimal acceptance rates) samples, for real PET scan data sets. For the majority of data sets, the ratios fall in the desired interval.

Convergence of averages

In [165] it was proposed to use the cumulative sums and plot the partial differences,

$$D_N^t = \sum_{i=1}^{t}(\varphi(X^{(i)}) - S^N), \qquad t = 1, \dots, N, \tag{4.28}$$

where

$$S^N = \frac{1}{N}\sum_{i=1}^{N}\varphi(X^{(i)}) \tag{4.29}$$

is the final average. A simple variant of this method is to plot the average of the first t samples, S^t. The use of these quantities can be appealing because they directly measure the stability of the estimator of interest. For instance, Figure 4.6 shows the posterior mean estimate of V_D for a three-compartments PET model. The posterior mean from five samplers initialized with different values converge to the same value.

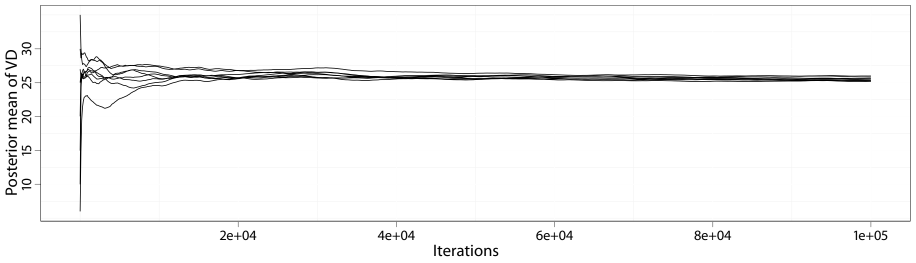


Figure 4.6 Estimates of V_D when starting the MCMC chain from different values for a typical data set of a PET model with three component. Three slice of the brain are shown in the plot. Each are close to the middle of the brain along each of the three axises in the three-dimensional space.

A more robust approach was proposed in [133]. The idea is to use several convergent estimators based on the same samples. The chain is first iterated until all estimators are close to each other in the sense that the differences are smaller than a preset tolerance value. And then onwards simulations are used for inferences. One obvious estimator is the empirical average of all samples used for estimation. Another one is similar to the importance sampling estimator,

$$\varphi^N_{\text{MCMC-IS}} = \sum_{i=1}^{N} \varphi(Y^{(i)}) \frac{\pi(Y^{(i)})}{q(Y^{(i)}|X^{(i)})} \tag{4.30}$$

where $\{Y^{(i)}\}_{i=1}^N$ are samples from $q(y^t|x^t)$, a distribution that depends on the current state X^t. When the distributions are known only up to some normalizing constants, then a variant similar to that in Equation (4.7) is used. Unlike the importance sampling in Section 4.2, this estimator is based on dependent samples instead of i.i.d samples. However, as shown in [135, Lemma 12.11], the weighted terms in the sum are uncorrelated.

In the particular case of the Gibbs sampling, the distribution,

$$q(y^t|x^{t-1}) \propto \prod_{i=1}^{p} \pi_i(y_i^t|y_{1:i-1}^t, x_{i+1:p}^{t-1}) \tag{4.31}$$

is a natural choice to be used as the proposal distribution and samples from Gibbs sampler can be used. In this case, $Y^{(i)} = X^{(i)}$.

For the generic Metropolis-Hastings algorithm, the samples simulated from the proposal distributions (both those accepted and rejected) can be recycled to calculate the importance sampling estimate. In this case, $Y^{(i)}$ is the proposed value, $X^{(i)}$ are the accepted values and the distribution $q(\cdot|x^t)$ is simply the proposal distribution. This approach is more robust in the sense that a Markov chain in a trapping state is more likely to be detected than the simple plots of averages.

For example, consider an independent Metropolis-Hastings algorithm whose proposal η has only one high probability region A while the target π has two high probability regions A and B such that $Pr(x \in A) \approx \Pr(x \in B)$. The chain is likely to be trapped in A and S^N might appear to be stable. However, $\varphi^N_{\text{MH-IS}}$ is much less stable since those values occasionally proposed within B, though very likely to be accepted, they also have extreme large value of the weight $\pi(Y^{(i)})/\eta(Y^{(i)})$. This leads to a large variance of estimator $\varphi^N_{\text{MCMC-IS}}$. In this case, $\varphi^N_{\text{MH-IS}}$ is stable when values of high target density values are also proposed frequently.

4.3.7 *Application to Bayesian model comparison*

It is clear that RJMCMC can be used directly for the purpose of Bayesian model selection, as it generates samples from the full posterior with the model posterior distribution $\pi(\mathcal{M}_k|\boldsymbol{y})$ as a marginal.

For within model simulations, that is the algorithms generate Markov chains targeting the posterior distribution $\pi(\theta_k|\boldsymbol{y}, \mathcal{M}_k) \propto f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)$, the dependent samples can be used for the purpose of Bayesian model comparison for a finite a set of models through approximating the marginal likelihood and thus the Bayes factor, which is the ratio of the marginal likelihood of two models. A few methods are discussed here.

Generalized harmonic mean estimator

Recall that, the marginal likelihood is written as,

$$p(\boldsymbol{y}|\mathcal{M}_k) = \int f(\boldsymbol{y}|\theta_k, \mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)\,\mathrm{d}\theta_k.$$

For the purpose of simplicity, in this section we drop the dependency on the model $\mathcal{M}_k$ and simply write $p(\boldsymbol{y}) = \int f(\boldsymbol{y}|\theta)\pi(\theta)\,\mathrm{d}\theta$. With samples generated by an MCMC algorithm targeting the posterior distribution $\pi(\theta|\boldsymbol{y}) \propto f(\boldsymbol{y}|\theta)\pi(\theta)$ available, say $\{\theta^{(i)}\}_{i=1}^{N}$, an estimator of $p(\boldsymbol{y})$ can be obtained by the harmonic mean [121],

$$\widehat{p(\boldsymbol{y})}_{\text{HM}}^{N} = \left(\frac{1}{N}\sum_{i=1}^{N}\frac{1}{f(\boldsymbol{y}|\theta^{(i)})}\right)^{-1} \tag{4.32}$$

Unfortunately this estimator can suffer instability problem when samples with small likelihoods are generated. In fact this estimator does not always have a finite variance and therefore in general does not satisfy a Central Limit Theorem (CLT). An improvement seen in [96] is,

$$\widehat{p(\boldsymbol{y})}_{\text{GHM}}^{N} = \left(\frac{1}{N}\sum_{i=1}^{N}\frac{\gamma(\theta^{(i)})}{f(\boldsymbol{y}|\theta^{(i)})\pi(\theta^{(i)})}\right)^{-1}, \tag{4.33}$$

where γ is a proper density function. This is based on the identity,

$$\frac{1}{p(\boldsymbol{y})} = \int \frac{\gamma(\theta)}{p(\boldsymbol{y},\theta)}\frac{p(\boldsymbol{y},\theta)}{p(\boldsymbol{y})}\,\mathrm{d}\theta = \int \frac{\gamma(\theta)}{p(\boldsymbol{y},\theta)}\pi(\theta|\boldsymbol{y})\,\mathrm{d}\theta \tag{4.34}$$

It can be seen that the distribution γ plays a role similar to that of the target distribution for the importance sampling in the sense that the posterior distribution now acts as a proposal distribution. For similar reasons, high efficiency is most likely to be obtained when γ is roughly proportional to $f(\boldsymbol{y}|\theta)$ [96]. The above equation suggests that the estimator has a finite variance if the tails of γ are thin enough compared to the posterior distribution $\pi(\theta|\boldsymbol{y})$. In particular, the choice of γ shall ensures that the value $\gamma(\theta)/p(\boldsymbol{y},\theta)$ is at least bounded and vanishes to zero at the tails of the likelihood function. In [52] the use of a multivariate Normal distribution with moments approximated from the samples as a natural choice of γ

was suggested. Though Normal distributions often have thinner tails than others, it is not generally true and it shall be considered when applied to specific algorithms. On the other hand, if more knowledge of the structure of the posterior or the likelihood function is available, other distributions such as a multivariate t distribution, can be used in place of the multivariate Normal distribution. Even though it has heavier tails, it might be more flexible in the sense that it is possible to mimic the posterior distribution more closely.

The variance of the estimator can also be estimated for $1/\widehat{p(\boldsymbol{y})}^{N}_{\text{GHM}}$ from the posterior samples through,

$$\widehat{\text{var}}\left[\frac{1}{\widehat{p(\boldsymbol{y})}^{N}_{\text{GHM}}}\right] = \frac{1}{N^2}\sum_{i=1}^{N}\left(\frac{\gamma(\theta^{(i)})}{f(\boldsymbol{y}|\theta^{(i)})\pi(\theta^{(i)})} - \frac{1}{\widehat{p(\boldsymbol{y})}^{N}_{\text{GHM}}}\right)^2. \tag{4.35}$$

The above estimator provides a way of monitoring the convergence of the estimator. Though more stable than the harmonic mean estimator $\widehat{p(\boldsymbol{y})}^{N}_{\text{HM}}$, this generalized estimator still requires considerable care in the implementation, especially the choice of the density γ, to ensure good performance and indeed a finite variance estimator. More discussion of the stability of the harmonic mean and related estimators can also be found in [131].

The method described above can be used for most MCMC algorithms, such as the Metropolis-Hastings algorithm, the Gibbs sampling and population MCMC (by only using the sub-chain that corresponds to the distribution of interest).

Results for PET compartmental model We conclude the discussion on the generalized harmonic mean estimator with results for the PET compartmental model with real data. For a r-compartments PET model (see Section 2.2), a Student t distributed error structure (see Section 2.4), and informative priors (see Section 3.2.3), we construct a random Metropolis-Hastings algorithm with four blocks. Again, recall the parameterization in Section 2.2,

1. Update $\phi_{1:r}$ with a multivariate Normal random walk proposal.
2. Update $\theta_{1:r}$ with a multivariate Normal random walk proposal.

3. Update τ with a Normal random walk proposal on the logarithm scale, i.e., on $\log\tau$.
4. Update ν with a Normal random walk proposal on the logarithm scale, i.e., on $\log\nu$.

There are a 10,000 iterations used for adaptation (the burn-in period) and 10,000 iterations are used for estimation. Results in [167] showed that the long burn-in period is more than enough for the majority of the samplers to converge well. The generalized harmonic mean estimator $\hat{\varphi}^{N}_{\mathrm{GHM}}$ is used to compute the Bayes factor. The model selection results, along with those from AIC and BIC methods for the purpose of comparison, are shown in Figure 4.7. It can be seen that the Bayesian model selection results shows more plausible structure than the information criteria. With the information criteria, the distribution of the model orders across the image appears to be more or less random while the Bayesian results show some structure with more active regions within the brain having higher order models. See also the discussions in [167] and references therein.

As usual for real data, it is difficult to determine if the selected models are indeed the "true model". However, the Bayesian model selection results are more plausible for two reasons. First, in this experiment, those regions with higher model orders are also regions where the concentrations of the tracers are higher. The greater uptake of tracers usually indicates higher level biochemical activities and hence models with two or three compartments are more plausible than a model with one compartment. Second, the regions outside the brain are regions where no biochemical reactions could possibly happen (they are not actually part of the brain). They are not masked out in the three-dimensional images to avoid masking out regions that are actually of interest. The Bayesian model comparison method is able to correctly identify that there is only one compartment in those regions while other methods fail to do so.

The convergence results for this simulation were already shown in Figure 4.5. It should be noted that, though as shown in Figure 4.6, accurate estimation of the parameter V_D does not really need this many iterations. However, accurate

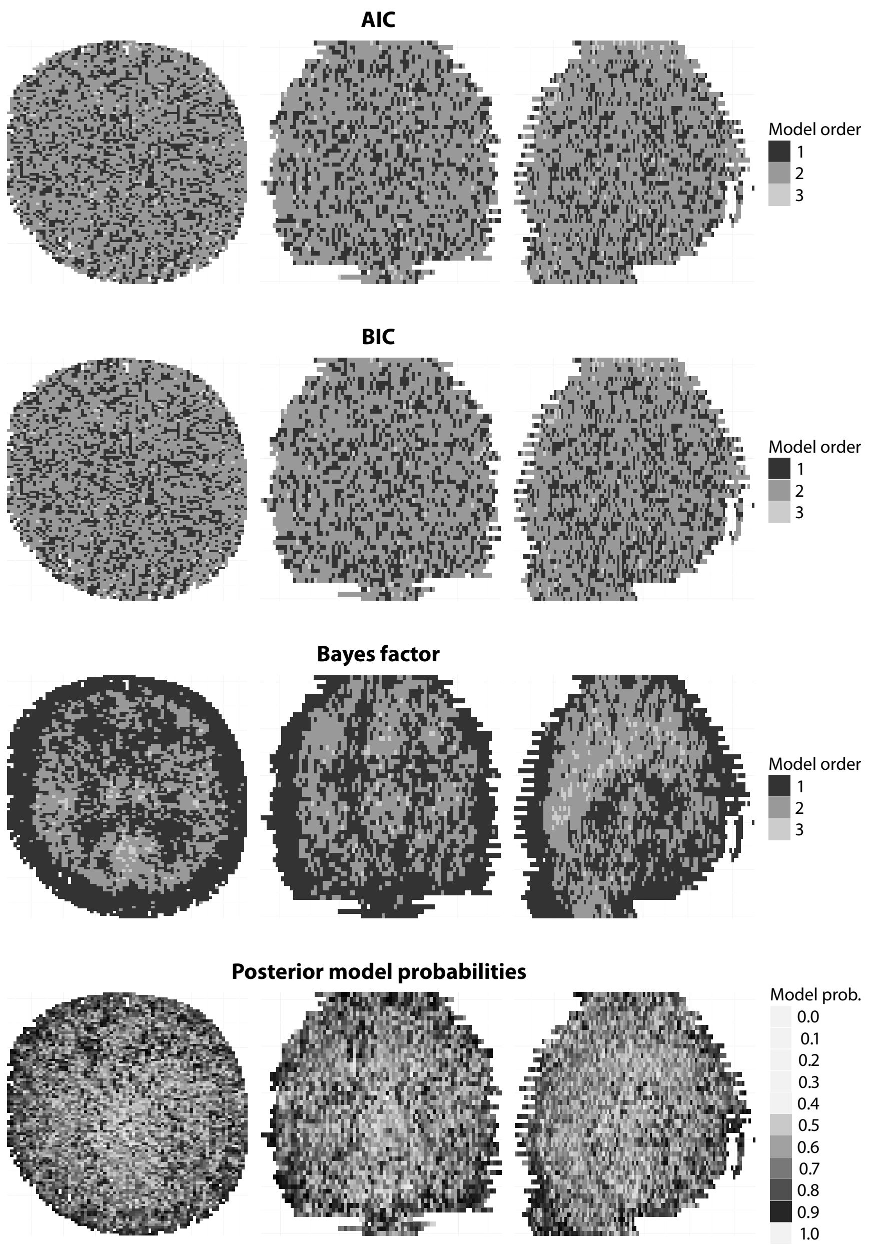


Figure 4.7 Model selection results for PET model using real data set. In each row, three slice of the brain are shown in the plot. Each are close to the middle of the brain along each of the three axises in the three-dimensional space. From top to bottom: Model order selected by AIC$_c$ (see Section 3.1.3); Model order selected by BIC (see Section 3.2.2); Model order selected by using Bayesian model comparison with marginal likelihood approximated by generalized harmonic mean estimator; The posterior model probability $\pi(\mathcal{M}_k|\boldsymbol{y})$ (see Section 3.2.1) with uniform prior model probability $\pi(\mathcal{M}_k)$.

estimation of the marginal likelihood requires considerably more samples.

Estimator using the Gibbs sampling

In the particular case of the Gibbs sampling, [28] provides an alternative estimator based on that the identity,

$$p(\boldsymbol{y}) = \frac{f(\boldsymbol{y}|\theta)\pi(\theta)}{\pi(\theta|\boldsymbol{y})}, \tag{4.36}$$

holds for any value of θ. Therefore an estimator can be obtained by substituting θ with a specific value, say θ^*, which is usually chosen from the high probability region of the posterior distribution and approximating the denominator using outputs from the Gibbs sampler.

Formally, assume it is possible to construct a Gibbs sampler for the decomposition $\theta = (\theta_1, \dots, \theta_p)$. Write $\pi(\theta|\boldsymbol{y})$ as,

$$\pi(\theta|\boldsymbol{y}) = \pi(\theta_1|\boldsymbol{y}) \prod_{i=2}^{p} \pi(\theta_i|\boldsymbol{y}, \theta_{1:i-1}) \tag{4.37}$$

and given θ^*, we have the estimator,

$$\widehat{p(\boldsymbol{y})}^N_{\text{GS}} = \frac{f(\boldsymbol{y}|\theta^*)\pi(\theta^*)}{\pi(\theta_1^*|\boldsymbol{y}) \prod_{i=2}^{p} \pi(\theta_i^*|\boldsymbol{y}, \theta_{1:i-1}^*)}. \tag{4.38}$$

The value of $\pi(\theta_1^*|\boldsymbol{y})$, the marginal ordinate of $\pi(\theta|\boldsymbol{y})$ can be approximated with output from the Gibbs sampler. For example,

$$\hat{\pi}^N(\theta_1^*|\boldsymbol{y}) = \frac{1}{N}\sum_{i=1}^{N} \pi(\theta_1^*|\boldsymbol{y}, \theta_{2:p}^{(i)}) \tag{4.39}$$

since

$$\begin{aligned}
\pi(\theta_1^*|\boldsymbol{y}) &= \int \pi(\theta_1^*, \theta_{2:p}|\boldsymbol{y}) \,\mathrm{d}\theta_{2:p} \\
&= \int \pi(\theta_1^*|\boldsymbol{y}, \theta_{2:p})\pi(\theta_{2:p}|\boldsymbol{y}) \,\mathrm{d}\theta_{2:p} \\
&= \mathbb{E}[\pi(\theta_1^*|\boldsymbol{y}, \theta_{2:p})|\boldsymbol{y}]
\end{aligned}$$

where the expectation is taken with respect to the marginal distribution of $\theta_{2:p}$ conditional on the data.

The term $\pi(\theta_i^*|\boldsymbol{y}, \theta_{1:i-1}^*)$ can be approximated based on the identity,

$$\begin{aligned}
\pi(\theta_i^*|\boldsymbol{y}, \theta_{1:i-1}^*) &= \int \pi(\theta_i^*, \theta_{i+1:p}|\boldsymbol{y}, \theta_{1:i-1}^*) \,\mathrm{d}\theta_{i+1:p} \\
&= \int \pi(\theta_i^*|\boldsymbol{y}, \theta_{1:i-1}^*, \theta_{i+1:p})\pi(\theta_{i+1:p}|\boldsymbol{y}, \theta_{1:i-1}^*) \,\mathrm{d}\theta_{i+1:p}
\end{aligned} \tag{4.40}$$

and using a Gibbs sampler operating on $\theta_{i:p}$ with $\pi(\theta_{i:p}|\boldsymbol{y}, \theta_{1:i-1}^*)$ as the target distribution and an estimator similar to that in Equation (4.39). This is possible because all the full conditionals required to construct such a Gibbs sampler can be sampled from, otherwise the original Gibbs sampler cannot be constructed.

In addition to the usual requirement of a Gibbs sampler, that all the full

5 SEQUENTIAL MONTE CARLO FOR BAYESIAN COMPUTATION

As reviewed in Chapter 4, MCMC algorithms, though widely used for the purpose of Bayesian computation, have many limitations. Algorithms such as RJMCMC are conceptually appealing, yet often difficult to design in practice. Other algorithms such as the Metropolis-Hastings algorithm and the Gibbs sampling, though provide generic frameworks, within which problems in many fields can be solved, the design of efficient, high performance algorithms still requires considerable expertise and sometimes extensive experience.

In recent years, there is a tendency of considering population based algorithms. A common theme in these algorithms is that, instead of simulating directly from a complex target distribution, related yet simpler distributions are used to "help" the simulation. One such algorithm, which is essentially a generalization of the Metropolis-Hastings algorithm, population MCMC is reviewed in Section 4.3.5, in which the easier to simulate distribution "lends" information to the target and accelerates its mixing. Another population based algorithm, SMC sampler, is the central topic of this chapter

Sequential Monte Carlo (SMC) samplers, in various forms have been around for many years and are widely used in many fields. Until recently there has been little interest in using them for Bayesian model comparison for a few reasons. One of the more important one is that, when MCMC algorithms are available, an SMC sampler could cost more computational resources than a well designed MCMC sampler. However we believe there are at least two important reasons that SMC can be preferable to MCMC for the purpose of Bayesian model comparison in many interesting problems. First, it provides a generic and robust framework for simulation from complex distributions that are difficult for MCMC algorithms, especially for high dimensional multimodal ones. Though it is not impossible to design MCMC algorithms for these problems, it can be hugely difficult in practice. The SMC framework provides an alternative that is easy to use. It has the potential to enable statisticians to construct more realistic, useful models that were previously difficult to use due to the computational complexity. Second, most Monte Carlo algorithms has to be implemented on computers to be useful. Therefore it is realistic to consider the trend of today's computer technologies, in particular, parallel computing. SMC is much more suitable for this kind of computing than conventional MCMC. As we will see later, SMC has certain advantages over some other parallelized algorithms.

In this chapter, we first give a review of SMC algorithms in Section 5.1. It is followed by a section that details the use of SMC in the context of Bayesian model comparison. Next, Section 5.3 develops some extensions and refinements of existing practices. It is followed by a discussion of how the presented framework leads to automatic and generic algorithms. This chapter is concluded with extensive empirical performance study of various proposed strategies.

5.1 SEQUENTIAL MONTE CARLO SAMPLERS

SMC samplers allow us to obtain, iteratively, collections of weighted samples from a sequence of distributions $\{\pi_t\}_{t\ge0}$ over essentially any random variables on some spaces $\{E_t\}_{t\ge0}$. It is an extension to the *sequential importance sampling* (SIS) and resampling algorithms. In the remainder of this section, sequential importance sampling and resampling algorithms are introduced. Then, how they are generalized to SMC samplers for the purpose of the current work is discussed. To simplify the discussion, we will assume that the distributions are continuous and their density functions will also be denoted by $\{\pi_t\}_{t\ge0}$.

5.1.1 *Sequential importance sampling and resampling*

Sequential importance sampling (SIS) generalizes the importance sampling (see Section 4.2) technique for a sequence of distributions $\{\pi_t\}_{t\ge0}$ defined on spaces $\{\prod_{k=0}^{t} E_k\}_{t\ge0}$. The algorithm operates as the following.

At time $t = 0$, draw $\{X_0^{(i)}\}_{i=1}^N$ from η_0 and compute the weights $W_0^{(i)} \propto \pi_0(X_0^{(i)})/\eta_0(X_0^{(i)})$. At time $t \geq 1$, each sample $X_{0:t-1}^{(i)}$, usually termed *particles* in the literature, is extended to $X_{0:t}^{(i)}$ ($X_{0:0}^{(i)} = X_0^{(i)}$) by sampling from a proposal distribution $q_t(\cdot|X_{0:t-1}^{(i)})$. The weights are recalculated as $W_t^{(i)} \propto \pi_t(X_{0:t}^{(i)})/\eta_t(X_{0:t}^{(i)})$ where

$$\eta_t(X_{0:t}^{(i)}) = \eta_{t-1}(X_{0:t-1}^{(i)})q_t(X_{0:t}^{(i)}|X_{0:t-1}^{(i)}) \tag{5.1}$$

and thus

$$\begin{aligned} W_t^{(i)} \propto \frac{\pi_t(X_{0:t}^{(i)})}{\eta_t(X_{0:t}^{(i)})} &= \frac{\pi_t(X_{0:t}^{(i)})\pi_{t-1}(X_{0:t-1}^{(i)})}{\eta_{t-1}(X_{0:t-1}^{(i)})q_t(X_{0:t}^{(i)}|X_{0:t-1}^{(i)})\pi_{t-1}(X_{0:t-1}^{(i)})} \\ &= \frac{\pi_t(X_{0:t}^{(i)})}{q_t(X_{0:t}^{(i)}|X_{0:t-1}^{(i)})\pi_{t-1}(X_{0:t-1}^{(i)})} W_{t-1}^{(i)}. \end{aligned} \tag{5.2}$$

The importance sampling approximation of $\mathbb{E}_{\pi_t}[\varphi_t(X_{0:t})]$ can be obtained using $\{W_t^{(i)}, X_{0:t}^{(i)}\}_{i=1}^N$, where φ_t is some function of interest.

However, this approach fails as t becomes large. The weights tend to become concentrated on a few particles as the discrepancy between η_t and π_t becomes larger. Resampling techniques are applied such that, a new particle system $\{\bar{W}_t^{(i)}, \bar{X}_{0:t}^{(i)}\}_{i=1}^M$ is obtained with the property,

$$\mathbb{E}\Big[\sum_{i=1}^M \bar{W}_t^{(i)}\varphi_t(\bar{X}_{0:t}^{(i)})\Big] = \mathbb{E}\Big[\sum_{i=1}^N W_t^{(i)}\varphi_t(X_{0:t}^{(i)})\Big] \tag{5.3}$$

where φ_t is the function of interest and both $\{\bar{W}_t^{(i)}\}_{i=1}^N$ and $\{W_t^{(i)}\}_{i=1}$ are normalized weights, that is they are scaled such that they sum up to one. In other words, the resampling step does not change the expectation of the estimate. In practice, the resampling algorithm is usually chosen such that $M = N$ and $\bar{W}^{(i)} = 1/N$ for $i = 1, \dots, N$. Resampling can be performed at each iteration t or adaptively based on some criteria of the discrepancy between the distribution of the particles and the target distribution π_t, accumulated since the last time resampling was performed. One popular quantity used to monitor this discrepancy is *effective sample size* (ESS), introduced by [108], defined as

$$\text{ESS}_t = \frac{1}{\sum_{i=1}^N (W_t^{(i)})^2} \tag{5.4}$$

where $\{W_t^{(i)}\}_{i=1}^N$ are the normalized weights. Resampling can be performed when ESS $\leq \alpha N$ where $\alpha \in [0, 1]$.

The common practice of resampling is to replicate particles with large weights and discard those with small weights. In other words, instead of generating a random sample $\{\bar{X}_{0:t}^{(i)}\}_{i=1}^N$ directly, a random sample of integers $\{R_t^{(i)}\}_{i=1}^N$ is generated, such that $R_t^{(i)} \geq 0$ for $i = 1, \dots, N$ and $\sum_{i=1}^N R_t^{(i)} = N$. Each particle value $X_{0:t}^{(i)}$ is then replicated $R_t^{(i)}$ times in the new particle system. The distribution of $\{R_t^{(i)}\}_{i=1}^N$ should fulfill the requirement of Equation (5.3). One such distribution is a multinomial distribution of size N and weights $(W_t^{(i)}, \dots, W_t^{(N)})$ and the resulting algorithm is called the *multinomial resampling*. See [40] for some widely used resampling algorithms. Here we briefly review some of the most commonly used algorithms besides multinomial resampling.

Residual resampling This was introduced in [108]. In this approach, for $i = 1, \dots, N$, we have

$$R_t^{(i)} = \lfloor NW_t^{(i)} \rfloor + \bar{R}_t^{(i)} \tag{5.5}$$

where $\lfloor\rfloor$ denotes the integer part and $\{\bar{R}_t^{(i)}\}_{i=1}^N$ is distributed according to a multinomial distribution with size $N - \bar{N}_t$ and weights $(\bar{W}_t^{(i)}, \dots, \bar{W}_t^{(N)})$ with

$$\bar{N}_t = \sum_{i=1}^N \lfloor NW_t^{(i)} \rfloor \qquad \text{and} \qquad \bar{W}_t^{(i)} = \frac{NW_t^{(i)} - \lfloor NW_t^{(i)} \rfloor}{N - \bar{N}_t}$$

It was shown that residual resampling can lead to significant variance reduction for the importance sampling estimator when compared to the multinomial resampling [40]. It is easy to see that, unlike multinomial resampling, in residual resampling, the replication number $R_t^{(i)}$ will be no less than $NW_t^{(i)} - 1$. The next two resampling algorithms also share this property.

Stratified resampling This can be seen in [99]. Let Q denote the generalized inverse function of the cumulative distribution function of a multinomial distribution with size N and weights $(W_1, \dots, W_N)$. That is $Q(x) = i$ for $x \in (\sum_{j=1}^{i-1} W_j, \sum_{j=1}^{i} W_j]$. The stratified resampling proceeds by first drawing uniform random variates $U_t^{(i)}$ on $((i-1)/N, i/N]$ for $i = 1, \dots, N$, and then setting $I_t^{(i)} = Q(U_t^{(i)})$. The new particle system is formed by $\{X_{0:t}^{(I_t^{(i)})}\}_{i=1}^N$. This algorithm also results in smaller variance of the importance sampling estimator than that of the multinomial resampling [40].

Systematic resampling This was mentioned in [163]. Similar to the stratified resampling, the systematic resampling also uses the inversion method. However, instead of generating N uniform random variates, it only generates one U_t from $(0, 1/N]$ and deterministically set $U_t^{(i)} = (i-1)/N + U_t$ for $i = 1, \dots, N$. Though it has the most straightforward implementation among all the algorithms introduced so far, it is more complicated to study the behavior of the conditional variance of the generated samples. As shown in [40], there exists counter-examples that the systematic resampling does not outperform the multinomial resampling.

Combination of residual and stratified/systematic resampling Both the stratified and systematic resampling can be used together with the residual resampling. It operates by first computing the integer part and the residual of $NW_t^{(i)}$ for $i = 1, \dots, N$, and then stratified or systematic resampling is performed using the residuals as weights. It has the advantage that the resulting algorithm provides better performance than each of the algorithms involved [40].

There are other specialized resampling algorithms. The algorithms shown above have a common drawback. They require the knowledge of all the weights being available before the algorithm can proceed. Therefore, in some situations the performance of SMC algorithms can be limited by the fact that the resampling step cannot be parallelized. Parallelized resampling is an area being actively researched (e.g., [94, 118]). However, we will not discuss such specialized algorithms in this thesis.

5.1.2 *SMC samplers*

SMC samplers generalize the SIS algorithm for a sequence of distributions $\{\pi_t\}_{t\geq 0}$ over essentially any random variables on some spaces $\{E_t\}_{t\geq 0}$, by constructing a sequence of auxiliary distributions $\{\tilde{\pi}_t\}_{t\geq 0}$ on spaces of increasing dimensions,

$$\tilde{\pi}_t(x_{0:t}) = \pi_t(x_t)\prod_{s=0}^{t-1} L_s(x_{s+1}, x_s), \tag{5.6}$$

where the sequence of Markov kernels $\{L_s\}_{s=0}^{t-1}$, termed *backward kernels*, is formally arbitrary but critically influences the estimator variance. See [39] for further details and guidance on the selection of these kernels (also see Section 5.1.5).

Standard SIS and resampling algorithms can then be applied to the sequence of synthetic distributions, $\{\tilde{\pi}_t\}_{t\geq 0}$. The calculation of the importance weights is straightforward. At time $t-1$, assume that a set of weighted particles approximating $\tilde{\pi}_{t-1}$ is available, $\{W_{t-1}^{(i)}, X_{0:t-1}^{(i)}\}_{i=1}^{N}$, then at time t, the path of each particle is extended with a Markov kernel say, $K_t(x_{t-1}, x_t)$ and the set of particles $\{X_{0:t}^{(i)}\}_{i=1}^{N}$ reach the distribution $\eta_t(x_{0:t}^{(i)}) = \eta_0(x_0^{(i)})\prod_{k=1}^{t} K_t(x_{t-1}^{(i)}, x_t^{(i)})$ (assuming no resampling has occurred), where η_0 is the initial distribution of the particles. To correct the discrepancy between η_t and $\tilde{\pi}_t$, Equation (5.2) is applied to calculate the new weights,

$$W_t^{(i)} \propto \frac{\tilde{\pi}_t(X_{0:t}^{(i)})}{\eta_t(X_{0:t}^{(i)})} = \frac{\pi_t(X_t^{(i)})\prod_{s=0}^{t-1} L_s(X_{s+1}^{(i)}, X_s^{(i)})}{\eta_0(X_0^{(i)})\prod_{k=1}^{t} K_k(X_{k-1}^{(i)}, X_k^{(i)})} \propto \tilde{w}_t(X_{t-1}^{(i)}, X_t^{(i)})W_{t-1}^{(i)} \tag{5.7}$$

where $\tilde{w}_t$, termed the *incremental weights*, are calculated as,

$$\tilde{w}_t(X_{t-1}^{(i)}, X_t^{(i)}) = \frac{\pi_t(X_t^{(i)})L_{t-1}(X_t^{(i)}, X_{t-1}^{(i)})}{\pi_{t-1}(X_{t-1}^{(i)})K_t(X_{t-1}^{(i)}, X_t^{(i)})}. \tag{5.8}$$

If π_t is only known up to a normalizing constant, say $\pi_t(x_t) = \gamma_t(x_t)/Z_t$, then we can use the *unnormalized* incremental weights

$$w_t(X_{t-1}^{(i)}, X_t^{(i)}) = \frac{\gamma_t(X_t^{(i)})L_{t-1}(X_t^{(i)}, X_{t-1}^{(i)})}{\gamma_{t-1}(X_{t-1}^{(i)})K_t(X_{t-1}^{(i)}, X_t^{(i)})} \tag{5.9}$$

for importance sampling estimation. Further, with the normalized weights of the last generation of the particle system, $\{W_{t-1}^{(i)}\}_{i=1}^{N}$, we can estimate the ratio of normalizing constant Z_t/Z_{t-1} by

$$\frac{\hat{Z}_t}{Z_{t-1}} = \sum_{i=1}^{N} W_{t-1}^{(i)} w_t(X_{t-1}^{(i)}, X_t^{(i)}). \tag{5.10}$$

Iteratively, the ratio of the normalizing constants between the initial distribution π_0 and some target π_T, $T \geq 1$ can be estimated. The incremental weights clearly depend on the choice of the backward kernels. See [39] and Section 5.1.5 for details on calculating the incremental weights.

5.1.3 *Sequence of distributions*

There are many ways to specify the sequence of distributions. For many applications, such a sequence arises from the problem setting naturally.

In [30] a data tempering scheme was considered in the context of Bayesian inference for static parameters. Suppose data $\boldsymbol{y} = (y_1, \dots, y_n)$ are available and it is of interest to inference the posterior distribution of some parameter vector θ, $\pi(\theta|\boldsymbol{y})$. Then one can construct the following sequence of distributions $\{\pi_t\}_{t=1}^{n}$,

$$\pi_t(\theta) = \pi(\theta|y_1, \dots, y_t). \tag{5.11}$$

That is, the data is introduced one by one into the posterior. However, this scheme can be sensitive to the order of data being introduced. A modification is to introduce a batch of data at each iteration, also introduced in [30]. The number of data points to be incorporated in each iteration can still be difficult to determine. The more data points introduced at each step, the more degeneracy (measured by, e.g., ESS) will be induced. It is natural to consider introducing data such that a constant level of degeneracy is maintained. It is intuitive to see that with enough data (large t) already introduced, the addition of the same amount of data will have less influence on the posterior than when there have only been a few data points (small t). It was shown in [30] that, to maintain a constant level of degeneracy, it can be expected that the number of data points at each step increases geometrically.

Another generic scheme is called the *geometric path*. Given the target distribution π and another distribution η, which usually has the same support but heavier tails than that of π, a sequence of distributions $\{\pi_t\}_{t=0}^T$ can be constructed,

$$\pi_t(x) = \pi(x)^{\alpha(t/T)}\eta(x)^{1-\alpha(t/T)} \tag{5.12}$$

where $\alpha : [0,1] \to [0,1]$ is a monotonically increasing mapping with $\alpha(0) = 0$ and $\alpha(1) = 1$. Some variants of this scheme adapted particularly for the purpose of Bayesian modeling can be seen in Section 5.2.1 and 5.2.2. The sequence of distributions moves smoothly from η, which is usually easy to sample from or to construct an efficient proposal distribution for, towards the target distribution π. However, this scheme has one important drawback. For a high dimensional target with many well separated modes, it can be difficult for η (or its proposal distribution) to produce samples within each of all the modes and the sampler may never reach part of the support of the target distribution π. This problem can be partially solved by increase the number of particles.

Despite this limitation, the geometric scheme has a significant advantage as we will see later (Section 5.1.5 and 5.3.2). In short, when combined with certain transition kernels and backward kernels, it allows easy computation of the weights using quantities already computed in the last iteration without actually simulating the samples of the current iteration. Therefore it provides a way to conduct adaptive sampling with low computational cost.

There are other sequences, which often have particular use for certain applications. For example, for global optimization of a function f, such that $\int f(x)\,\mathrm{d}x < \infty$ (that is, it can be normalized into a density function), one can simulate from a sequence of distributions, $\{\pi_t\}_{t\ge0}$, defined by,

$$\pi_t(x) \propto f(x)^{\alpha(t)} \tag{5.13}$$

where $\alpha : [0,\infty) \to [0,\infty)$ is a monotonically increasing mapping with $\alpha(t) \to \infty$ as $t \to \infty$. The sequence of distributions will concentrate more and more around the modes of f (see also [111]).

5.1.4 *Sequence of transition kernels*

It is easy to see that, the optimal proposal kernel is $K_t(x_{t-1}, x_t) = \pi_t(x_t)$, in the sense of minimizing the Monte Carlo variance of the importance weights. However, this choice is not possible except for trivial cases. Some sensible alternatives have been proposed in the past.

One approach is to use independent proposals, $K_t(x_{t-1}, x_t) = \mu_t(x_t)$ for some distribution μ_t at each iteration. Usually, μ_t belongs to a family of distributions with parameters determined by certain statistics of the particle system of the last generation, for example, a multivariate Normal distribution with the mean vector and the covariance matrix estimated from current samples. For general use, this can be overly restrictive and the performance can be difficult to calibrate, especially in high dimensional problems. In this situation, it is difficult for the independent proposal to capture the characteristics of the target distribution without knowing it in advance. And thus it can lead to large variance of importance weights and poor performance of the estimator.

An important alternative, advocated in [39] is to use MCMC kernels targeting π_t. This strategy is particularly justified if the sequence of distributions moves smoothly or the kernel is fast mixing. When the sequence of distributions moves slowly from one to another, that is π_t is not very different from π_{t-1}, and thus samples from η_{t-1} is a good approximation to π_t, the kernel is likely to successfully move particles towards high probability regions of π_t. What makes it more attractive is the fact that we can use the vast literature on the design of efficient MCMC algorithms to build the proposal distributions. In addition, as we will see very soon, when combined with certain backward kernels, this approach enables us to calculate the importance weights without actually simulating samples. And therefore it leads to low computational cost adaptive algorithms that can improve the performance considerably.

5.1.5 *Optimal and suboptimal backward kernels*

The sequence of backward kernels $\{L_t\}_{t=0}^{T-1}$ should be optimized with respect to the sequence of transition kernels $\{K_t\}_{t=1}^{T}$. Let $\eta_t(x_t)$ denote the marginal distribution of X_t. That is,

$$\eta_t(x_t) = \eta_0(x_0)\prod_{k=1}^{t} K_k(x_{k-1}, x_k) \tag{5.14}$$

if no resampling has occurred and,

$$\eta_t(x_t) = \pi_l(x_l)\prod_{k=l+1}^{t} K_k(x_{k-1}, x_k) \tag{5.15}$$

if the last resampling occurs at time l.

As shown in Proposition 1 in [39], the backward kernel $L_{t-1}(x_t, x_{t-1})$ that minimizes the variance of unnormalized importance weights is given by,

$$L_{t-1}^{\text{OPT}}(x_t, x_{t-1}) = \frac{\eta_{t-1}(x_{t-1})K_t(x_{t-1}, x_t)}{\eta_t(x_t)} \tag{5.16}$$

and in this case the weights are,

$$W_t^{(i)} \propto \frac{\pi_t(X_t^{(i)})}{\eta_t(X_t^{(i)})}. \tag{5.17}$$

The marginal $\eta_t(x_t)$ is typically not available and thus the above optimal backward kernel cannot be used in practice.

One sensible alternative is to substitute π_{t-1} for η_{t-1}, that is,

$$L_{t-1}(x_t, x_{t-1}) = \frac{\pi_{t-1}(x_{t-1})K_t(x_{t-1}, x_t)}{\int \pi_{t-1}(x_{t-1})K_t(x_{t-1}, x_t)\,\mathrm{d}x_{t-1}}. \tag{5.18}$$

This approach is justified if the particle system has has been resampled at time $t-1$, in which case η_{t-1} is indeed equal to π_{t-1} or when resampling was at least performed occasionally such that the degeneracy between η_{t-1} and π_{t-1} is controlled. The incremental weights can be computed if the integration above can be computed. Usually this is done through the unnormalized distribution γ_{t-1} instead of π_{t-1}. When γ_{t-1} is known analytically, the unnormalized incremental weights are,

$$w_t(X_{t-1}^{(i)}, X_t^{(i)}) = \frac{\gamma_t(X_t^{(i)})}{\int \gamma_{t-1}(x_{t-1})K_t(x_{t-1}, X_t^{(i)})\,\mathrm{d}x_{t-1}}. \tag{5.19}$$

The requirement of the knowledge of the above integration can limit the use of the kernel in some applications.

When using an MCMC kernel K_t that is invariant to π_t as the transition kernel, and when $\pi_{t-1} \approx \pi_t$, by substitute π_t for π_{t-1}, Equation (5.18) becomes,

$$
\begin{aligned}
L_{t-1}(x_t, x_{t-1}) &= \frac{\pi_t(x_{t-1}) K_t(x_{t-1}, x_t)}{\int \pi_t(x_{t-1}) K_t(x_{t-1}, x_t) \, \mathrm{d} x_{t-1}} \\
&= \frac{\pi_t(x_{t-1}) K_t(x_{t-1}, x_t)}{\pi_t(x_t)}
\end{aligned} \tag{5.20}
$$

where the second equation is due to the fact that K_t is invariant to π_t. It is easy to see that the unnormalized incremental weights are,

$$
w_t(X_{t-1}^{(i)}, X_t^{(i)}) = \frac{\gamma_t(X_{t-1}^{(i)})}{\gamma_{t-1}(X_{t-1}^{(i)})}. \tag{5.21}
$$

Note that, the incremental weights no longer depend on the samples from iteration t, $\{X_t^{(i)}\}_{i=1}^N$. Therefore, it can be calculated before the sampling step, which moves the particles according to the kernel K_t. Since the incremental weights solely depends on the specification of γ_t, which usually can be computed point-wise, given the current samples, it is possible to specify γ_t (and therefore π_t) according to the calculated weights using information from the current samples before carrying out the actual simulation of the current iteration.

However the expression (5.21) is not without drawbacks. Compared to the expression (5.19), which is more intuitive since it considers the transition kernel K_t, which depends on the current samples, it benefits less from fast mixing kernels. If π_t is not close to π_{t-1}, then the variance of the incremental weights is likely to be large even when the kernel K_t mixes fast. Indeed, later we will show empirically that, it is preferable to use more distributions rather than using multiple passes of MCMC moves in a single iteration, provided that they use the same computational resources.

5.2 APPLICATION TO BAYESIAN MODEL COMPARISON

The application of SMC samplers to Bayesian model comparison is straightforward. However, it has been overlooked in recent years. In this section, we outline common strategies of using SMC samplers for Bayesian model comparison. In the next section, we introduce some innovative extensions and refinements to existing practices.

As reviewed in Section 3.2.1, the problem of interest is characterizing the posterior distribution over $\{\mathcal{M}_k\}_{k\in\mathcal{K}}$, a set of possible models, with model $\mathcal{M}_k$ having parameter vector $\theta_k \in \Theta_k$ which also usually need to be inferred. Given prior distributions $\pi(\mathcal{M}_k)$ and $\pi(\theta_k|\mathcal{M}_k)$ and the likelihood function $p(\boldsymbol{y}|\theta_k, \mathcal{M}_k)$, we seek the posterior distribution $\pi(\mathcal{M}_k|\boldsymbol{y}) \propto p(\boldsymbol{y}|\mathcal{M}_k)\pi(\mathcal{M}_k)$. There are three fundamentally different approaches to the computations,

1. Calculate posterior model probability distribution $\pi(\mathcal{M}_k|\boldsymbol{y})$ directly.
2. Calculate the evidence, the marginal likelihood $p(\boldsymbol{y}|\mathcal{M}_k)$, of each model.
3. Calculate pairwise evidence ratios, the Bayes factor $B_{k_1k_2}$ for two models $\mathcal{M}_{k_1}$ and $\mathcal{M}_{k_2}$ directly.

Each approach admits a natural SMC strategy.

5.2.1 *SMC1: An all-in-one approach*

One could consider obtaining samples from the same distribution employed in the RJMCMC (see Section 4.3.4) approach to model comparison, namely,

$$\pi^{(1)}(\mathcal{M}_k, \theta_k) \propto \pi(\mathcal{M}_k)\pi(\theta_k|\mathcal{M}_k)p(\boldsymbol{y}|\theta_k, \mathcal{M}_k) \qquad (5.22)$$

which is defined on the disjoint union space $\bigcup_{k\in\mathcal{K}}(\{\mathcal{M}_k\} \times \Theta_k)$.

One obvious SMC approach is to define a sequence of distributions $\{\pi_t^{(1)}\}_{t=0}^T$ such that $\pi_0^{(1)}$ is easy to sample from, $\pi_T^{(1)} = \pi^{(1)}$ and the intermediate distributions move smoothly between them. In the remainder of this section, we use the notation $(\mathcal{M}_t, \theta_t)$ to denote a random sample on the space $\bigcup_{k\in\mathcal{K}}(\{\mathcal{M}_k\} \times \Theta_k)$ at time t. One simple approach, which might be expected to work well, is the use of an annealing

```
*Initialisation:* Set $t \leftarrow 0$.
    Sample $X_0^{(i)} = (\theta_0^{(i)}, \mathcal{M}_0^{(i)}) \sim \nu$ for some proposal distribution $\nu$ (usually the joint prior).
    Weight $W_0^{(i)} \propto w_0(X_0^{(i)}) = \pi(\mathcal{M}_0^{(i)})\pi(\theta_0^{(i)}|\mathcal{M}_0^{(i)})/\nu(\theta_0^{(i)}, \mathcal{M}_0^{(i)})$.
    Apply resampling if necessary (e.g., if ESS less than some threshold; see Section 5.1.1).
*Iteration:* Set $t \leftarrow t + 1$.
    Weight $W_t^{(i)} \propto W_{t-1}^{(i)} p(\boldsymbol{y}|\theta_{t-1}^{(i)}, \mathcal{M}_{t-1}^{(i)})^{\alpha(t/T)-\alpha([t-1]/T)}$.
    Apply resampling if necessary.
    Sample $X_t^{(i)} \sim K_t(\cdot|X_{t-1}^{(i)})$, a $\pi_t^{(1)}$-invariant kernel.
*Repeat* the *Iteration* step *until* $t = T$.
```

Algorithm 5.1 SMC1: An All-in-One Approach to Model Comparison.

scheme such that,

$$\pi_t^{(1)}(\mathcal{M}_t, \theta_t) \propto \pi(\mathcal{M}_t)\pi(\theta_t|\mathcal{M}_t)p(\boldsymbol{y}|\theta_t, \mathcal{M}_t)^{\alpha(t/T)}, \tag{5.23}$$

for some monotonically increasing $\alpha : [0, 1] \to [0, 1]$ with $\alpha(0) = 0$ and $\alpha(1) = 1$. Other approaches are possible and might prove more efficient for some problems (such as the "data tempering" approach (see Section 5.1.3), which [30] proposed for parameter estimation which could easily be incorporated in our framework), but this strategy provides a convenient generic approach. These choices lead to Algorithm 5.1.

This approach might outperform RJMCMC when it is difficult to design fast-mixing Markov kernels. There are many examples of such an annealed SMC strategy outperforming MCMC at a given computational cost – see, for example, [43, 92, 44]. Such trans-dimensional SMC has been proposed in several contexts such as [126], and an extension was proposed and analyzed by [85].

We include this approach for completeness and study it empirically later. However, the more direct approaches described in the following sections lead more

```
For each model $k \in \mathcal{K}$ perform the following algorithm.
*Initialisation:* Set $t \leftarrow 0$.
    Sample $\theta_0^{(k,i)} \sim \nu_k$ for some proposal distribution $\nu_k$ (usually the parameter prior).
    Weight $W_0^{(k,i)} \propto w_0(\theta_0^{(k,i)}) = \pi(\theta_0^{(k,i)}|\mathcal{M}_k)/\nu_k(\theta_0^{(k,i)})$.
    Apply resampling if necessary.
*Iteration:* Set $t \leftarrow t + 1$.
    Weight $W_t^{(k,i)} \propto W_{t-1}^{(k,i)} p(\boldsymbol{y}|\theta_{t-1}^{(k,i)}, \mathcal{M}_k)^{\alpha_k(t/T_k)-\alpha_k([t-1]/T_k)}$.
    Apply resampling if necessary.
    Sample $\theta_t^{(k,i)} \sim K_t(\cdot|\theta_{t-1}^{(k,i)})$, a $\pi_t^{(k,2)}$-invariant kernel.
*Repeat* the *Iteration* step *until* $t = T_k$.
```

Algorithm 5.2 SMC2: A Direct-Evidence-Calculation Approach.

naturally to easy-to-implement strategies with good performance.

5.2.2 *SMC2: A direct-evidence-calculation approach*

An alternative approach would be to estimate explicitly the evidence associated with each model. We propose to do this by sampling from a sequence of distributions for each model, starting from the parameter prior and sweeping through a sequence of distributions to the posterior.

Numerous strategies are possible to construct such a sequence of distributions, but one option is to use for each model $\mathcal{M}_k \in \mathcal{M}$, the sequence $\{\pi_t^{(2,k)}\}_{t=0}^{T_k}$, defined by,

$$\pi_t^{(2,k)}(\theta_t) \propto \pi(\theta_t|\mathcal{M}_k)p(\boldsymbol{y}|\theta_t, \mathcal{M}_k)^{\alpha_k(t/T_k)}. \tag{5.24}$$

where the number of distributions, T_k, and the annealing schedule, $\alpha_k : [0, 1] \rightarrow [0, 1]$, may be different for each model. This leads to Algorithm 5.2.

The estimator of the posterior model probabilities depends upon the approach taken to estimate the normalizing constant. Direct estimation of the evidence can be performed using the output of this SMC algorithm and the standard estimator, termed SMC2-DS (see also Equation (5.10)), given by,

$$\sum_{i=1}^{N} \frac{\pi(\theta_0^{(k,i)}|\mathcal{M}_k)}{\nu_k(\theta_0^{(k,i)})} \times \prod_{t=2}^{T} \sum_{i=1}^{N} W_{t-1}^{(k,i)} p(y|\theta_{t-1}^{(k,i)}, \mathcal{M}_k)^{\alpha_k(t/T_k)-\alpha_k([t-1]/T_k)} \tag{5.25}$$

where $W_{t-1}^{(k,i)}$ is the normalized importance weight of the i^{th} particle during iteration $t-1$ for model $\mathcal{M}_k$. An alternative approach to computing the evidence is also worthy of consideration. As has been suggested, and shown to perform well empirically previously [91], it is possible to use all of the samples from every generation of an SMC sampler to approximate the path sampling estimator and hence to obtain an estimate of the ratio of normalizing constants. Section 5.2.4 provides details for the use of path sampling for both this and other SMC algorithms discussed later.

This approach is appealing for several reasons. One is that it is designed to estimate directly the quantity of interest, the evidence, producing samples from that distribution at the same time. Another advantage of this approach over SMC1 and the RJMCMC is that it provides as good a characterization of each model as is required: It is possible to obtain a good estimate of the parameters of every model, even those for which the posterior probability is small. Perhaps most significant is the fact that this approach does not require the design of proposal distributions or Markov kernels which move from one model to another: Each model is dealt with in isolation. Whilst this may not be desirable in every situation, there are circumstances in which efficient moves between models are almost impossible to devise.

This approach also has some disadvantages. In particular, it is necessary to run a separate simulation for each model – rendering it impossible to deal with countable collections of models (although this is not much of a substantial problem in many interesting cases). The ease of implementation may often offset this limitation.

5.2.3 *SMC3: A relative-evidence-calculation approach*

A final approach can be thought of as *sequential model comparison*. Rather than estimating the evidence associated with any particular model, we could estimate pairwise evidence ratios directly. The SMC sampler starts with an initial distribution being the posterior of one model (which could comes from a separate SMC sampler starting from its prior) and moves towards the posterior of another related model. Then the sampler can continue towards another related model.

Given a finite collection of models $\{\mathcal{M}_k\}_{k\in\mathcal{K}}$, suppose the models are ordered in a sensible way (e.g., $\mathcal{M}_{k-1}$ is nested within $\mathcal{M}_k$ or θ_k is of higher dimension than θ_{k-1}). For each $k \in \mathcal{K}$, we consider a sequence of distributions $\{\pi_t^{(3,k)}\}_{t=0}^{T_k}$, such that,

$$\begin{aligned}
\pi_0^{(3,k)}(\mathcal{M},\theta) &= \pi(\theta|\boldsymbol{y},\mathcal{M}_k)\mathbb{I}_{\{\mathcal{M}_k\}}(\mathcal{M}) \\
\pi_{T_k}^{(3,k)}(\mathcal{M},\theta) &= \pi(\theta|\boldsymbol{y},\mathcal{M}_{k+1})\mathbb{I}_{\{\mathcal{M}_{k+1}\}}(\mathcal{M}) = \pi_0^{(3,k+1)}(\mathcal{M},\theta).
\end{aligned}$$

where $(\mathcal{M},\theta)$ denote a random variable on the disjoint union space $(\{\mathcal{M}_k\}\times\Theta_k)\cup(\{\mathcal{M}_{k+1}\}\times\Theta_{k+1})$. When it is possible to construct an SMC sampler that iterates over this sequence of distributions, the estimate of the ratio of normalizing constants is the Bayes factor estimate of model $\mathcal{M}_{k+1}$ in favor of model $\mathcal{M}_k$.

This approach is conceptually appealing, but requires the construction of a smooth path between the posterior distributions of interest. The geometric annealing strategy which has been advocated as a good generic strategy in the previous sections is only appropriate when the support of successive distributions is non-increasing. This is unlikely to be the case in interesting model comparison problems.

Here we consider a sequence of distributions on the disjoint union space $(\{\mathcal{M}_k\}\times\Theta_k\})\cup(\{\mathcal{M}_{k+1}\}\times\Theta_{k+1}\})$, with the sequence of distributions $\{\pi_t^{(3,k)}\}_{t=0}^{T_k}$ defined proportional to that of the full posterior (see Section 3.2.1) restricted to the space of these two models,

$$\pi_t^{(3,k)}(\mathcal{M}_t,\theta_t) \propto \pi_t(\mathcal{M}_t)\pi(\theta_t|\mathcal{M}_t)p(\boldsymbol{y}|\theta_t,\mathcal{M}_t) \tag{5.26}$$

where $\mathcal{M}_t \in \{\mathcal{M}_k, \mathcal{M}_{k+1}\}$ and the prior of models at time t, $\pi_t(\mathcal{M}_t)$ is defined by

$$\pi_t(\mathcal{M}_{k+1}) = \alpha_k(t/T_k) \tag{5.27}$$

for some monotonically increasing $\alpha_k : [0,1] \to [0,1]$ such that $\alpha_k(0) = 0$ and $\alpha_k(1) = 1$. It is clear that the MCMC moves between iterations need to be similar to those in the RJMCMC or SMC1 algorithms. The difference is that instead of efficient exploration of the whole model space, only moves between two models are required and the sequence of distributions employed helps to ensure exploration of both model spaces. The algorithm for this particular sequence of distributions is outlined in Algorithm 5.3. It can be extended to other possible sequence of distributions between models.

An advantage of this approach is that it provides direct estimate of the Bayes factor which is of interest for model comparison purpose while not requiring exploration of as complicated a space as that employed within RJMCMC or SMC1. The estimator of normalizing constants in SMC3 follows in exactly the same manner as in the SMC2 case. In SMC3, the same estimator provides a direct estimate of the Bayes factor.

5.2.4 *Path sampling via SMC2/SMC3*

The estimation of the normalizing constants associated with our sequences of distributions can be achieved by a Monte Carlo approximation to the *path sampling* formulation given by [56]. This is similar to the technique for population MCMC as described in Section 4.3.7. In the context of SMC, this approach is also very closely related to the use of annealed importance sampling (AIS) for the same purpose [120] but as will be demonstrated below the incorporation of some other elements of more general SMC algorithms can improve performance at negligible cost. Recall that, given a parameter α which defines a family of distributions, $\{\pi_\alpha = \gamma_\alpha/Z_\alpha\}_{\alpha\in[0,1]}$ which move smoothly from $\pi_0 = \gamma_0/Z_0$ to $\pi_1 = \gamma_1/Z_1$ as α increases from zero to one, one can estimate the logarithm of the ratio of their normalizing constants via a

```
Initialisation: Set $k \leftarrow 1$.
    Use Algorithm 5.2 to obtain weighted samples for $\pi_{T_1}^{(3,1)}$, the parameter posterior for model $\mathcal{M}_1$
Relative Evidence Calculation
    Set $k \leftarrow k+1, t \leftarrow 0$.
    Let $\{W_0^{(k,i)}, X_0^{(k,i)}\}_{i=1}^N$ where $X_0^{(k,i)} = (\theta_0^{(k,i)}, \mathcal{M}_0^{(k,i)})$ denote the current samples.
    Apply resampling if necessary.
    Iteration: Set $t \leftarrow t+1$.
        Weight $W_t^{(k,i)} \propto W_{t-1}^{(k,i)} \pi_t(\mathcal{M}_{t-1}^{(k,i)}) / \pi_{t-1}(\mathcal{M}_{t-1}^{(k,i)})$.
        Apply resampling if necessary.
        Sample $(\theta_t^{(k,i)}, \mathcal{M}_t^{(k,i)}) \sim K_t(\cdot|\theta_{t-1}^{(k,i)}, \mathcal{M}_{t-1}^{(k,i)})$, a $\pi_t^{(3,k)}$-invariant kernel.
    Repeat the Iteration step up to $t = T_k$.
Repeat the Relative Evidence Calculation step until sequentially all relative evidences are calculated.
```

Algorithm 5.3 SMC3: A Relative-Evidence-Calculation Approach to Model Comparison.

simple integral relationship which holds under very mild regularity conditions,

$$\log\left(\frac{Z_1}{Z_0}\right) = \int_0^1 \mathbb{E}_{\pi_\alpha}\left[\frac{\mathrm{d}\log\gamma_\alpha(X)}{\mathrm{d}\alpha}\right] \mathrm{d}\alpha,$$

where the inner expectation is taken with respect to π_α. Note that the sequence of distributions in the SMC2 and SMC3 algorithms above, can both be interpreted as belonging to such a family of distributions, with $\alpha = \alpha_k(t/T_k)$, where the mapping $\alpha_k : [0,1] \to [0,1]$ is again monotonically increasing with $\alpha_k(0) = 0$ and $\alpha_k(1) = 1$.

The SMC sampler provides us with a set of weighted samples obtained from a sequence of distributions suitable for approximating this integral. At each iteration t we can obtain an estimate of the expectation within the integral for $\alpha = \alpha_k(t/T)$ via the usual importance sampling estimator, and this integral can then be approximated via numerical integration. Whenever the sequence of distributions employed by SMC3 has appropriate differentiability it is also possible to employ path sampling to estimate, directly, the Bayes factor via this approach. In general, given an increasing sequence $\{\alpha_t\}_{t=0}^T$ where $\alpha_0 = 0$ and $\alpha_T = 1$, a family of distributions $\{\pi_\alpha\}_{\alpha\in[0,1]}$ as before, and an SMC sampler that iterates over the sequence of distribution $\{\pi_t = \pi_{\alpha_t} = \gamma_{\alpha_t}/Z_{\alpha_t}\}_{t=0}^T$, then with the weighted samples $\{W_t^{(i)}, X_t^{(i)}\}_{i=1}^N$, and $t = 0,\dots,T$, a path sampling estimator of the ratio of normalizing constants $\Xi_T = \log(Z_1/Z_0)$ can be approximated (using an elementary Trapezoidal scheme) by,

$$\hat{\Xi}_T^N = \sum_{t=1}^T \frac{1}{2}(\alpha_t - \alpha_{t-1})(U_t^N + U_{t-1}^N), \tag{5.28}$$

where

$$U_t^N = \sum_{i=1}^N W_t^{(i)} \frac{\mathrm{d}\log\gamma_\alpha(X_t^{(i)})}{\mathrm{d}\alpha}\Big|_{\alpha=\alpha_t}. \tag{5.29}$$

We term these estimators SMC2-PS and SMC3-PS in the followings. The combination of SMC and path sampling is somewhat natural and has been proposed before, e.g., [91] although not there in a Bayesian context. Despite the good performance observed in the setting of rare event simulation, the estimation of normalizing constants by this approach seems to have received little attention in the literature. We suspect that this is because of widespread acceptance of the suggestion of [39], that SMC doesn't outperform AIS when normalizing constants are the object of inference or that of [24] that all simulation-based estimators based around path sampling can be expected to behave similarly. We will demonstrate below that these observations, whilst true in certain contexts, do not hold in full generality.

5.3 EXTENSIONS AND REFINEMENTS

The algorithms introduced in the last section can be seen as straightforward application of the well established SMC algorithms to Bayesian model comparison. By construction, SMC algorithms can be more robust than many MCMC and other algorithms. However, as with any Monte Carlo algorithms, without careful design, the performance can be far from satisfactory for realistic applications. In this section, we introduce some extensions and refinement that can further improve the presented framework. Of course, they cannot guarantee that the algorithms will perform well for all possible situations. However, they provide robust and reliable solutions for many realistic applications with minimal manual tuning. For more difficult problems, they also provide solid foundations on top of which algorithms with higher performance can be built.

We will use the PET compartmental model example for illustrative purpose in this section. More comprehensive performance comparisons can be found in Section 5.5. We will consider both the simulated and the real data (see Section 2.3). For the real data, to ease the presentation, instead of visualizing results for a quarter of a million data sets, we consider three typical voxels, shown in Figure 5.1. As we can see, they vary considerably in characteristics though all can be described as "typical" PET data. Most graphical presentations will be for these three data sets while summarizing statistics such as the reduction of variances will be based on simulations for all the real data sets. The purpose of the current work is to advocate robust and self-tuning algorithms. The variability in the data sets provides excellent test examples. In addition, in this section, the models are configured with the non-informative priors without ordering (see Section 3.2.3) and thus the parameters are exchangeable, similar to that of a mixture model. This creates a multimodal posterior surface for models with two or more compartments.

5.3.1 *Improved univariate numerical integration*

As seen in the last section, the path sampling estimator requires evaluation of the expectation,

$$\mathbb{E}_{\pi_\alpha}\Big[\frac{\mathrm{d}\log\gamma_\alpha(X)}{\mathrm{d}\alpha}\Big]$$

for $\alpha \in [0,1]$, which can be approximated by importance sampling using samples generated by an SMC sampler operating on the sequence of distributions $\{\pi_t =$

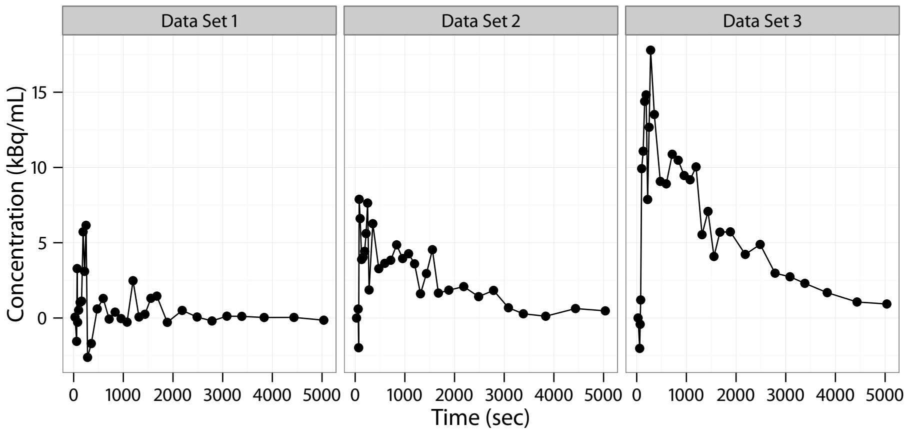


Figure 5.1 Typical real PET data.

$\pi_{\alpha_t} = \gamma_{\alpha_t}/Z_t\}_{t=0}^{T}$ directly for $\alpha \in \{\alpha_t\}_{t=0}^{T}$. For arbitrary $\alpha \in [0,1]$, finding t such that $\alpha \in (\alpha_{t-1}, \alpha_t)$, the expectation can be easily approximated using existing SMC samples – the quantities required in the importance weights to obtain such an estimate have already been calculated during the running of the SMC algorithm and such computations have little computational cost.

As noted by [48] we can use more sophisticated numerical integration strategies to reduce the path sampling estimator bias. For example, higher order Newton-Cotes rules rather than the Trapezoidal rule can be implemented straightforwardly. In the case of SMC it is especially straightforward to estimate the required expectations at arbitrary α and thus we can use higher order integration schemes. We can also use numerical integrations which make use of a finer mesh $\{\alpha_t{}'\}_{t=0}^{T'}$ than $\{\alpha_t\}_{t=0}^{T}$. Since higher order numerical integrations based on approximations of derivatives obtained from Monte Carlo methods may potentially be unstable in some situations, the second approach can be more appealing in some applications. A demonstration of the bias reduction effect is provided in Section 5.5.3.

5.3.2 *Adaptive specification of distributions*

In settings in which the importance weights at time t depend only upon the samples at time $t-1$, such as that considered here, it is relatively straightforward to consider sample-dependent, adaptive specification of the sequence of distributions (typically by choosing the value of a tempering parameter, such as $\alpha_t = \alpha_k(t/T_k)$ in Algorithm 5.2) based upon the current samples. In [84] such a method of adaptively placing the distributions in SMC algorithms based on controlling the rate at which the effective sample size (ESS; [100]) falls was proposed. With very little computation cost, this provides an automatic method of specifying a tempering schedule in such a way that the ESS decays in a regular fashion. In [140, Algorithm 2] a similar technique is used but by moving the particle system only when it resamples. They are in a setting which would be equivalent to resampling at every time step (with longer time steps, followed by multiple applications of the MCMC kernel) in our formulation. We advocate resampling only adaptively when ESS is smaller than a certain preset threshold, and here we propose a more general adaptive scheme for the selection of the sequence of distributions which has significantly better properties when adaptive resampling is employed.

The ESS was designed to assess the loss of efficiency arising from the use of simple weighted samples (rather than random samples from the distribution of interest) in the computation of expectations. It is obtained by considering a sample approximation of a low order Taylor expansion of the variance of the importance sampling estimator of an arbitrary test function to that of the simple Monte Carlo estimator; the test function itself vanishes from the expression as a consequence of this low order expansion.

In our context, the ESS calculated using the current weight of each particle is simply,

$$\mathrm{ESS}_t = \left[\sum_{i=1}^{N}\left(\frac{W_{t-1}^{(i)}w_t^{(i)}}{\sum_{j=1}^{N}W_{t-1}^{(j)}w_t^{(j)}}\right)^2\right]^{-1} = \frac{\left(\sum_{i=1}^{N}W_{t-1}^{(i)}w_t^{(i)}\right)^2}{\sum_{j=1}^{N}\left(W_{t-1}^{(j)}\right)^2\left(w_t^{(j)}\right)^2} \tag{5.30}$$

where $\{W_{t-1}^{(i)}\}_{i=1}^{N}$ denote the *normalized weights* at the end of iteration $t-1$, and $\{w_t^{(i)}\}_{i=1}^{N}$ denote the *unnormalized* incremental weights during iteration t. It is clearly appropriate to use this quantity (which corresponds to the coefficient of variation of the current normalized importance weights) to assess weight degeneracy and to make decisions about appropriate resampling times [38] but it is rather less apparent that it is the correct quantity to consider when adaptively specifying a sequence of distributions in an SMC sampler.

The ESS of the current sample weights tells us about the accumulated mismatch between proposal and target distributions (on an extended space including the full trajectory of the sample paths) since the last resampling time. Fixing either the relative or absolute reduction in ESS between successive distributions does *not* lead to a common discrepancy between successive *target* distributions unless resampling is conducted after every iteration as will be demonstrated below.

When specifying a sequence of distributions it is natural to aim for a similar discrepancy between each pair of successive distributions. In the context of our setting, the natural question to ask is consequently, how large can we make $\alpha_t - \alpha_{t-1}$ whilst ensuring that π_t remains sufficiently similar to π_{t-1}. One way to measure the discrepancy would be to consider how good an importance sampling proposal π_{t-1} would be for the estimation of expectations under π_t and a natural way to measure this is via the sample approximation of a Taylor expansion of the relative variance of such an estimator exactly as in the ESS.

The exact ESS of an importance sample of size N with proposal π_{t-1} and target π_t is defined as [100],

$$\text{Exact ESS}_t = \frac{N}{1 + \operatorname{var}_{\pi_{t-1}}\left[\frac{\pi_t(X)}{\pi_{t-1}(X)}\right]} \tag{5.31}$$

which is widely approximated by the empirical equivalent, replacing the denominator with the empirical mean squared normalized importance weights.

In the context of adaptive specification of an SMC tempering schedule, we are interested in the discrepancy between adjacent distributions, π_{t-1} and π_t, and so the ESS defined in Equation (5.31) is a natural quantity to consider.

However, the ESS as used in the recent SMC literature is invariably computed using the empirical mean of squared normalized importance weights of the current population. If these importance weights have been accumulated over several iterations, then they coincide with an ESS based on the excursion since the last resampling epoch. A new quantity, termed *CESS* later, proposed in this work instead uses a weighted sample from π_{t-1} to approximate exactly the quantity described by Equation (5.31). The different name is used to distinguish this quantity from that usually termed the ESS in the SMC literature.

The approximation leading to the CESS, given samples $\{W_{t-1}^{(i)}, X_{t-1}^{(i)}\}_{i=1}^N$ and normalized incremental weights $w_t^{(i)} = \pi_t(X_{t-1}^{(i)})/\pi_{t-1}(X_{t-1}^{(i)})$ is simply,

$$\begin{aligned}\text{Exact ESS}_t = \frac{N}{1 + \mathrm{var}_{\pi_{t-1}}\left[\frac{\pi_t(X)}{\pi_{t-1}(X)}\right]} &\approx \frac{N}{\sum_{i=1}^N W_{t-1}^{(i)}\left(\frac{\pi_t(X_{t-1}^{(i)})}{\pi_{t-1}(X_{t-1}^{(i)})}\right)^2} \\ &\approx \frac{N}{\sum_{i=1}^N W_{t-1}^{(i)}\left(\frac{w_t^{(i)}}{\sum_{j=1}^N W_{t-1}^{(j)} w_t^{(j)}}\right)^2}.\end{aligned}$$

The first approximation is obtained by replacing the expectation under π_{t-1} with its weighted sample average. That is, given the last generation of the particle system $\{W_{t-1}^{(i)}, X_{t-1}^{(i)}\}_{i=1}^N$, which approximates π_{t-1}, the variance is expressed and approximated as,

$$\begin{aligned}\mathrm{var}_{\pi_{t-1}}\left[\frac{\pi_t(X)}{\pi_{t-1}(X)}\right] &= \mathbb{E}_{\pi_{t-1}}\left[\left(\frac{\pi_t(X)}{\pi_{t-1}(X)}\right)^2\right] - \left(\mathbb{E}_{\pi_{t-1}}\left[\frac{\pi_t(X)}{\pi_{t-1}(X)}\right]\right)^2 \\ &\approx \sum_{i=1}^N W_{t-1}^{(i)}\left(\frac{\pi_t(X_{t-1}^{(i)})}{\pi_{t-1}(X_{t-1}^{(i)})}\right)^2 - \left(\int \pi_{t-1}(x)\frac{\pi_t(x)}{\pi_{t-1}(x)}\,\mathrm{d}x\right)^2 \\ &= \sum_{i=1}^N W_{t-1}^{(i)}\left(\frac{\pi_t(X_{t-1}^{(i)})}{\pi_{t-1}(X_{t-1}^{(i)})}\right)^2 - 1\end{aligned}$$

The second approximation is simply obtained by rewriting the normalized inremental weights, $\pi_t(X_{t-1})/\pi_{t-1}(X_{t-1})$ as $(\gamma_t(X_{t-1})/\gamma_{t-1}(X_{t-1}))(Z_{t-1}/Z_t)$ where Z_t and Z_{t-1} are the normalizing constants of π_t and π_{t-1}, respectively. Then the ratio of the normalizing constant is replaced by the approximation 5.10

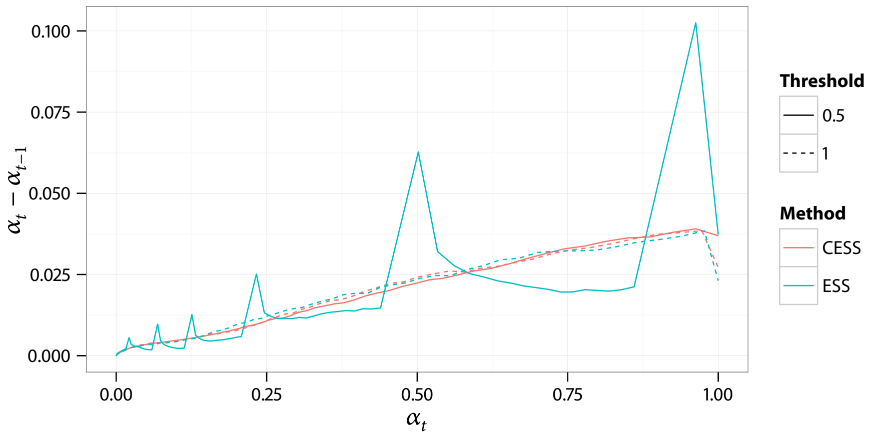


Figure 5.2 A typical plot of $\alpha_t - \alpha_{t-1}$ against α_t for the two-compartments PET model with the simulated data set using the SMC2 algorithm. The threshold is the value of ESS/N below which resampling is performed. The specifications of the adaptive parameter (ESS or CESS) are adjusted such that all four samplers use roughly the same number of distributions (about 100).

Such a procedure leads us to a quantity which we have termed the *conditional* ESS (CESS). By scaling the above approximation,

$$\text{CESS}_t = \left[\sum_{i=1}^{N} NW_{t-1}^{(i)} \left(\frac{w_t^{(i)}}{\sum_{j=1}^{N} NW_{t-1}^{(j)} w_t^{(j)}} \right)^2 \right]^{-1} = \frac{\left(\sum_{i=1}^{N} W_{t-1}^{(i)} w_t^{(i)}\right)^2}{\sum_{j=1}^{N} \frac{1}{N} W_{t-1}^{(j)} \left(w_t^{(j)}\right)^2} \qquad (5.32)$$

which is equal to the ESS only when resampling is conducted during every iteration. The factor of $1/N$ in the denominator arises from the fact that $\{W_{t-1}^{(i)}\}_{i=1}^{N}$ is normalized to sum to unity rather than to have expectation unity. The bracketed term coincides with a sample approximation (using the actual samples which are properly weighted to target π_{t-1}) of the expected sum of the unnormalized weights squared divided by the square of a sample approximation of the expected sum of unnormalized weights when considering sampling from π_{t-1} and targeting π_t by simple importance sampling.

More specifically, in practice, when using CESS to adaptively place distributions, a value CESS$^\star \in (0, 1)$ is chosen, and at each iteration t, α_t is chosen such that $\textsc{cess}_t = \textsc{cess}^\star$ (with a preset numeric error tolerance). This can be done using an algorithm such as binary search in our setting of SMC1–SMC3, since it is clear that $\textsc{cess}_t$ is monotonically decreasing when α_t increases. Figure 5.2 shows the variation of α_t when fixed reductions in ESS and CESS are used to specify the sequence of distributions, both when resampling is conducted during every iteration (or equivalently, when the value of $\textsc{ess}/N$ falls below a threshold of 1.0, where N is the number of particles) and when resampling is conducted only when the value of $\textsc{ess}/N$ falls below a threshold of 0.5. It is found that for the simulated PET data sets, the CESS-based scheme leads to a reduction in estimator variance around 20% relative to a manually tuned ($\alpha_k(t/T) = (t/T)^5$) schedule while the ESS-based strategy provides little improvement over the linear case ($\alpha_k(t/T) = t/T$) unless resampling is conducted during every iteration. The effect for the real data sets, which vary considerably from each other as seen in Figure 5.1, is more prominent. The variance reduction can be more than 50% when using the CESS-based strategy. More performance comparisons for various settings of the samplers can be found in Section 5.5.2 and 5.5.3.

Relationship of $\textsc{cess}^\star$, number of distributions, and estimator variance

It is intuitively seen that, when using a CESS-based adaptive scheme, where at each iteration t, $\textsc{cess}_t$ is fixed with a value $\textsc{cess}^\star$, the more close $\textsc{cess}^\star/N$ is to 1, the larger the number of distributions that will be placed and better estimates can be obtained. Unfortunately it is not trivial to establish a quantitative relationship among these three quantities even asymptotically. However, it is straightforward to conduct an empirical study of these relations.

We consider the simulated PET data set using the SMC2 sampler with 1,000 particles. Figure 5.3 plots the average number of distributions (from 100 simulations for each value of $\textsc{cess}^\star$) against the value of $(1 - \textsc{cess}^\star/N)$ on log scales. It can be seen that the number of distributions is proportional to $(1 - \textsc{cess}^\star/N)^{-1}$. This relation holds even for sampler configurations where a very small number of

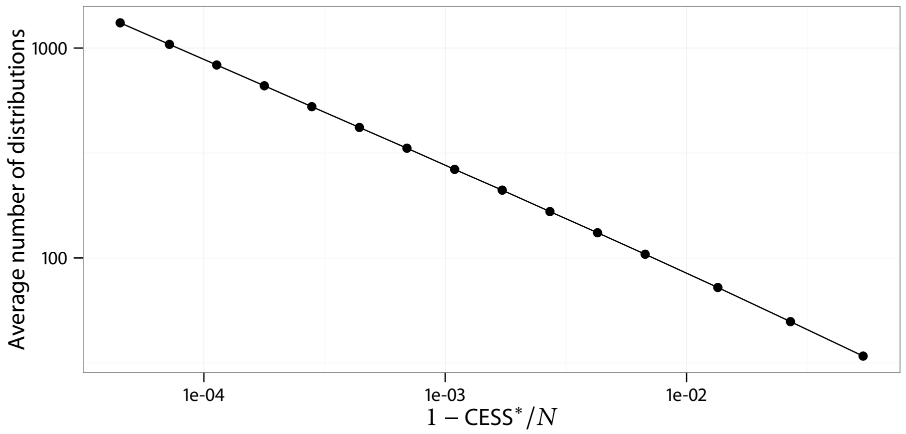


Figure 5.3 Relationship between average number of distributions and CESS* for the two-compartments PET model with the simulated data set using the SMC2 algorithm (on logarithm scale). The averages are calculated from 100 simulations for each sampler configuration.

distributions are used. Similarly, Figure 5.4 shows the relation between the variance of path sampling estimator and the value of CESS*. Similar relationship can be observed for the standard estimator (Equation (5.25)), which is not shown here.

Though the coefficient of the proportionality varies for different applications or data sets, the relation shown in Figures 5.3 and 5.4 provide a useful guideline to select the value of CESS*. A small sample experiment can be conducted before using a value of CESS* more close to 1 to obtain satisfactory estimates or to utilize given computational resources.

5.3.3 *Adaptive specification of proposals*

The SMC sampler is remarkably robust to the mixing speed of the MCMC kernels employed as can be seen in the empirical study later. However, as with any sampling algorithms, faster mixing does not harm performance and in some cases will considerably improve it. In the particular case of Metropolis random walk kernels,

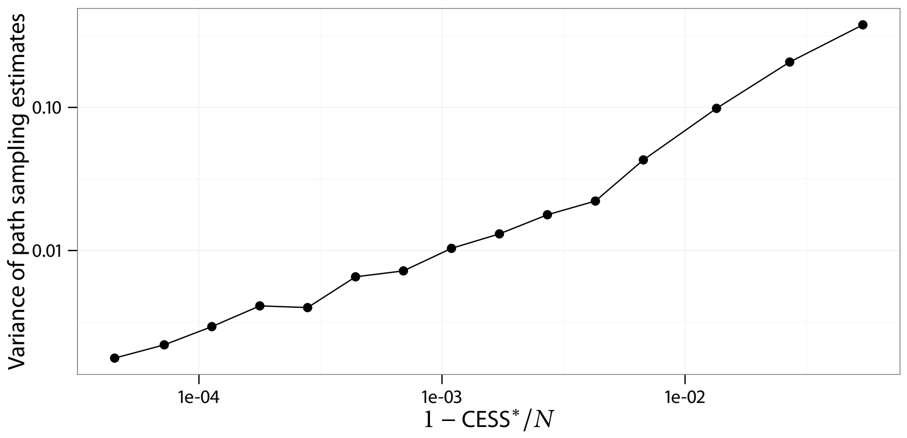


Figure 5.4 Relationship between the variance of the path sampling estimator and CESS* for the two-compartments PET model with the simulated data set using the SMC2 algorithm (on logarithm scale). The variances are calculated from 100 simulations for each sampler configuration.

the mixing speed relies on adequate proposal scales.

We use a simple approach based on [84]. They applied an idea used within adaptive MCMC methods [9] to SMC samplers by using variance of parameters estimated from its particle system approximation as the proposal scales for the next iteration, suitably scaled with reference to the dimensions of the parameters to be proposed. Although, in practice we found that such an automatic approach does not always lead to optimal acceptance rates, it generally produces satisfactory results and is simple to implement. In difficult problems alternative approaches to adaptation could be employed; one approach demonstrated in [84] is to simply employ a pair of acceptance rate thresholds and to alter the proposal scale from the simply estimated value whenever the acceptance rate falls outside those thresholds.

More sophisticated proposal strategies could undoubtedly improve performance further and warrant further investigation. One possible approach is using the Metropolis adjusted Langevin algorithm (MALA; see [138]). In summary, MALA derives a Metropolis-Hastings proposal kernel for a target π which satisfies suitable differentiability and positivity conditions, from the Langevin diffusion,

$$\mathrm{d}\,L_t = \frac{1}{2}\nabla \log \pi(L_t)\,\mathrm{d}\,t + \mathrm{d}\,B_t$$

where B_t is the standard Brownian motion. Given a state X^{t-1}, a new state is proposed by discrete approximation to the above diffusion. That is, for a fixed $h > 0$,

$$X^t \sim \mathcal{N}\Big(X^{t-1} + \frac{1}{2}\nabla \log \pi(X^{t-1}), hI_d\Big) \tag{5.33}$$

where I_d is the identity matrix and d is the dimension of the state space. The new proposed state is accepted or rejected through the usual Metropolis-Hastings algorithm. Compared to a "vanilla" random walk, which often has very robust theoretical properties, MALA is attractive when it is possible and its convergence conditions [138] can be met, because only one discrete approximation parameter h needs to be tuned for optimal performance. In addition, results from [137] suggested that MALA can be more efficient than a random walk when using optimal scalings. We could also use the particle approximation at time $t-1$ to estimate the covariance matrix of π_t and thus tune the scale h on-line. As these algorithms are known to be somewhat sensitive to scaling, and we seek approaches robust enough to employ with little user intervention, we have not investigated this strategy further in this work.

An adaptive specification of proposals is most useful when manual tuning is difficult or even impossible. We consider the three real PET data sets shown in Figure 5.1. When using adaptive proposal scales, the average acceptance rates of the four random walk blocks (for parameters $\phi_{1:r}$, $\theta_{1:r}$, τ and ν, respectively), are shown in Figure 5.5.The results are not close to the optimal value 0.234, which is commonly used in practice, but they are more than acceptable. In contrast, in Figure 5.6 we show the average acceptance rates when using a scheme of proposal scales which is obtained from another typical real PET data set. The scheme is tuned such that for that particular data set and for all parameters, the acceptance rates fall within the range $[0.2, 0.4]$ for $\alpha_t \in [0, 1]$. It can be seen that, not only do the results vary considerably due to the variety of the data sets, which is expected, but also for one of

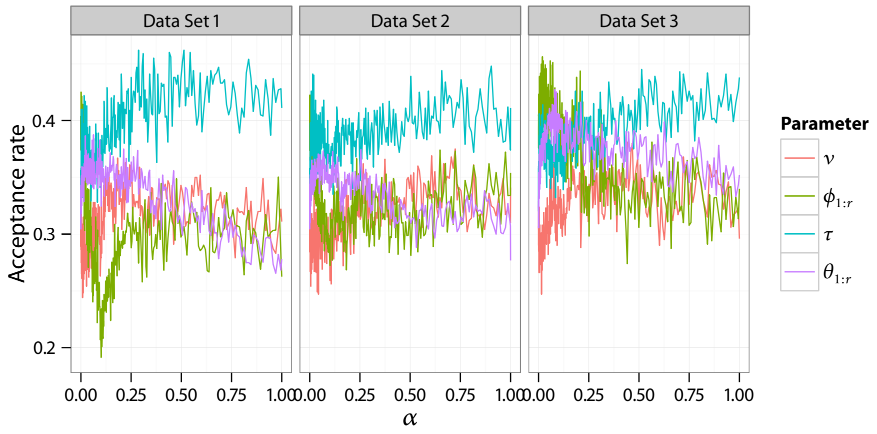


Figure 5.5 Average random walk acceptance rates for the two-compartments PET model with data sets in Figure 5.1 using adaptive proposal scales.

the data sets, the parameter ν fails to move at all. Considering that there are about a quarter of a million such data sets to be estimated in a single PET scan, adaptively specifying the proposal scales is the only feasible approach. Note that, this problem is not unique to the PET compartmental model at all. In many realistic applications, there are a large number of data sets which vary considerably from each other.

5.3.4 *An automatic and adaptive algorithm*

With the above refinements, we are ready to implement the SMC2 algorithm with minimal tuning and application specific effort while providing robust and accurate estimates of the model evidence $p(\boldsymbol{y}|\mathcal{M}_k)$. First the geometric annealing scheme that connects the prior $\pi(\theta_k|\mathcal{M}_k)$ and the posterior $\pi(\theta_k|\boldsymbol{y},\mathcal{M}_k)$, provides a smooth path for a wide range of problems.

Second, the actual annealing schedule under this scheme can be determined through the adaptive schedule as described above. The advantage of the adaptive schedule will be shown empirically later.

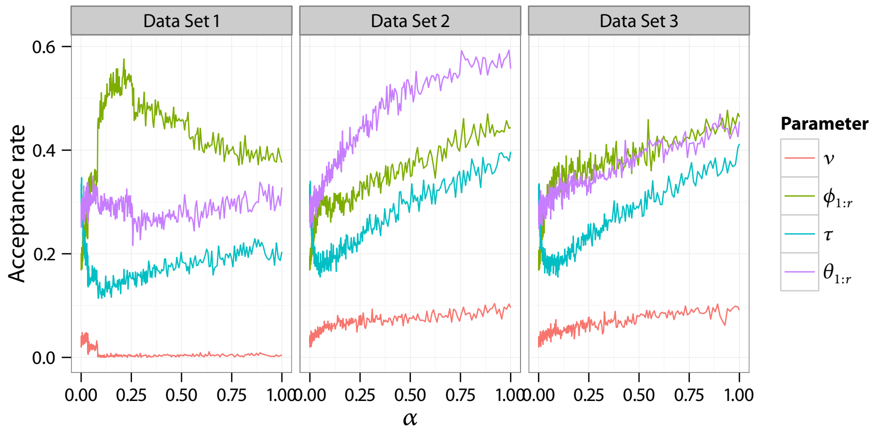


Figure 5.6 Average random walk acceptance rates for the two-compartments PET model with data sets in Figure 5.1 using fixed proposal scales. The proposal scales are calibrated for a single data set other than the three shown in Figure 5.1. It is clear that clear that proposal scales calibrated for a single data set cannot be applied other data sets efficiently. Adaptive and automatic scheme is required in this situation.

Third, we can adaptively specify the Metropolis random walk (or MALA) proposal scales through the estimation of their scaling parameters as the sampler iterates. In contrast to the MCMC setting, where such adaptive algorithms will usually require a burn-in period, which will not be used for further estimation, in SMC, the variance and covariance estimates come at almost no cost, as all the samples will later be used for marginal likelihood estimation. Additionally, adaptation within SMC does not require separate theoretical justification in the sense that in principle the Strong Law of Large Numbers (SLLN) holds directly – something which can significantly complicate the development of effective, theoretically justified schemes in the MCMC setting. Nonetheless, some asymptotic results can be found in [21], including a version of the Central Limit Theorem for adaptive resampling among other results. Alternatively, we can also specify the proposal scales in a deterministic, but sensible way. Since SMC algorithms are relatively robust to the change of scales,

```
Accuracy control
    Set constant CESS$^\star \in (0, 1)$, using a small pilot simulation if necessary.
Initialization: Set $t \leftarrow 0$.
    Perform the Initialization step as in Algorithm 5.1 or 5.2
Iteration: Set $t \leftarrow t + 1$
    Step size selection
        Use a binary search to find $\alpha^\star$ such that CESS$_{\alpha^\star}$ = CESS$^\star$
        Set $\alpha_t \leftarrow \alpha^\star$ if $\alpha^\star \leq 1$, otherwise set $\alpha_t \leftarrow 1$
    Proposal scale calibration
        Computing the importance sampling estimates of first two moments of parameters.
        Set the proposal scale of the Markov proposal $K_t$ with the estimated parameter variances.
    Perform the Iteration step as in Algorithm 5.1 or 5.2 with the found $\alpha_t$ and proposal scales.
Repeat the Iteration step until $\alpha_t = 1$ then set $T = t$.
```

Algorithm 5.4 An Automatic, Generic Algorithm for Bayesian Model Comparison

such deterministic scales will not require the same degree of tuning as is required to obtain good performance in MCMC algorithms.

Though we described the algorithm in the setting of SMC2, it can also be applied to other SMC strategies. SMC1 is less straightforward as the between model moves still require efforts to design and implement. In SMC3, the specification of the sequences between posterior distributions are less generic compared to the geometric annealing scheme in SMC2. However, the adaptive schedule and automatic tuning of MCMC proposal scales, both can be applied in these two algorithms in principal. We outline the strategy in Algorithm 5.4.

As laid out above, the algorithms require minimal tuning. Theirs robustness,

Table 5.1 The standard Bayes factor estimates for a simulated PET data set using the SMC2 algorithm.

	Proposal scales	
	Fixed	Adaptive
Fixed annealing ($\alpha_k(t/_k) = (t/T_k)^5$)	1.6 ± 0.27	1.6 ± 0.22
CESS-based adaptive annealing	1.6 ± 0.19	1.6 ± 0.15

All four samplers are configured such that about 200 distributions are used. The estimates ($\log B_{2,1} \pm$ S.D.) are obtained from 100 simulations for each sampler.

accuracy and efficiency will be shown in Section 5.5 through comprehensive empirical studies. Here we show some interesting yet intuitively expected results. We consider the simulated PET data sets, using an SMC2 sampler with 1,000 particles. It is already shown that using either the adaptive specification of distribution placement or the MCMC proposal scales can give better results. The combination of the two can lead to even better results. The use of the CESS-based schedule scheme will not only place more distributions where the target distributions changes more rapidly, but also it will place more distributions where the MCMC algorithm mixes more slowly. To illustrate the idea, we consider four configurations of the sampler. The proposal scales are specified either adaptively or using a fixed scheme which is manually tuned. The placement of the distributions is either CESS-based or using a fixed schedule $\alpha_t = \alpha_k(t/T_k) = (t/T_k)^5$.

The results are shown in Table 5.1. More comprehensive results can be found in Section 5.5.3. However, from this particular example, we can see that the combination of the two adaptive schemes provides superior performance at very little computational cost. Table 5.2 shows actually equivalent results. Instead of fixing the number of distributions, we configured the samplers such that they all give roughly the same precision of the estimates. It is rather obvious to see that the use of adaptive methods actually give a considerable reduction of computational cost for a given desired precision of the estimates.

Table 5.2 The number of distributions used for a simulated PET data set using the SMC2 algorithm.

	Proposal scales	
	Fixed	Adaptive
Fixed annealing ($\alpha_k(t/T_k) = (t/T_k)^5$)	200	182
CESS-based adaptive annealing	175	157

All four samplers are configured such that the standard Bayes factor estimates ($\log B_{2,1}$ ± S.D.), obtained from 100 simulations for each sampler, have a standard deviation of approximately 0.27.

Although further enhancements and refinements are clearly possible, we focus in the remainder of this chapter on this simple, generic algorithm which can be easily implemented in any application and has proved sufficiently powerful to provide good estimation in the examples we have encountered thus far.

5.4 THEORETICAL CONSIDERATIONS

The convergence results for the standard estimator can be found in [39] and references therein. In this work, given our advocation of SMC2-PS, we extend the results for the path sampling estimator from SMC samplers. Here we present Proposition 5.1, which is specific to path sampling estimator using the simplest Trapezoidal approach to numerical integration. It follows as a simple corollary to a more general result given in Appendix B.1 which could be used to characterize more general numerical integration schemes.

PROPOSITION 5.1. *Under the same regularity conditions as are required for the central limit theorem given in [39] to hold, given an* SMC *sampler that iterates over a sequence of distributions* $\{\pi_t = \gamma_{\alpha_t}/Z_{\alpha_t}\}_{t=0}^{T}$ *and applies multinomial resampling at each iteration, the path sampling estimator,* $\hat{\Xi}_T^N$, *as defined in Equation* (5.28) *obeys a central limit theorem in the following sense: Let* $\xi_t(\cdot) = \frac{\mathrm{d}\log\gamma_\alpha(\cdot)}{\mathrm{d}\,\alpha}\Big|_{\alpha=\alpha_t}$, $\beta_0 = \alpha_0/2$,

$\beta_T = \alpha_T/2$ and for $t = 1,\dots,T-1$, $\beta_t = (\alpha_{t+1}-\alpha_{t-1})/2$, then, provided ξ_t is bounded,

$$\lim_{N\to\infty} \sqrt{N}(\hat{\Xi}_T^N - \Xi_T) \xrightarrow{D} \mathcal{N}(0, V_T(\xi_{0:T})) \tag{5.34}$$

where $\xrightarrow{D}$ denotes convergence in distribution and V_t, $0 \le t \le T$ is defined by the following recursion,

$$V_0(\xi_0) = \beta_0^2 \int \pi_0(x_0)(\xi_0(x_0) - \pi_0(\xi_0))^2 dx_0 \tag{5.35}$$

$$\begin{aligned} V_t(\xi_{0:t}) =& V_{t-1}\Big(\xi_{0:t-2}, \xi_{t-1} + \frac{\beta_t}{\beta_{t-1}}\frac{\pi_t(\cdot)}{\pi_{t-1}(\cdot)} \int K_t(\cdot, x_t)(\xi_t(x_t) - \pi_t(\xi_t))\,\mathrm{d}\,x_t\Big) \\ &+ \beta_t^2 \int \frac{\pi_t(x_{t-1})^2}{\pi_{t-1}(x_{t-1})} K_t(x_{t-1}, x_t)(\xi_t(x_t) - \pi_t(\xi_t))^2)\,\mathrm{d}\,x_{t-1}\,\mathrm{d}\,x_t. \end{aligned} \tag{5.36}$$

Application of similar arguments to those used in [39] to the historical process associated with the SMC sampler would lead to essentially the same result, but we find this approach more transparent. We note that much recent analysis of SMC algorithms has focused on relaxing the relatively strong assumptions used in the results upon which this result is based – looking at more general resampling schemes [38] and relaxing compactness assumptions [162] for example. However, we feel that this simple result is sufficient to show the relationship between the path sampling and simple estimators and that in this instance the relative simplicity of the resulting expression justifies these stronger assumptions.

5.5 PERFORMANCE COMPARISON

In this section, we will use three examples to illustrate the algorithms. The Gaussian mixture model is discussed first, with implementations for all three SMC algorithms with comparison to RJMCMC and population MCMC. It will be shown that all five algorithms agree on the results while the performance in terms of Monte Carlo variance varies considerably. It will also be demonstrated how the adaptive refinements of the algorithms behaves in practice. We will reach the conclusion that considering ease of implementation, performance and generality, the SMC2 algorithm is most promising among all three strategies.

Then two more realistic examples, a nonlinear ODE model and the PET compartmental model are used to study the performance and robustness of algorithm SMC2 compared to AIS and population MCMC. Various configurations of the algorithms are considered including both sequential and parallelized implementations.

The C++ implementations, which make use of the vSMC library of [166], of all examples can be found at `https://github.com/zhouyan/vSMCExample` and the library is also introduced in Chapter 6.

5.5.1 *Gaussian mixture model*

Since [132], the Gaussian mixture model (GMM) has provided a canonical example of a model-order-determination problem. We use the model formulation of [39] to illustrate the efficiency and robustness of the methods proposed in this chapter compared to other approaches. The model is as follows; data $\boldsymbol{y} = (y_1, \dots, y_n)$ are independently and identically distributed as

$$y_i | \theta_r \sim \sum_{j=1}^{r} \omega_j \mathcal{N}(\mu_j, \lambda_j^{-1})$$

where $\mathcal{N}(\mu_j, \lambda_j^{-1})$ denotes the Normal distribution with mean μ_j and precision λ_j; $\theta_r = (\mu_{1:r}, \lambda_{1:r}, \omega_{1:r})$ and r is the number of components in each model. The parameter space is thus $\mathbb{R}^r \times (\mathbb{R}^+)^r \times \Delta_r$ where $\Delta_r = \{\omega_{1:r} : 0 \le \omega_j \le 1; \sum_{j=1}^{r} \omega_j = 1\}$ is the standard r-simplex. The priors which are the same for each component are taken to be $\mu_j \sim \mathcal{N}(\xi, \kappa^{-1})$, $\lambda_j \sim \mathcal{G}a(\nu, \chi)$ and $\omega_{1:r} \sim \mathcal{D}(\rho)$ where $\mathcal{D}(\rho)$ is the symmetric Dirichlet distribution with parameter ρ and $\mathcal{G}a(\nu, \chi)$ is the Gamma distribution with shape ν and scale χ. The prior parameters are set in the same manner as in [132]. Specifically, let y_{MIN} and y_{MAX} be the minimum and maximum of data $\boldsymbol{y}$, the prior parameters are set such that

$$\xi = (y_{\text{MAX}} + y_{\text{MIN}})/2, \quad \kappa = (y_{\text{MAX}} - y_{\text{MIN}})^{-2}, \quad \nu = 2, \quad \chi = 50\kappa, \quad \rho = 1$$

The data is simulated from a four components model with $\mu_{1:4} = (-3, 0, 3, 6)$, and $\lambda_j = 2$, $\omega_j = 0.25$, $j = 1, \dots, 4$.

We consider several algorithms. First the RJMCMC algorithm as in [132], and second an implementation of the SMC1 algorithm. Next AIS, population MCMC and SMC2 are used for within-model simulations. The last is an implementation of the SMC3 algorithm. In all the algorithms, the local move which does not change the dimension of the model is constructed as a composition of Metropolis-Hastings random walk kernels:

1. Update $\mu_{1:r}$ using a multivariate Normal random walk proposal.
2. Update $\lambda_{1:r}$ using a multivariate Normal random walk on logarithmic scale, i.e., on $\log \lambda_j$, $j = 1, \dots, r$.
3. Update $\omega_{1:r}$ using a multivariate Normal random walk on logit scale, i.e., on ω_j / ω_r, $j = 1, \dots, r - 1$.

The RJMCMC, SMC1 and SMC3 algorithms use two additional pairs of reversible jump moves. The first is a combine and split move; the second is a birth and death move. Both are constructed in the same manner as in [132]. Also in these implementations, an adjacency condition was imposed on the means $\mu_{1:r}$, such that $\mu_1 < \mu_2 < \cdots < \mu_r$. No such restriction was used for other algorithms.

In the SMC1, SMC2, AIS and population MCMC implementations, the distributions are chosen with a geometric schedule, i.e., as in Equation (5.23) for SMC1 and Equation (5.24) for the other three. This annealing scheme has been used in [39, 82] and many other works. The geometric scheme can also be seen in [24] for population MCMC tempering. A schedule $\alpha(t/T) = (t/T)^p$, with $p = 2$ was used. The rationale behind this particular schedule can be seen in [24] and other values of p were also tried while $p \approx 2$ performs best in this particular example. The adaptive schedule was also implemented for SMC2 and AIS algorithms.

The proposal scales for each block of the random walks are specified dynamically according to values of $\alpha(t/T)$ for the SMC2 and AIS algorithms and also manually tuned for other algorithms such that the acceptance rates fall in $[0.2, 0.5]$. Later for the SMC2 and AIS algorithms, we also consider adaptive schedule of the distribution specification parameter $\alpha(t/T)$ and the proposal scales of the random walks.

For SMC2, SMC3 and AIS we consider both the direct estimator and the path sampling estimator. For population MCMC we consider the path sampling estimator.

Results

The SMC1 implementation uses 10,000 particles and 500 distributions. The RJMCMC implementation uses five million iterations in addition to one million iterations of burn-in period for adaptation. The resulting estimates of model probabilities are shown in Table 5.3.

The SMC2, SMC3 and AIS implementations use 1,000 particles and 500 iterations. Population MCMC implementation uses 50 chains and 10,000 iterations in addition to 10,000 iterations used for adaptation (the burn-in period) – these implementations have approximately equal computational costs. For all algorithms where resampling is needed, the stratified resampling algorithm is applied (see Section 5.1.1).

From the results obtained under the SMC1 and RJMCMC algorithms it is clear that, in this particular example, simulations for models with fewer than ten components are adequate to characterize the model space. Therefore, under this configuration, the cost is roughly the same in terms of computational resources as that of the SMC1 and RJMCMC algorithms. From the results of RJMCMC and SMC1, we consider the four and five components models (i.e., the true model and the most competitive one amongst the others). The estimates are shown in Table 5.4 which, like all of the other tables in this section, summarizes the Monte Carlo variability of 100 replicate runs of each algorithm.

From Tables 5.3 and 5.4, it can be seen that the standard estimators (RJMCMC, SMC1, SMC2-DS, SMC3-DS and AIS-DS) agree with each other. Among the path sampling estimators, SMC2-PS and AIS-PS have little bias. SMC3-PS shows a little more bias. The population MCMC algorithm has a considerably larger bias as the

Table 5.3 Gaussian mixture model posterior model probability estimates obtained via SMC1 and RJMCMC.

		Number of components						
Quantity	Algorithm	≤ 2	3	4	5	6	7	≥ 8
$\mathbb{P}(\mathcal{M}_k \mid \boldsymbol{y})$	SMC1	0	0.0022	0.89	0.10	0.0064	0.0014	0
	RJMCMC	0	0.0013	0.89	0.10	0.0062	0.0025	0
$\log B_{4,k}$	SMC1	∞	6.00	0	2.15	4.93	6.45	∞
	RJMCMC	∞	6.53	0	2.15	4.97	5.87	∞

Table 5.4 Gaussian mixture model the Bayes factor estimates obtained via SMC2, SMC3, AIS and population MCMC.

Algorithm	SMC2		SMC3		AIS		PMCMC
Estimator	DS	PS	DS	PS	DS	PS	PS
$\log B_{4,5}$	2.15	2.15	2.16	2.21	2.16	2.17	2.63
S.D.	0.25	***0.22***	0.61	0.62	1.12	1.10	0.41

number of distributions is relatively small (as noted previously, a larger number will negatively affect the mixing speed). In terms of Monte Carlo variance, in Table 5.4, SMC2 clearly has an advantage compared to its no-resampling variant, AIS. The differences of Monte Carlo standard deviation between SMC2, SMC3 and population MCMC, although they do not affect model selection in this particular example, are considerable.

Effects of resampling It is clear from these results that resampling (when required) can substantially improve the estimation of normalizing constants within an SMC framework. This does not contradict the statement in [39] which suggests that resampling may not much help when the normalizing constant is the object of interest, the theoretical argument which supports this relies upon the assumption

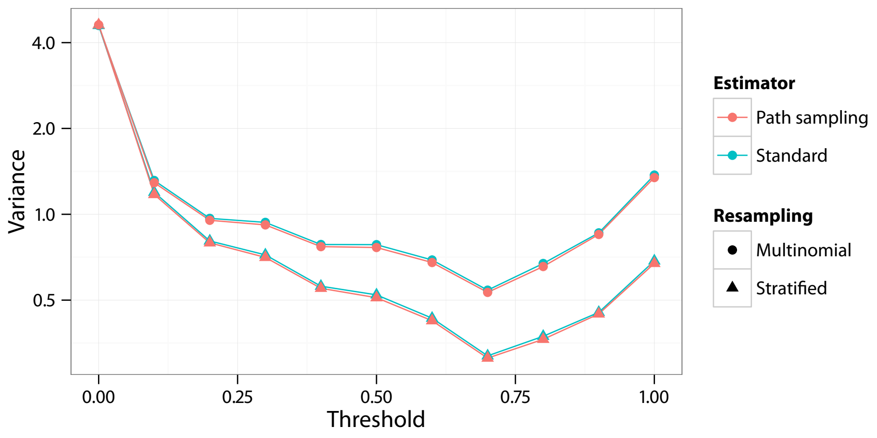


Figure 5.7 Monte Carlo variance of standard estimator and path sampling estimator using different threshold of ESS/N for Gaussian mixture model using the SMC2 algorithm and the stratified resampling (on logarithm scale). The variances are calculated from 100 simulations for each sampler configuration.

that the Markov kernel used to mutate the particles mixes extremely rapidly and the result is obtained under the assumption that resampling is performed after every iteration. When the Markov kernel is not so rapidly mixing, the additional stability provided by the resampling operation can out-weight the attendant increase in Monte Carlo variance and that is what we observed here (and in the case of the other examples considered below; results not shown.)

On the other hand, in this work we advocate resampling adaptively instead of resampling every iteration. The proposed adaptive schedule also has significant advantage in this situation. It is of interest to see if adaptive resampling has performance improvement when compared to resampling every iteration. We consider using the SMC2 algorithm with 1,000 particles, 100 distributions scheduled as $\alpha(t/T) = (t/T)^2$, and various threshold of ESS/N under which resampling is performed. The Monte Carlo variance of both the standard estimator and the path sampling estimator is shown in Figure 5.7. It can be shown that neither performing resampling every iteration or never performing resampling (i.e., the AIS algorithm), gives optimal results. Though it may be difficult to determine an optimal value, the commonly used value 0.5 seems to be suitable for this particular example, in the sense that the variance is only slightly higher than that of sampler performing resampling at every iteration while the computational cost is reduced. The reduction in computational cost might be negligible for some applications. However, as noted before, when parallel computing is considered, frequent resampling can be a bottleneck of performance in reality.

Effects of adaptive schedules To assess the evolution of distributions with an adaptive schedule, we consider the relation between $\alpha_t - \alpha_{t-1}$ and α_t. As stated before, one of the motivations of using CESS for adaptive placement of distribution is to ensure that α_t evolves the same path regardless of the resampling strategies. Earlier in Figure 5.2 (page 5.2) we showed the evolution of α_t when fixing ESS or CESS and resampling every iteration or only when $\text{ESS}/N < N$, where N is the number of particles. As shown in the plot, when fixing CESS, the evolution of the distributions is not affected by the resampling strategy. In contrast, fixing ESS yields a sequence of distributions which depends strongly upon the resampling strategy.

In terms of the actual performance when using the CESS adaptive strategy in the SMC2 and AIS algorithms, a reduction of standard deviation of 20% was observed comparing to $\alpha(t/T) = (t/T)^2$, the one shown in Table 5.4. When applied to the SMC3 algorithm, 50% reduction was observed. If the ESS adaptive strategy is used instead, similar standard deviation reduction is observed when resampling is performed every iteration but no significant effect was observed when resampling was only performed when $\text{ESS}/N < 0.5$ (i.e., using ESS rather than CESS entirely eliminated the benefit.)

Effects of adaptive proposal scales When using the SMC2 algorithm, if the adaptive strategy of [9] is applied, where the important sampling estimates of the variance of parameters are included in the adaptation, the acceptance rates fall within $[0.2, 0.5]$ dynamically without manual tuning as for the results in Table 5.4. It should be noted that in this particular example, it is the variance of $\log \lambda_{1:r}$ being estimated as the corresponding random walk block operates on the log scale. The same principle applies to the weight parameters $\omega_{1:r}$, which have random walks on the logit scale. Approximately 20% reduction in standard deviation was observed.

5.5.2 *Nonlinear ordinary differential equations*

In this section, this will now be further explored in a more complex model, a non-linear ordinary differential equations (ODE) system. This model, which was studied in [24], is known as the Goodwin model. The ODE system, for an r-component model, is,

$$\begin{aligned}
\frac{\mathrm{d}X_1(t)}{\mathrm{d}t} &= \frac{a_1}{1 + a_2 X_r(t)^\rho} - \alpha X_1(t) \\
\frac{\mathrm{d}X_i(t)}{\mathrm{d}t} &= k_{i-1} X_{i-1}(t) - \alpha X_i(t) && i = 2, \dots, r \\
X_i(0) &= 0 && i = 1, \dots, r
\end{aligned}$$

The parameters $\{\alpha, a_1, a_2, k_{1:r-1}\}$ have common prior distribution $\mathcal{G}a(0.1, 0.1)$. Under this setting, $X_{1:r}(t)$ can exhibit either unstable oscillation or a constant steady state. The data are simulated for $r = 3$ and 5 at equally spaced time points from 0 to 60, with time step 0.5. The last 80 data points of $(X_1(t), X_2(t))$ are used for inference. Normally-distributed noise with standard deviation $\sigma = 0.2$ is added to the simulated data. Following [24], the variance of the additive measurement error is assumed to be known. Therefore, the posterior distribution has $r + 2$ parameters for an r-component model.

As shown in [24], when $\rho > 8$, due to the possible instability of the ODE system, the posterior can have a considerable number of local modes. In this example, we set $\rho = 10$. Also, as the solution to the ODE system is somewhat unstable, slightly different data can result in very different posterior distributions.

The example from the previous section suggests that SMC2 performs well relative to the other SMC possibilities. Given the wide range of settings in which it can be easily deployed, we will now concentrate further on this method. It also suggests that in the simple case of Gaussian mixtures, a complete adaptive strategy for both distribution specification and proposal scales works well.

Results

We compare results from the SMC2 and population MCMC algorithms. For the SMC implementation, 1,000 particles and 500 iterations were used, with the distributions specified specified by Equation (5.24), with $\alpha_k(t/T_k) = (t/T)^5$, or via the completely adaptive specification. For population MCMC algorithm, 50,000 iterations are performed for the adaptation of the proposal scales (the burn-in period), and another 10,000 iterations are used for inference. The same tempering as was used for SMC is used here. Note that, in a sequential implementation of population MCMC, with each iteration updating one local chain and attempting a global exchange, the computational cost of after burn-in iterations is roughly the same as the entire SMC algorithm. In addition, changing T within the range of the number of cores available does not substantially change the computational cost of a generic parallel implementation of population MCMC algorithm. We compare results from $T = 10, 30$, and 100. For the SMC algorithms, stratified resampling is applied.

The results for data generated from the simple model ($r = 3$) and complex model ($r = 5$), again summarizing variability amongst 100 runs of each algorithm, are shown in Table 5.5 and 5.7, respectively. The model selection results, displayed as the frequencies of each model being selected by the Bayes factor using estimators from each algorithm, are shown in Table 5.6 and 5.8, respectively. It can be seen that all SMC algorithms and population MCMC algorithms with sufficient large numbers of distributions can give accurate results of model selection. With a smaller number of distributions, the population MCMC algorithm may occasionally choose a wrong model. However, the accuracy of the estimators differs considerably among these algorithms.

As shown in both cases, the number of distributions can affect the performance of population MCMC algorithms considerably. When using 10 distributions, large bias from numerical integration for path sampling estimator was observed, as expected. With 30 distributions, the performance is comparable to the SMC2 sampler, though some bias is still observable. With 100 distributions, there is a much larger variance because, with more chains, the information travels more slowly from rapidly mixing chains to slowly mixing ones and consequently the mixing of the overall system is inhibited.

The SMC algorithm provides results comparable to the best of three population MCMC implementations in essentially all settings, including one in which both the annealing schedule and proposal scaling were fully automatic. In fact, the completely adaptive strategy was the most successful.

It can be seen that increasing the number of distributions not only reduces the bias of numerical integration for path sampling estimator, but also reduces the variance considerably. On the other hand increasing the number of particles can only reduce the variance of the estimates, in accordance with the Central Limit Theorem (see [39] for the standard estimator and extensions for path sampling estimator, Proposition 5.1). The bias arises mostly from the numerical integration scheme and cannot be reduced by using more particles. Though there exists the trade-off between the number of particles and the number distributions, increasing either of them can always benefit the accuracy of the estimators. We will study this trade-off more carefully with the next more realistic example.

Table 5.5 Nonlinear ODE model marginal likelihood and the Bayes factor estimates with data generated from a simple (three components) model.

				Marginal likelihood $\log p(\boldsymbol{y}\|\mathcal{M}_k) \pm$ S.D.		Bayes factor $\log B_{3,5} \pm$ S.D.
T	Proposal	Annealing	Algorithm	$r = 3$	$r = 5$	
10	Manual	Prior (5)	PMCMC	-109.7 ± 3.2	-120.3 ± 2.5	10.6 ± 3.8
30				*-105.0±1.2*	*-116.1±2.2*	*11.2±2.5*
100				-134.7 ± 7.9	-144.1 ± 6.2	9.4 ± 11.2
500	Manual	Prior (5)	SMC2-DS	-104.6 ± 2.0	-112.7 ± 1.8	8.1 ± 2.8
			SMC2-PS	-104.5 ± 1.8	-112.7 ± 1.5	8.2 ± 2.5
500	Manual	Adaptive	SMC2-DS	-104.5 ± 1.1	-112.7 ± 1.1	8.1 ± 1.6
			SMC2-PS	-104.6 ± 1.0	-112.8 ± 1.0	8.2 ± 1.5
500	Adaptive	Adaptive	SMC2-DS	-104.5 ± 0.5	-112.7 ± 0.4	8.1 ± 0.8
			SMC2-PS	***-104.6±0.4***	***-112.8±0.3***	***8.1±0.6***

T: The number of distributions.
Proposal: The proposal scales of the MCMC kernels.
Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.
Red: The estimate with the smallest variance among all algorithms settings.
Green: The estimate with the smallest variance for a single algorithm (SMC2 or PMCMC) among different settings.

Table 5.6 Frequencies of models selected by the Bayes factor (%) for the nonlinear ODE model with data generated from a simple (three components) model.

				Number of components				
T	Proposal	Annealing	Algorithm	2	3	4	5	6
10	Manual	Prior (5)	PMCMC	2	96	2	0	0
30				0	100	0	0	0
100				0	100	0	0	0
500	Manual	Prior (5)	SMC2-DS	0	100	0	0	0
			SMC2-PS	0	100	0	0	0
500	Manual	Adaptive	SMC2-DS	0	100	0	0	0
			SMC2-PS	0	100	0	0	0
500	Adaptive	Adaptive	SMC2-DS	0	100	0	0	0
			SMC2-PS	0	100	0	0	0

T: The number of distributions.

Proposal: The proposal scales of the MCMC kernels.

Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.

Table 5.7 Nonlinear ODE model marginal likelihood and the Bayes factor estimates with data generated from a complex (five components) model.

				Marginal likelihood $\log p(\boldsymbol{y}\|\mathcal{M}_k) \pm$ S.D.		Bayes factor $\log B_{3,5} \pm$ S.D.
T	Proposal	Annealing	Algorithm	$r = 3$	$r = 5$	
10	Manual	Prior (5)	PMCMC	−1651.0±27.9	−85.1 ± 36.6	1565.9 ± 42.1
30				*-1639.7±7.4*	*-78.9±11.2*	*1560.8±12.8*
100				−1624.6±15.7	−75.7 ± 24.8	1548.9 ± 25.6
500	Manual	Prior (5)	SMC2-DS	−1640.7±10.8	−78.5 ± 9.8	1562.2 ± 10.1
			SMC2-PS	−1640.8 ± 8.4	−79.2 ± 7.9	1561.6 ± 8.5
500	Manual	Adaptive	SMC2-DS	−1639.7 ± 6.9	−78.6 ± 4.8	1561.1 ± 7.1
			SMC2-PS	−1640.1 ± 5.4	−78.8 ± 3.7	1561.3 ± 6.8
500	Adaptive	Adaptive	SMC2-DS	−1639.8 ± 2.2	−79.4 ± 1.7	1560.4 ± 3.1
			SMC2-PS	***-1640.2±1.9***	***-78.5±1.5***	***1561.7±2.3***

T: The number of distributions.

Proposal: The proposal scales of the MCMC kernels.

Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.

Red: The estimate with the smallest variance among all algorithms settings.

Green: The estimate with the smallest variance for a single algorithm (SMC2 or PMCMC) among different settings.

Table 5.8 Frequencies of models selected by the Bayes factor (%) for the nonlinear ODE model with data generated from a complex (five components) model.

				Number of components				
T	Proposal	Annealing	Algorithm	2	3	4	5	6
10	Manual	Prior (5)	PMCMC	0	0	6	94	0
30				0	0	2	98	0
100				0	0	0	100	0
500	Manual	Prior (5)	SMC2-DS	0	0	0	100	0
			SMC2-PS	0	0	0	100	0
500	Manual	Adaptive	SMC2-DS	0	0	0	100	0
			SMC2-PS	0	0	0	100	0
500	Adaptive	Adaptive	SMC2-DS	0	0	0	100	0
			SMC2-PS	0	0	0	100	0

T: The number of distributions.
Proposal: The proposal scales of the MCMC kernels.
Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.

5.5.3 PET *compartmental model*

It is now interesting to compare the proposed algorithm with other state-of-art algorithms using the more realistic PET compartmental model example.

As mentioned before, real neuroscience data sets involve a very large number (about a quarter of a million per brain) of time series, which are typically somewhat heterogeneous (also see Figure 5.1). Robustness is therefore especially important. An application-specific MCMC algorithm was developed for this problem in [167] and its results are shown in the Section 4.3.7.1. A significant amount of tuning of the algorithms was required to obtain good results. The results shown in Figure 5.8 and discussed later are very close to those of [167] but, as is shown later, they were obtained with almost no manual tuning effort and at similar computational cost.

For the SMC and population MCMC algorithms, the requirement of robustness means that the algorithm must be able to calibrate itself automatically to different data (and thus different posterior surfaces). A sequence of distributions which performs well for one time series may not perform even adequately for another series. Specification of proposal scales that produces fast-mixing kernels for one data series may lead to slow mixing for another (as we already see in Section 5.3.3.) In the following experiment, we will use the simulated data sets, and choose schedules that perform both well and poorly for this particular time series. The objective is to see if the algorithm can recover from a relatively poorly specified schedule and obtain reasonably accurate results.

Results

In this example we focus on the comparison between SMC2 and population MCMC. We also consider parallelized implementations of algorithms. In this case, due to its relatively small number of chains, population MCMC can be parallelized completely (and often cannot fully utilize the hardware capability if a naïve approach to parallelization is taken; while we appreciate that more sophisticated parallelization

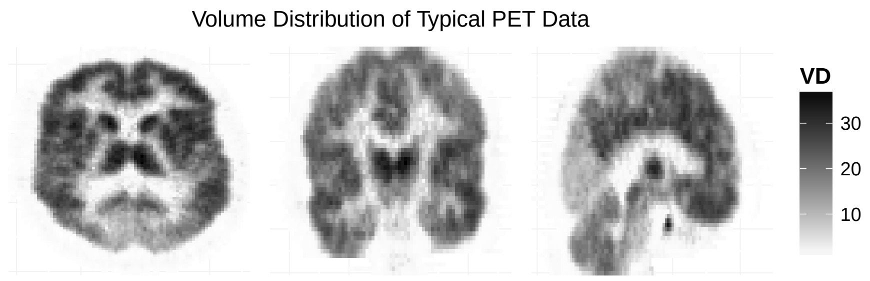


Figure 5.8 Volume of distribution estimates of real PET compartmental model data. Three slice of the brain are shown in the plot. Each are close to the middle of the brain along each of the three axises in the three-dimensional space.

strategies are possible, these depend intrinsically upon the model under investigation and the hardware employed and given our focus on automatic and general algorithms, we don't consider such strategies here.) Population MCMC algorithm under this setting is implemented such that each chain is updated at each iteration. Further, for the SMC algorithms, we consider two cases. In the first we can parallelize the algorithm completely (in the sense that each core has a single particle associated with it.) In this setting we use a relatively small number of particles and a larger number of time steps. In the second, we need a few passes to process a large number of particles at each time step, and accordingly we use fewer time steps to maintain the same total computation time. These two settings allow us to investigate the trade-off between the number of particles and time steps. In both implementations, we consider three schedules, $\alpha(t/T) = t/T$ (linear), $\alpha(t/T) = (t/T)^5$ (prior), and $\alpha(t/T) = 1 - (1 - t/T)^5$ (posterior). The linear schedule can be seen as an off-the-shelf choice while the prior schedule is expected to perform generally well for many applications. The posterior schedule is expected to perform poorly compared to the others. It places more distributions close to the posterior than the prior, where with the introducing of the likelihood function, the intermediate distributions are likely to change more dramatically with respect to the change of α. This is included there to test if the algorithm is capable of producing sensible results when the sequence of distributions is specified poorly, which is quite a possible scenario in realistic applications. In addition, the adaptive schedule based upon CESS is also implemented for the SMC2 algorithm. For the SMC algorithms, stratified resampling is applied.

Result from 100 replicate runs of the two algorithms under various regimes can be found in Table 5.9 and 5.10 for the marginal likelihood and Bayes factor estimates, respectively. The SMC algorithms consistently outperforms population MCMC algorithms in the parallel settings. The Monte Carlo standard deviation of SMC algorithms is typically of the order of one fifth of the corresponding estimates from population MCMC in most scenarios. In some settings with the smaller number of samples, the two algorithms can be comparable. Also at the lowest computational costs, the samplers with more time steps and fewer particles outperform those with the converse configuration by a fairly large margin in terms of estimator variance. It shows that with limited resources, ensuring the similarity of consecutive distributions, and thus good mixing, can be more beneficial than a larger number of particles. However, when the computational budget is increased, the difference becomes negligible.

In the particular case of the posterior schedule, it is not surprising that all path sampling estimates suffer considerably large biases. The standard estimator using the SMC2 algorithm is able to give relatively accurate results (though with a larger variance compared to other schedules). In summary, the SMC algorithm is much more robust than population MCMC algorithm in this example.

Effects of adaptive schedule A set of samplers with adaptive schedules are also used. Due to the nature of the schedule, it cannot be controlled to have exactly the same number of time steps as non-adaptive procedures. However, the CESS was controlled such that the average number of time steps are comparable with the fixed schedules and in most cases slightly less than the fixed numbers.

It is found that, with little computational overhead, adaptive schedules do provide the best results (or very nearly so) and do so without user intervention.

Table 5.9 Marginal likelihood estimates of two components PET model.

Proposal scales			Manual			Adaptive
Annealing scheme			Prior (5)	Posterior (5)	Adaptive	
T	N	Algorithm	Marginal likelihood estimates ($\log p(\theta_k \mid \boldsymbol{y}) \pm$ S.D.)			
500	30	PMCMC	−39.1 ± 0.56	−926.8 ± 376.99		
500	192	SMC 2-DS	*-39.2±0.25*	*-39.7±1.06*	*-39.2±0.18*	***-39.1±0.12***
		SMC 2-PS	*-39.2±0.25*	−91.3 ± 21.69	*-39.2±0.18*	−39.1 ± 0.13
100	960	SMC 2-DS	−39.3 ± 0.36	−40.6 ± 1.41	−39.2 ± 0.31	−39.2 ± 0.19
		SMC 2-PS	−39.3 ± 0.35	302.1 ± 46.29	−39.3 ± 0.31	−39.2 ± 0.18
1000	30	PMCMC	−39.3 ± 0.46	−884.1 ± 307.88		
1000	192	SMC 2-DS	*-39.2±0.19*	*-39.4±0.68*	*-39.2±0.17*	***-39.1±0.10***
		SMC 2-PS	*-39.2±0.19*	−66.0 ± 13.26	*-39.2±0.17*	***-39.1±0.10***
200	960	SMC 2-DS	−39.2 ± 0.22	−39.8 ± 1.21	−39.2 ± 0.18	−39.1 ± 0.11
		SMC 2-PS	−39.2 ± 0.22	175.5 ± 26.84	−39.2 ± 0.18	−39.2 ± 0.11
2000	30	PMCMC	−39.3 ± 0.28	−928.7 ± 204.93		
2000	192	SMC 2-DS	−39.2 ± 0.14	*-39.3±0.41*	−39.1 ± 0.12	−39.1 ± 0.07
		SMC 2-PS	−39.2 ± 0.14	−51.2 ± 4.30	−39.2 ± 0.12	−39.1 ± 0.07
400	960	SMC 2-DS	*-39.2±0.13*	−39.4 ± 0.73	*-39.2±0.11*	−39.2 ± 0.07
		SMC 2-PS	*-39.2±0.13*	106.0 ± 14.36	*-39.2±0.11*	***-39.2±0.06***
5000	30	PMCMC	−39.3 ± 0.21	−917.6 ± 129.54		
5000	192	SMC 2-DS	−39.2 ± 0.09	*-39.2±0.20*	−39.2 ± 0.08	−39.1 ± 0.04
		SMC 2-PS	−39.2 ± 0.09	−43.8 ± 2.13	−39.2 ± 0.08	−39.1 ± 0.04
1000	960	SMC 2-DS	*-39.2±0.08*	−39.2 ± 0.31	*-39.2±0.07*	***-39.2±0.03***
		SMC 2-PS	*-39.2±0.08*	−65.7 ± 5.54	*-39.2±0.07*	***-39.2±0.03***

T: The number of distributions.

N: The number of particles.

Proposal: The proposal scales of the MCMC kernels.

Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.

Red: The estimate with the smallest variance among all algorithms settings.

Green: The estimate with the smallest variance for a single algorithm (SMC2 or PMCMC) among different settings.

Table 5.10 The Bayes factor estimates of two-compartments PET model.

Proposal scales			Manual			Adaptive
Annealing scheme			Prior (5)	Posterior (5)	Adaptive	
T	N	Algorithm	Bayes factor estimates ($\log B_{2,1} \pm$ S.D.)			
500	30	PMCMC	1.7 ± 0.62	−70.9 ± 525.79		
500	192	SMC 2-DS	*1.6±0.27*	*1.3±1.13*	*1.6±0.20*	***1.6±0.15***
		SMC 2-PS	*1.6±0.27*	−3.9 ± 30.02	*1.6±0.20*	***1.6±0.15***
100	960	SMC 2-DS	1.6 ± 0.37	0.5 ± 1.55	1.6 ± 0.34	1.6 ± 0.21
		SMC 2-PS	1.6 ± 0.37	−13.1 ± 66.30	1.6 ± 0.33	1.6 ± 0.21
1000	30	PMCMC	1.6 ± 0.49	−67.3 ± 400.21		
1000	192	SMC 2-DS	*1.6±0.21*	*1.5±0.79*	1.6 ± 0.20	1.6 ± 0.13
		SMC 2-PS	*1.6±0.21*	−0.6 ± 15.47	1.6 ± 0.20	1.6 ± 0.13
200	960	SMC 2-DS	1.6 ± 0.25	1.1 ± 1.25	1.6 ± 0.19	1.6 ± 0.12
		SMC 2-PS	1.6 ± 0.24	−11.7 ± 34.68	*1.6±0.18*	***1.6±0.11***
2000	30	PMCMC	1.6 ± 0.31	−95.5 ± 264.74		
2000	192	SMC 2-DS	*1.6±0.14*	*1.6±0.44*	1.6 ± 0.13	1.6 ± 0.09
		SMC 2-PS	*1.6±0.14*	1.6 ± 6.06	1.6 ± 0.13	1.7 ± 0.09
400	960	SMC 2-DS	1.6 ± 0.16	1.5 ± 0.74	*1.6±0.12*	***1.6±0.08***
		SMC 2-PS	1.6 ± 0.16	−4.2 ± 17.15	*1.6±0.12*	***1.6±0.08***
5000	30	PMCMC	1.6 ± 0.24	−60.3 ± 198.10		
5000	192	SMC 2-DS	1.6 ± 0.10	*1.6±0.23*	1.6 ± 0.09	1.6 ± 0.05
		SMC 2-PS	1.6 ± 0.10	1.3 ± 2.98	1.6 ± 0.09	1.6 ± 0.05
1000	960	SMC 2-DS	*1.6±0.09*	1.6 ± 0.33	*1.6±0.08*	***1.6±0.04***
		SMC 2-PS	*1.6±0.09*	−0.2 ± 6.63	*1.6±0.08*	***1.6±0.04***

T: The number of distributions.

N: The number of particles.

Proposal: The proposal scales of the MCMC kernels.

Annealing: The annealing scheme of the distributions, $\alpha_k(t/T_k)$ in Equation 5.24.

Red: The estimate with the smallest variance among all algorithms settings.

Green: The estimate with the smallest variance for a single algorithm (SMC2 or PMCMC) among different settings.

The reduction of Monte Carlo standard deviation varies among different configurations. For moderate or larger number of distributions, a reduction about 50% was observed. In addition, it should be noted that, in this example, the bias of the path sampling estimates are much more sensitive to the schedules than the previous Gaussian mixture model example. A vanilla linear schedule does not provide a low bias estimator at all even when the number of distributions is increased to a considerably larger number. The prior schedule though provides a nearly unbiased estimator, there is no clear theoretical evidence showing that this should work for other situations. Even it has more general usage, as suggested in [24], the power still has to be chosen (in the previous GMM example, $p = 2$ was the best choice while in this PET example $p = 5$ is more suitable). In contrast, The adaptive schedule, without any manual calibration, can provide a nearly unbiased estimator, even when path-sampling is employed, in addition to potential variance reduction.

Bias reduction for path sampling estimator As seen in Table 5.9 and 5.10, a bad choice of schedule $\alpha(t/T)$ can results in considerable bias for the basic path sampling estimator, here for SMC2-PS but the problem is independent of the mechanism by which the samples are obtained. Increasing the number of iterations can reduce this bias but at the cost of additional computation time. As outlined in Section 5.3.1, in the case of the SMC algorithms discussed here, it is possible to reduce the bias without increasing computational cost significantly. To demonstrate the bias reduction effect, we constructed SMC sampler for the above PET example with only 1,000 particles and about 20 iterations specified using the CESS based adaptive strategy. The path sampling estimator was approximated using Equation (5.28) as well as other higher order numerical integration or by integrating over a grid that contains $\{\alpha_t\}$ at which the samples was generated. The results are shown in Table 5.11

Table 5.11 Path sampling marginal likelihood estimates bias reduction for a simulated PET data set.

	Number of grid points (compared to sampled iterations)			
Integration rule	$\times 1$	$\times 2$	$\times 4$	$\times 8$
Trapezoid	-52.2 ± 5.01	-45.5 ± 1.93	-42.1 ± 1.21	-40.5 ± 1.06
Simpson	-43.2 ± 1.39	-41.0 ± 1.10	-40.0 ± 1.04	-39.4 ± 1.04
Simpson 3/8	-42.1 ± 1.21	-40.5 ± 1.06	-39.7 ± 1.04	-39.3 ± 1.04
Boole	-40.9 ± 1.09	-39.9 ± 1.04	-39.4 ± 1.04	-39.2 ± 1.05

The estimator ($p(\boldsymbol{y}|\theta_k, \mathcal{M}_k) \pm$ S.D.) was approximated using samples from SMC2 algorithm with 1,000 particles and 20 iterations, with different numerical integration strategies. Large sample result (see Table 5.9) shows that the accurate value is about -39.2.

Trade-off between the number of particles and distributions It can be seen through the nonlinear ODE and the PET examples that, there is a trade-off between the number of particles and distributions. Increasing either of them can improve the accuracy of estimates. We consider a range of number of particles (from 100 to more than 10,000) and a range of number of distributions using the prior schedule ($\alpha(t/T) = (t/T)^5$; from as small as 10 to more than 1,000.) When applied to the simulated data sets, the variance of the path sampling estimates is plotted against the total number of samples (the product of these two quantities) in Figure 5.9. It can be seen that, for the same total number of samples, samplers with larger number of distributions outperform those with larger number of particles by a considerable large margin. However, as the number of samples increase, the difference becomes smaller and smaller. This suggests that it will be better to first allocate a fixed number of particles, which is at least large enough to approximate the initial distribution well, according to considerations such as computation hardwares. And then use the number of distributions as a performance parameter to tune the sampler for desired accuracy. When using the adaptive algorithms proposed in this work, it is equivalent to tune the value of CESS*. As shown in Section 5.3.2, the relation between CESS*

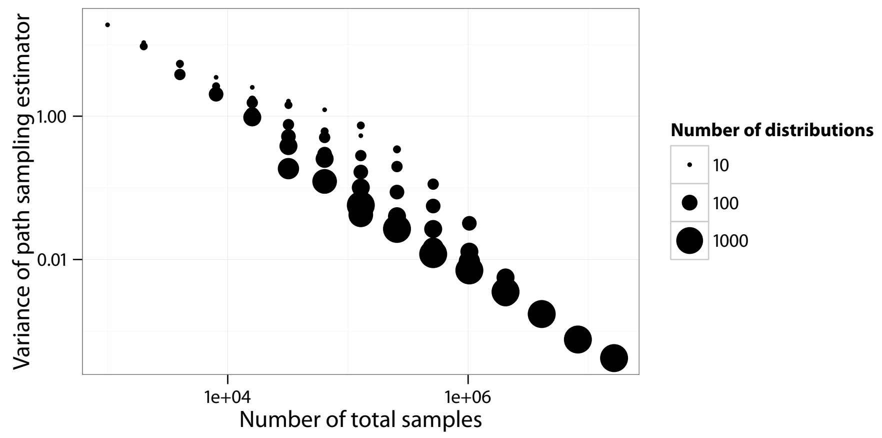


Figure 5.9 Variance of path sampling estimator and total number of samples using the SMC2 algorithm. Multiple samplers are used to evluate the trade-off between the number of particles N and the number of distributions T. They are configured such that the total number of samples NT is a constant. In this figure, the variance of the path sampling estimator from 100 simulations of each sampler is plotted against the total number of samples NT on the logarithm scale. Different configurations of T are indicated with different sizes of the dots, larger dots representing larger T (and thus smaller N). It can be seen that for a particular value of NT, changing T may change the variance considerably and larger T is preferred for most values of NT.

and the variance of estimators provides a predictable way to configure the samplers, at least for the particular examples considered in this chapter.

Fast mixing MCMC kernels and number of distributions As mentioned in Section 5.1.5, the suboptimal backward kernel and its associated incremental weights used in the above examples could perform poorly if the adjacent distributions are not close, even when the transition kernel mixes well. However, both fast mixing kernels and more intermediate distributions (and thus a smoother sequence), can improve the performance of the sampler. To improve the mixing speed of the kernel,

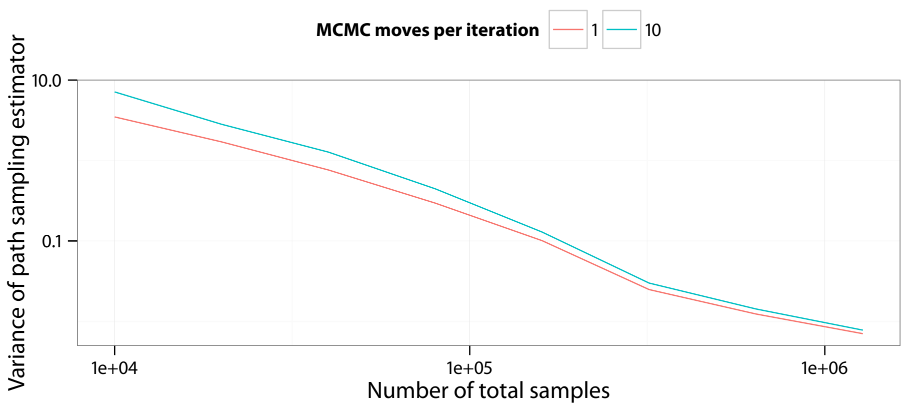


Figure 5.10 Variance of the path sampling estimates and total number of samples using the SMC2 algorithm (on logarithm scale). The variances are calculated from 100 simulations for each sampler configuration. All samplers use 1,000 particles but with different number of distributions T and passes of MCMC moves at each iteration K. And TK is the number of total samples generated. Samplers with the same value of TK have roughly the same computational cost.

one can apply multiple passes of MCMC moves at each iteration. Here we compare two samplers using the simulated data sets, both with 1,000 particles. One sampler uses 10 MCMC moves at each iteration. The other only use one MCMC move but ten times the number of distributions. The annealing scheme, for simplicity, is chosen to be $\alpha(t/T) = (t/T)^5$. The results of the path sampling variance is shown in Figure 5.10. It can be seen that, given the same total number of samples simulated, using more distributions almost always outperforms using more MCMC moves. However, with a sufficiently large number of samples, the difference is minimal.

Real data results Finally, the methodology of SMC2-PS was applied to measured positron emission tomography data using the same compartmental setup as in the simulations. The data that lead to the V_D estimation as shown in Figure 5.8 comes from a study into opioid receptor density in Epilepsy, with the data being described in detail in [89] (also see section 2.3). It is expected that there will be considerable spatial smoothness to the estimates of the volume of distribution, as this is in line with the biology of the system being somewhat regional. Some regions will have much higher receptor density while others will be much lower, yielding higher and lower values of the volume of distribution, respectively. While we did not impose any spatial smoothness but rather estimated the parameters independently for each time series at each spatial location, as can be seen, smooth spatial estimates of the volume of distribution consistent with neurological understanding were found using the approach. This method is computationally feasible for the entire brain on a voxel-by-voxel basis, due to the ease of parallelization of the SMC algorithm. In the analysis performed here 1,000 particles were used, along with an adaptive schedule using a constant $\text{CESS}^\star/N = 0.999$, resulting in about 180 to 200 intermediate distributions. The model selection results are very close to those obtained by a previous study of the same data [167], although the present approach requires much less implementation effort and has roughly the same computational cost.

5.5.4 *Summary*

These three illustrative applications have essentially shown three aspects of using SMC as a generic tool for Bayesian model selection. Firstly, as seen in the Gaussian mixture model example, all the different variants of SMC proposed, including both direct and path sampling versions, produce results which are competitive with other model selection methods such as RJMCMC and population MCMC. In addition, in this somewhat simple example, SMC2 performs well, and leads to low variance estimates with no appreciable bias. The effect of adaptation was studied more carefully in the nonlinear ODE example, and it was shown that using both adaptive selection of distributions as well as adaptive proposal variances leads to very competitive algorithms, even against those with significant manual tuning. This suggests that an automatic process of model selection using SMC2 is possible. In the final example, considering the easy parallelization of algorithms such as SMC2 suggests that great gains in variance estimation can be made using settings such as GPU computing for application where computational resources are of particular importance (such as in image analysis as in the PET example). It is also clear that the negligible cost of the bias reduction techniques described means that one should always consider using these to reduce the bias inherent in path sampling estimation.

5.6 DISCUSSIONS

It has been shown that SMC is an effective Monte Carlo method for Bayesian inference for the purpose of model comparison. Three approaches have been outlined and investigated in several illustrative applications including the challenging scenarios of nonlinear ODE models and PET compartmental systems. The proposed strategy is always competitive and often substantially outperforms the state of the art in this area.

It has been demonstrated that it is possible to use the SMC algorithms to estimate the model probabilities directly (SMC1), or through individual model evidence (SMC2), or pair-wise relative evidence (SMC3). In addition, both SMC2 and SMC3 algorithms can be coupled with the path sampling estimator.

Among the three approaches, SMC1 is applicable to very general settings. It can provide a robust alternative to RJMCMC when inference on a countable collection of models is required (and could be readily combined with the approach of [85] at the expense of a little additional implementation effort). However, like all Monte Carlo methods involving between model moves, it can be difficult to design efficient algorithms in practice. The SMC3 algorithm is conceptually appealing. However, the existence of a suitable sequence of distributions between two posterior distributions may not be obvious.

The SMC2 algorithm, which only involves within-model simulation, is most straightforward to implement in many interesting problems. It has been shown to be exceedingly robust in many settings. As it depends largely upon a collection of within-model MCMC moves, any existing MCMC algorithms can be reused in the SMC2 framework. However, much less tuning is required because the algorithm is fundamentally less sensitive to the mixing of the Markov kernel and it is possible to implement effective adaptive strategies at little computational cost. With adaptive placement of the intermediate distributions and specification of the MCMC kernel proposals, they provide a robust and essentially automatic model comparison method.

Compared to population MCMC, SMC2 has greater flexibility in the specification of distributions. Unlike population MCMC, where the number and placement of distributions can affect the mixing speed and hence performance considerably, increasing the number of distributions will always benefit an SMC sampler given the same number of particles. When coupled with a path sampling estimator, this leads to less bias and variance. Compared to its no-resampling variant (i.e., AIC), it has been shown that SMC samplers with resampling can reduce the variance of normalizing constant estimates considerably.

Even after three decades of intensive development, no Monte Carlo method can solve the Bayesian model comparison problem completely automatically without any manual tuning. However, SMC algorithms and the adaptive strategies demonstrated in this chapter show that even for realistic, interesting problems, these samplers can provide good results with very minimal tuning and few design difficulties. For many applications, they could already be used as near automatic, robust solutions. For more challenging problems, the robustness of the algorithms can serve as solid foundation for specific algorithm designs.

6 vSMC: A C++ LIBRARY FOR PARALLEL SMC

The vSMC library was developed during the research to assist the implementations of various SMC algorithms, including but not limited to the illustrative and performance comparison examples in previous chapters. It evolves into a sophisticated C++ framework. Generic SMC samplers and other related algorithms can be implemented using the library with relative ease. In addition, the library supports building both sequential and parallel programs using the same user implementation of a given algorithm.

The library makes use of some modern C++ techniques, ranging from basic object-oriented programming to template metaprogramming. One should not need to be an expert on most of them to use the library. Most of the examples in this chapter are self-explanatory to readers with some basic knowledge of C++. For those interested, Appendix C.3 serves as a brief introduction to C++ templates and callable objects, two elementary features used extensively in the interface of the vSMC library.

Section 6.1 gives background on parallel computing and the state of softwares for Monte Carlo computing. Section 6.2 provides an overview of the library and the structure of a program written with the library that implements a generic SMC sampler. Section 6.3 to 6.6 discuss the implementations of the four main components of a generic SMC sampler: the particle system, initializing, updating and monitoring a sampler. Parallelization is not difficult with vSMC. However, it could be an unfamiliar subject to some. Therefore, instead of a more technical discussion, we introduce this feature through an example in Section 6.4 and demonstrate it by examples throughout this chapter. In Section 6.7, we use a realistic example to show the performance of the library. In the same section, the productivity of the library, considering the performance gain, is also discussed. This chapter is concluded by a discussion of techniques introduced and future directions.

6.1 background

6.1.1 *Parallel computing*

Parallel computing is a form of computation in which many calculations are carried out simultaneously. It operates on the principle that large problems can be divided into independent smaller ones and can be solved concurrently ("in parallel"). Each smaller problem is solved by an individual *computing unit*, also called an *worker*. They can be individual cores in a multicore processor or nodes in a cluster. In the remaining of this chapter, we will use these two terms interchangeably.

Parallelism has been practiced for many years in the form of high performance computing. In recent years, it has also become the dominant paradigm for desktop computing in the form of multicore processors. However, many of today's popular statistical softwares are written with serialization in mind; and they do not easily take advantage of contemporary computer architectures. A discussion on the commonly used parallel computers can be found in Appendix c.1. Though it is possible to obtain superior performance by using hardware specific features, we are more concerned with providing a generic solution that can be used by non-experts while offering better performance through parallelization.

Parallelism strategies

The best overall strategy for *scalable parallelism* is *data parallelism* [73]. There are various definitions of data parallelism. Narrower definitions only permit collection-oriented operations, such as applying the same function to all elements in an array. A wider view is that the parallelism grows (preferably linearly) as the data size or the problem size grows. For example, parallelizing a vanilla Monte Carlo integration algorithm belongs to this strategy. As the number of samples increases, one can always use more parallel computing resource to run the sampler with the same amount of time without increasing the speed of each computing unit. Note that, here we ignored issues such as generating random numbers in parallel, which will be discussed later, and other factors that may slow down the performance when the number of parallel computing units increases beyond certain limit. The SMC algorithms can also use increased parallelism in a similar fashion by increasing the number of particles to improve accuracy of estimators. In contrast, an MCMC algorithm usually cannot be parallelized in a scalable way. To obtain better statistical results, often the only way is to increase the number of iterations, and thus no matter how much parallel computational resources are available, the computing time will increase without increasing the speed of the processors. There are also algorithms that fall between the two extremes. For example, the population MCMC algorithm can be parallelized. And the parallelism grows linearly as the number of distributions grows. However, unlike SMC algorithms, increased parallelism does not always lead to better accuracy of estimators.

The opposite of data parallelism is *functional parallelism*, an approach that runs different functional parts of a program in parallel. At best, functional parallelism can improve performance by a constant speedup. For example, say a program performs functions $f_1, \dots, f_k$, then at best the computing time can be reduced by k-fold through parallelism. In the remainder of this chapter, we focus on data parallelism.

For specific problems, there are various design patterns to parallelize the computation. Interested readers can read Appendix C.2 for a discussion on this topic.

Importance of parallel computing

Parallel computers have been developed for decades. Several reasons have led to an increased level of parallel computing in individual, mainstream personal computers.

The most significant one is the hardware trend. From 1973 to 2003, clock rates of processors increased from 1 MHz to 1 GHz. Since then there have been little improvement on this front. Now most high-end workstations have processors with clock rates at around 2.5 GHz. However, virtually all processors produced now have multiple cores [113]. Eight to twelve cores configurations are common in middle to high-end workstations and personal computers often have at least two cores with quadric configurations more and more commonly seen. The clock rates did not only increase significantly, but also has the trend of decreasing. These changes are due to various technical difficulties in increasing the clock rates among other reasons, which we will not elaborate further here.

Scientists are ever seeking to solve more complex problems, which often require more computations. The only way to solve larger problems without using significantly longer computing time in the foreseeable future, is to use parallelism. Parallel computing is also much more economically efficient in both power consumption and processors' production than sequential computing [113]. In reality, it means researchers can invest the same or less amount of funding, yet get more computational work done with the same or less amount of time.

Performance measurement

Unlike sequential computing, the performance of parallel computing is more difficult to study. In sequential situations, the computational cost can often be deduced from the algorithms easily. For example, a Monte Carlo algorithm can use the total number of samples to be generated as a measure of its computational cost. However, in the case of parallel computing, the total amount of computation, either measured as the number of arithmetic operations or data operations, cannot reflect the cost in reality. This is due to the fact that, today's parallel computers are much more cost efficient when more work is parallelized [113] and parallelization has its own overhead cost. In practice, instead of the total amount of computation, one is more interested in the speedup of a parallel program, defined as the ratio between computing time of a sequential program and the one of a parallel program that does the same amount of computational work.

Let P be the number of hardware workers, e.g., cores in a multicore processor or nodes in a cluster, and T_P be the total time of computation. T_1 is usually called the *work* of the program and T_∞ is called the *span*. The speedup, defined as $S_P = T_1/T_P$, is upper bounded,

$$S_P \le \frac{T_1}{T_1/P} = P \tag{6.1}$$

In addition, assuming that adding processors never slows down the program (in reality, it is only the case when P is modest),

$$S_P \le T_1/T_\infty \tag{6.2}$$

Implementations on different hardwares often concern one of the three quantities, T_1, T_P and T_∞. For sequential implementations, clearly T_1 is the only one of interest. For multicore and SMP systems, T_P is of interest for a particular value P, and attaining a speedup of P ($T_P = T_1/P$) is desired. In the context of SMC algorithms, it is often the case that $T_\infty \ll T_P$ and thus T_∞ (attaining the second upper bound of speedup) was previously of less interest. However, the recent development on massive parallel computers (see Appendix C.1) has made it close to a reality for many algorithms to attain a speedup $1/T_\infty$. In this form of parallel computing, there are often hundreds or even thousands parallel computing units working concurrently. For many applications, this effectively means that all computational work can be parallelized as long as the algorithm permits.

Limitations

Parallel computing is not without drawbacks. Two main factors that limit its widespread use in practice is the difficulty in reasoning of the program and the tuning of performance.

Parallel programs are more difficult to construct *correctly* than an equivalent sequential program. Because of its parallel nature, many operations may be performed concurrently and may happen at random orders or at the same time. However, due to reasons such as data dependency, some operations have to be performed in a deterministic order to give meaningful results. Informally, when two workers try to modify the same location of data, the behavior is undefined. This is also called *data race*.

Another difficulty of parallel computing is the tuning and portability of performance. Since the development of languages such as Fortran and C, scientists have relied on them to develop portable softwares. There are two sides of portability. One is the programming portability, meaning that the same source code can be used to build softwares for different platforms with little or no modifications. The other one is the performance portability, meaning that the softwares built from the same source code for different platforms have comparable performance. In the early days, people needed to optimize programs for each platform individually. However, with the development of modern compiler techniques, such practices are much less seen.

In the era of parallel computing, many low level details need to be taken care of to obtain reasonable performance. For example, while using the OpenMP programming model, which is widely used by scientists to write parallel programs, issues such as thread affinity (associate each thread with a particular processor) can often cause large performance differences. More recently, devices such as GPUs are even less performance portable. The author has seen that the same OpenCL [101] program can have an order of magnitude difference in performance when running on devices from different vendors even though they have similar raw computational power.

We believe that with the development of developer tools for parallel computing, such issues will become less common and it will help the wider spread of parallel computing.

The last but not least problem with parallel computing is that, not all algorithms can be parallelized or at least not efficiently. Many MCMC algorithms are typical examples. And they can hardly benefit much from future computer technology advancement. This issue can be better solved by developing new algorithms that are more suitable for today's and future computers.

6.1.2 *Software for Monte Carlo computing*

Over the decades there are many softwares developed for the purpose of Monte Carlo computing, especially for more established algorithms such as MCMC. It is impossible to give a complete review of even those most important ones here. Most of them can be characterized by three aspects – *application areas*, *software environments* and *implementation level*.

Some softwares are designed with general application in mind. They can be used to solve a large array of problems. Some others target specific applications and some of them are designed to implement a particular model.

There are also the differences in software environments. Some are standalone software. They are often the easiest to use. Others depend on a larger software environment. For example, many numerical tools are developed using MatLab [157]. In recent years, the R programming language [129] has gain substantial popularity among statisticians. These softwares often require at least basic knowledge of the environment (e.g., MatLab or R) to use. There are also softwares developed for low level languages such as C++, distributed in the form of libraries. They may have an even steeper learning curve.

By *implementation level*, we mean how much of a given algorithm is implemented by the software and how much is left to be implemented by users. This is closely related to the application area of the software. At the lowest level, some softwares used to solve a particular problem requires the user to implement algorithms from the ground up. The softwares themselves only provide basic facilities such as linear algebra computations. Some softwares provide frameworks on top of which users only need to fill in problem specific information.

In the following, we review some more important development for Monte Carlo computing. Most of those that are relevant to the work in this thesis can be categorized by whether they solve MCMC or SMC problems.

Software for MCMC computing

The most well known software for using MCMC algorithms is perhaps BUGS [148, 110]. It provides an easy to use environment for Bayesian modeling using the Gibbs sampling. Users only need to specify the models using the BUGS model specification language, which describes the model parameters in a direct acyclic graph (DAG) and the data in a similar language. The software analyzes the model and chooses MCMC algorithms to do the sampling. It is an easy to use practical tool for Bayesian analysis. The output of BUGS is usually analyzed with R, using packages such as CODA [128].

The limitation of BUGS is that the resulting algorithms may not be well tuned and it can take a long time to get reasonable results. In particular, it is very difficult for users to provide input to the algorithm design process. The software chooses the algorithm tuning parameters, such as the proposal scales for Metropolis random walk algorithms. Even if users have insights such as what values of the proposal scales are more likely to lead to good performance, it is difficult for such insights to be used by the software. In addition, it does not allow more flexible design of data structures and thus it can be significantly inefficient for some applications.

There are also many packages for the R environment that implement MCMC algorithms. The MCMCpack [112] package provides model specific MCMC algorithms for a wide range of models commonly used in social and behavioral science. The mcmc [60] package can be used to implement Metropolis random walk algorithms for continuous distributions. There are also many more application specific packages. For example, the R project's task view on Bayesian inference[1] lists dozens of packages among which many implement MCMC algorithms for specific models. These packages are often very useful in their application areas but with less generality.

[1]`http://cran.r-project.org/web/views/Bayesian.html`

Another interesting development is PyMC [46]. It is distributed as a module for the Python programming language. It is highly influenced by BUGS while providing more flexibility and better performance through the integration with Python. The CppBugs [12] is similar to PyMC but is a library for C++. It is possible to obtain much better performance using CppBugs compared to BUGS while only a little more programming effort is required. Both PyMC and CppBugs give users access to a general programming language in the process of designing the algorithms. Much flexibility and better performance are gained through this integration with a general purpose programming language. On the other hand, to use them to their full potential, users do need to have some experiences of the programming languages.

Overall, for MCMC algorithms, it is often not very difficult to develop application specific software using a general purpose programming language such as R or C++ coupled with a suitable package or library. These widely used softwares tend to solve a particular class of problems instead of providing a more general framework for implementation of algorithms. Bayesian inference is certainly one of the more important application areas of MCMC. And many softwares have been developed for this purpose, with BUGS being perhaps the most influential one and relatively more general than others such as the various R packages targeting specific models.

Software for SMC computing

Unlike MCMC algorithms, even the simplest SMC algorithms can be difficult to implement in general purpose programming languages for researchers, as the implementations of resampling and other aspects of the algorithms are not always straightforward. There is a demand for softwares that help researchers to implement complex generic SMC algorithm.

Many SMC algorithms are often more computational intense than typical MCMC applications. This can be partially attributed to the fact that SMC algorithms are often used to simulate complex high dimensional distributions, for which MCMC algorithms often perform poorly. Therefore, applications of interest for SMC algorithms requires highly efficient implementations.

There is relatively little development of softwares for SMC algorithms. There has been some development using MatLab for the purpose of particle filtering, such as, the PFLib [27] toolbox.

More recently, the SMCTC library [90] was developed. It provides a framework for implementation of generic SMC algorithms in C++. It is possible to implement many realistic algorithms using the library with relative ease. It also provides very good performance. The generic framework was built with C++ template techniques. It allows a wide range of applications while requires some expertise in C++. These are two traits also shared by the vSMC library.

There are also a few R packages that provides implementations of SMC algorithms. For example the smc [62] package can be used to implement some generic SMC algorithms. However it is more of a skeleton of SMC algorithms with much of the implementation details such as resampling needed to be provided by users. Overall, R packages for SMC algorithms are much less common than those for MCMC algorithms.

As SMC algorithms gain more attentions in research areas, such as Bayesian inference, some application specific softwares have been also developed. The BiiPS [25] package aims to provide users an interface similar to that of BUGS with SMC as the underlying algorithms for inference instead of MCMC. It is built on top of SMCTC among other softwares. It can be used as a drop-in replacement of BUGS in many applications.

Another interesting development is the LibBi library [117]. It is particularly suited for Bayesian state-space modeling. It provides an easy to use interface using the Perl programming language. One does not need to know much of the language to use LibBi's interface. However, proficiency in Perl allows much flexibility in the design of the algorithm. This is similar to the PyMC module for MCMC algorithms. The library can also construct parallelized sampler for a wide range of hardware.

Parallelized Monte Carlo computing

Parallel computing can be used to accelerate Monte Carlo applications. However, due to its very sequential natural, MCMC algorithms have seen little development on this front. All softwares for MCMC computing discussed before are built with sequential implementations.

Driven by the need to simulate complex distributions efficiently and the desire to use parallel computing to solve larger problems, many algorithms that are particularly suitable for parallelization have been developed in the past decade. The SMC and related algorithms are clearly among them. The population MCMC (see Section 4.3.5) algorithm can also be parallelized though less efficiently. For example, see results in Section 5.5.2 and 5.5.3. More recently the particle MCMC algorithm [7] is also well suited for parallel computing.

There is no lack of interest in using parallel computing for these algorithms. For example, [104] studied the implementation of SMC algorithms on massive-parallel hardware (e.g., GPU). The results are encouraging. However, there are few softwares for the purpose of implementation of generic SMC algorithms in parallel computers. The LibBi library is a notable exception.

There is also some fundamental work done in this area. One more important aspect is generating random numbers in parallel. Conventional pseudo-RNG generates random numbers using an internal state, say x_t and iterates it with a deterministic transformation, $x_{t+1} = f(x_t)$. A data dependency exists between x_{t+1} and x_t, which prevents scalable parallelization. For example, algorithms in [104] used to generate random numbers have a cost greater than $O(N)$ where N is the number of parallel computing units. One solution to this problem is using state-less RNG. Informally, given a collection of values $\{x_i\}_{i=1}^N$, the collection $\{y_i\}_{i=1}^N$ where $y_i = f(x_i)$, appears to be random. There are no dependencies among $\{x_i\}_{i=1}^N$. Therefore the collection $\{y_i\}_{i=1}^N$ can be generated in parallel efficiently. The work by [139] provides accessible, high performance state-less RNG. It is also used by the vSMC library.

We believe there is demand of softwares similar to SMCTC, which provides a framework for implementation of generic SMC algorithms (in contrast to applicable for only a class of models) and for taking full advantages of today's parallel computers. The vSMC library aims to fill this gap. It is less easy to use than softwares such as LibBi or BiiPS. But it is possible to use it to obtain more flexibility in the design of the algorithm and better performance.

6.2 THE vSMC LIBRARY

To obtain and install the library, see detailed instructions in [166], which also documents the third-party dependencies and compiler support. A Doxygen [72] generated reference manual can be found at `http://zhouyan.github.io/vSMC/doc/html/index.html`. It is beyond the scope of this chapter to document every feature of the vSMC library. In many places we will refer to this reference manual for further information.

A more systematic tutorial of the library can be found in [166] and the reference manual. The remainder of this chapter is structured according to the common tasks performed by generic SMC samplers. Many features of the library are introduced in examples. Interested readers can see the tutorial [166] and the reference manual for details.

The vSMC library makes use of C++'s template generic programming to implement general SMC algorithms. The library is formed by a few major modules. In the remainder of this chapter, unless stated otherwise, all public classes and functions of the library reside in the namespace `vsmc`. Brief discussions of the most important modules are discussed below.

Core The highest level of abstraction of SMC samplers. Users interact with classes defined within this module to create and manipulate general SMC samplers. Classes in this module include `Sampler`, `Particle` and others. These classes use user defined callback to perform application specific operations, such as updating particle values and weights.

Symmetric Multiprocessing (SMP) This is the form of computing most people use everyday, including multiprocessor workstations, multicore desktops and laptops. Classes within this module make it possible to write generic operations which manipulate a single particle that can be applied either sequentially or in parallel through various parallel programming models. A method defined through classes of this module can be used by `Sampler` as callback objects.

Message Passing Interface MPI is the *de facto* standard for parallel programming on distributed memory architectures. This module enables users to adapt implementations of algorithms written for the SMP module such that the same sampler can be parallelized using MPI. In addition, when used with the SMP module, it allows easy implementation of hybrid parallelization such as MPI/OpenMP.

OpenCL This module is similar to the two above except it eases the parallelization through OpenCL, such as for the purpose of General Purpose GPU Programming (GPGPU). OpenCL is a framework for writing programs that can be executed across heterogeneous platforms. OpenCL programs can run on either CPUs or GPUs.

6.2.1 *Core classes*

There are over two hundred classes, large and small, in the vSMC library. It is beyond the scope of this chapter to document most of them. However, a few of them play central roles in the implementation of SMC algorithms. In this section, we provide an overview of them. Many of them are feature rich. Instead of documenting their interfaces here, we will introduce useful features through examples later.

Value collection type This is actually not a type defined by vSMC, but a user defined class that abstracts the collection of values $\{X^{(i)}\}_{i=1}^N$. The library allows much flexibility in the definition of this type. The important thing to note here is that this class needs to at least abstract the whole collection of all values instead of a single particle. In Section 6.3, we introduce a readily usable implementation provided by the vSMC library, on top of which users can build application specific classes.

Most core classes in the library are class templates with this value collection type as their template parameter. In the following, we use the generic name `T` to denote this value collection type.

Sampler A `Sampler<T>` object is used to execute various operations of an SMC algorithm. It is used to initialize the particles and to update them. It is also used to perform resampling and importance sampling approximation. In the body of a program, this is usually the only class that users need to interact with. In Section 6.2.2, we show how each step of a generic SMC algorithm is mapped to the operations provided by a `Sampler<T>` object.

Particles A `Sampler<T>` object contains, among other things, an object of type `Particle<T>` that abstracts the particle system. A particle system is formed by both the values $\{X^{(i)}\}_{i=1}^N$ and the importance weights $\{W^{(i)}\}_{i=1}^N$. The former is abstracted by user defined value collection type `T`. The later is abstracted by a `WeightSet<T>` object.

The `Particle<T>` object also provides various methods that manipulate the particle system, for example, it can perform the resampling algorithm on the particle system when required by the `Sampler<T>` object.

Weights As said, the importance weights in a particle system are manipulated through a sub-object of type `WeightSet<T>`. In addition to common weights manipulations, such as setting the weights directly or using the incremental weights, it also provides ways to query properties of the weights. For example, it can calculate the ESS and CESS values.

For most applications, the default `WeightSet<T>` is sufficient. However, like the value collection type, there could be special requirement of this class. It can be replaced by user defined classes through C++ template metaprogramming. The details are documented in the reference manual.

Monitors Given a real-valued function h, the library can use `Monitor<T>` type objects to compute the importance sampling approximation of $\mathbb{E}[h(X)]$ automatically as the sampler progresses. The function value of h is allowed to be a vector. And it is possible to use optimized linear algebra library to accelerate the computation in that case. There are also special support for path sampling, which requires essentially a simple importance sampling approximation and a numerical integration.

6.2.2 *Program structure*

Recall the SMC algorithms discussed in Section 5.1, regardless of specific applications or algorithm settings, in practice they can be dissembled into the following steps.

1. Initialize values $\{X^{(i)}\}_{i=1}^N$ and calculate importance weights $\{W^{(i)}\}_{i=1}^N$.
2. For $t = 1, \dots, T$, where T may not be finite (for example, a particle filter processing incoming data on-line), repeat
   (a) Update either values $\{X^{(i)}\}_{i=1}^N$ or importance weights $\{W^{(i)}\}_{i=1}^N$ or both.
   (b) Resampling.
   (c) Update either values $\{X^{(i)}\}_{i=1}^N$ or importance weights $\{W^{(i)}\}_{i=1}^N$ or both.

Note that steps 2.(a)-(c) are all optional, though it is unlikely that all three of them are absent. For example, an AIS algorithm does not have the resampling step. An

SMC algorithm such as Algorithm 5.2 only updates the weights before the possible resampling and only update the values after it while a particle filter might update both the values and weights at step 2.(a). Both step 2.(a) and 2.(c) may be formed by a few sub-steps. For example, the Markov kernel may be constructed as a composition of multiple Metropolis random walks.

In addition, after each iteration of step 2, we may be interested to evaluate some importance sampling estimates. For example, the path sampling estimator, as seen in Section 5.2.4, requires the importance sampling estimates of $\mathrm{d}\log\gamma_\alpha(X)/\,\mathrm{d}\alpha$ where γ_α is the unnormalized density function of the family of distributions that the SMC sampler operates on. Another example is particle filters, which often requires estimates of certain parameters at each iteration.

For demonstration purpose, let us assume that our program has all those steps and need to calculate both the path sampling and other importance sampling estimates. In the vSMC library, all these tasks are performed through the `Sampler` class. Below is an example of such a program,

```
int main ()
{
    Sampler<T> sampler(N, Stratified, 0.5);

    sampler.init(init_f);             // Step 1.
    sampler.move(move_f, false);      // Step 2.(a)
    sampler.mcmc(mcmc_f1, true);      // Step 2.(c)
    sampler.mcmc(mcmc_f2, true);      // Step 2.(c)

    // Path sampling
    sampler.path_sampling(path_eval);

    // Importance sampling estimates of moments
    sampler.monitor("moments", 2, moments_eval);
    // Runing the algorithm
    sampler.initialize(param);
    sampler.iterate(IterNum);

    // Results of path sampling
    std::cout << "Path sampling: " << std::endl;
    std::cout << sampler.path_sampling() << std::endl;

    // Output importance sampling approximation
    // and other aspects of the history of the sampler
    std::ofstream output("sampler_file");
    output << sampler << std::endl;
    output.close();

    return 0;
}
```

We will explain each line of this program in detail. For now, it is sufficient to point out that the following objects used in this program are user defined callback that implement application specific operations.

`init_f` Initialize the particle values. (Section 6.4)

`move_f` Update the particles. For example, updating the weights. These updates are performed before the possible resampling. (Section 6.5)

`mcmc_f1` and `mcmc_f2` Update the particles. For example, moving the particles with an MCMC kernel. These updates are performed after the possible resampling. (Section 6.5)

`path_eval` Evaluate the value of path sampling integrands, $\mathrm{d}\log\gamma_\alpha(X)/\,\mathrm{d}\alpha$. (Section 6.6)

`moments_eval` Evaluate importance sampling estimate integrands, for example moments of parameters. (Section 6.6)

6.3 THE PARTICLE SYSTEM

At the core of each implementation of SMC algorithms using the vSMC library is the definition of the value collection type that abstracts $\{X^{(i)}\}_{i=1}^{N}$. vSMC does not restrict how the values should be actually stored. They can be stored in the main memory, spread among nodes of a cluster, in GPU memory or even in a database. Users can define their own value collection type to fulfill various application specific needs. For full details on the requirement of the value collection type, see [166].

Given a value collection type `T`, one can construct a sampler,

```
Sampler<T> sampler(N, Stratified, 0.5);
```

The first argument is the number of particles. The second is the resampling methods. There are six built-in resampling schemes in the library. And user defined resampling algorithms can also be used. See the reference manual for details. The last argument is the threshold of ESS$/N$ at each iteration, below which resampling will be performed. The later two parameters are optional.

A `Sampler<T>` object has a sub-object, `Particle<T>`, which contains the type `T` object along with other data such as the importance weights. Each can be accessed as the following,

```
Sampler<T> sampler(N);
sampler.particle();         // Reference to Particle<T> object
sampler.particle().value(); // Reference to type T object
```

6.3.1 *A matrix of state values*

Many typical problems' value collections can be viewed as a matrix of certain type. For example, a simple particle filter whose state is a real-valued vector of length M can be viewed as an N by M matrix of type `double` where N is the number of particles. A trans-dimensional problem (e.g., [85]) can use an N by 1 matrix whose type is a user defined class, say `StateType`. For this kind of problems, a class template is provided by the library,

```
template <MatrixOrder Order, std::size_t Dim, typename StateType>
class StateMatrix;
```

The first template parameter (possible value `RowMajor` or `ColMajor`) specifies how the values are ordered in memory. Usually one should choose `RowMajor` to optimize data access. The second template parameter is the number of variables, an integer value no less than 1 or the special value `Dynamic`, in which case `StateMatrix` provides a member function `resize_dim` such that the number of variables can be changed at runtime. The third template parameter is the type of the state values. Each particle's state is thus a vector of length `Dim`, indexed from `0` to `Dim - 1`. To obtain the value at position `j` of the vector of particle `i` (the element at the `i`th raw and `j`th column of the matrix), one can use the `state` member function,

```
StateBase<RowMajor, Dim, StateType> value(N);
StateType val = value.state(i, j);
```

There are other ways to obtain and manipulate the values, see the reference manual for details. Note that, one can derive from the `StateMatrix` class to extend its functionality, as we will see in examples later.

6.3.2 *A single particle*

If the value collection type `T` satisfies certain requirements[2], then for a `Particle<T>` object, one can construct a `SingleParticle<T>` object that abstracts one of the particle from the collection. For example,

[2]See the reference manual for technique details. It is sufficient to note here that `StateMatrix` and any of its derived classes satisfy those requirements.

```
Particle<T> particle(N);
SingleParticle<T> (i, &particle);
```

create a `SingleParticle<T>` object corresponding to the particle `i`. There are a few member functions of `SingleParticle<T>` that make access to individual particles easier than through the interface of `Particle<T>`. For instance, for each particle, a `Particle<T>` object construct an independent C++11 RNG engine. For example, the following uses it to generate standard Normal random variates,

```
std::normal_distribution<double> rnorm(0, 1);
std::vector<double> z(particle.size());
for (std::size_t i = 0; i != particle.size(); ++i)
    z[i] = rnorm(particle.rng(i));
```

If we access each particle through `SingleParticle<T>`, then we can write

```
z[i] = rnorm(sp.rng());
```

Here `sp.rng()` is equivalent to `particle.rng(i)`.

The functionality of a `SingleParticle<T>` can be enhanced through template metaprogramming. For instance, if `T` is `StateMatrix` or its derived class, then `sp.state(j)` is equivalent to `particle.value().state(i, j)`.

6.3.3 *Example: The value collection of* GMM

The `StateMatrix` is a minimalistic class template. Users can derive from it and build application specific value collection classes. Here we demonstrate how the value collection, named `GMM`, in the SMC2 algorithm for the Gaussian mixture model (GMM; see Section 5.5.1) is designed.

Recall that, a GMM with r components has a parameter vector of length $3r$, $\theta_r = (\mu_{1:r}, \lambda_{1:r}, \omega_{1:r})$. In the SMC2 algorithm, we use the sequence of distributions $\{\pi_t\}_{t=0}^T$ taking the form,

$$\pi_t(\theta_t) = \pi_0(\theta_t|\mathcal{M})p(\boldsymbol{y}|\theta_t, \mathcal{M})^{\alpha(t/T)}.$$

The `GMM` class will abstract the GMM in addition to the state of all particle values at any given generation. Therefore, we have the following design goals for this class,

1. The data, which is associated with the model should be stored in and can be accessed through this class.
2. The calculation of the likelihood and the prior densities, which are characteristics of the model should be possible through this class.
3. The distribution specification parameter α and the MCMC proposal scales, which are properties of a given generation of the particle system, should be associated with this class.

This class is outlined as below.

```
template <std::size_t R>
class GMM :
    public StateMatrix<RawMajor, 3 * R, double>
{
    public :

    GMM (std::size_t N) :
        StateMatrix<RawMajor, 3 * R, double>(N),
        mu_scale_(1), lambda_scale_(1), omega_scale_(1) {}

    static const std::size_t ComponentNumber = R;

    static std::size_t mu_idx (std::size_t j)
    { return j; }

    static std::size_t lambda_idx (std::size_t j)
    { return R + j; }

    static std::size_t omega_idx (std::size_t j)
    { return R * 2 + j; }

    double mu_scale () const { return mu_scale_; }
    double &mu_scale () {return mu_scale_; }

    double lambda_scale () const { return lambda_scale_; }
    double &lambda_scale () {return lambda_scale_; }

    double omega_scale () const { return omega_scale_; }
    double &omega_scale () {return omega_scale_; }

    double alpha (std::size_t t) const;
    double &alpha (std::size_t t);

    void read_data (const std::string &data_file);

    double log_likelihood (std::size_t i) const;
    double log_prior (std::size_t i) const;

    private :

    double mu_scale_, lambda_scale_, omega_scale_;
    std::vector<double> alpha_;
    std::vector<double> data_;
};
```

First the number of components is set through a template parameter `R`. Of course, it is possible to make this parameter dynamic and changeable at runtime by using `Dynamic` for the second template parameter of `StateMatrix`.

Second, the static member functions `mu_idx`, etc., returns the index of the μ_j, etc., in each row of the `StateMatrix`. For example, to access λ_j of the i^{th} particle, we can use

```
particle.value().state(i, GMM<R>::lambda_idx(j));
```

or with the `SingleParticle` interface

```
sp.state(GMM<R>::lambda_idx(j));
```

instead of the much more difficult to read expression,

```
sp.state(GMM<R>::ComponentNumber * 2 + j);
```

It is trivial to see that the parameters are arranged as if in such a matrix,

$$\begin{pmatrix} \theta_r^{(1)} \\ \vdots \\ \theta_r^{(N)} \end{pmatrix} = \begin{pmatrix} \mu_1^{(1)}, \dots, \mu_r^{(1)}, & \lambda_1^{(1)}, \dots, \lambda_r^{(1)}, & \omega_1^{(1)}, \dots, \omega_r^{(1)} \\ \vdots & \vdots & \vdots \\ \mu_1^{(N)}, \dots, \mu_r^{(N)}, & \lambda_1^{(N)}, \dots, \lambda_r^{(N)}, & \omega_1^{(N)}, \dots, \omega_r^{(N)} \end{pmatrix}$$

Third, the setter and getter member functions such as `mu_scale` provide access to the proposal scales. In addition, the member function `alpha` provides access to $\alpha(t/T)$.

Fourth, the `read_data` member function, whose definition is omitted here, provides a way to read data into the `data_` member data.

And last, the `log_likelihood` and `log_prior` member functions calculate the log-likelihood and log-prior densities for a given particle. They accept the particle's index number as input. The actual implementations of these functions, distributed with the library, use more sophisticated data structures to ensure that the computation only occurs when the parameter values are changed. From a user's perspective, one only needs to know that these member functions will return the value of likelihood and prior densities for the current particle values when called, while the actual computation may or may not happen when the functions are called.

6.4 INITIALIZING

The particles are initialized by a user defined callback. The callable object has the following signature,

```
std::size_t init_f (Particle<T> &, void *);
```

It is added to the sampler by

```
sampler.init(init_f);
```

And it will be called when the following in the program (Section 6.2.2) is executed,

```
sampler.initialize(param);
```

where the input parameter `param` is optional and the default value is `NULL`. It will be passed on as the second argument of `init_f` with `sampler.particle()` being the first. The return value of `init_f` will be recorded as the acceptance count and can be later retrieved by,

```
sampler.accept_history(0, 0);
```

The optional parameter can be used to provide additional information needed to initialize the sampler.

If users do not do anything special, the sampler will also initialize the weights $\{W^{(i)}\}_{i=1}^{N}$ to be equal and normalized to $1/N$. In addition, any information recorded for previous generations of the particle system will be erased during the initialization.

6.4.1 *Example: Simulation of a Normal distribution*

We show here a very simple example, simulation of Normal random variables. And we will introduce an important feature of the library through it – parallelization.

Suppose for an SMC algorithm with a parameter vector of length k, we want to initialize each of the parameter to $\mathcal{N}(\mu, \sigma^2)$, where $\mathcal{N}$ denotes the Normal distribution. We can implement it as the following,

```
// Definition of constants: N, K

typedef StateMatrix<RawMajor, K, double> T;

struct Param { double mean; double sd; };

std::size_t init_f (Particle<T> &particle, void *param)
{
    const Param *p = static_cast<const Param *>(param);
    std::normal_distribution<double> rnorm(p->mean, p->sd);
    for (std::size_t i = 0; i != particle.size(); ++i) {
        for (std::size_t k = 0; k != K; ++k) {
            particle.value().state(i, k) =
                rnorm(particle.rng(i));
        }
    }
}
```

In this example, we used the second parameter `param` to pass information about the Normal distribution, its mean and standard deviation. In the body of the program (the `main` function), we can use it as the following,

```
sampler.init(init_f);

Param param = {Mean, Sd};
sampler.initialize(param);
```

6.4.2 *Parallelized implementation*

In the above example, we looped over all particles. The inner loop is repeated for each particle. There are no data dependencies among particles in this operation. It is perfectly reasonable to have the outer loop parallelized.

This kind of parallelization, not only for initializing particles, but also for updating particles, are supported in the vSMC library through a set of class templates. Here we introduce the ones specific to initialization,

```
template <typename T, typename D = Virtual> class InitializeSEQ;
template <typename T, typename D = Virtual> class InitializeOMP;
template <typename T, typename D = Virtual> class InitializeTBB;
```

Each of the above three implement sequential, OpenMP parallelization and Intel TBB parallelization, respectively. There are a few other similar classes for other parallel programming models not listed here. We first use `InitializeSEQ` as an example to demonstrate how it is used. The interface of `InitializeSEQ` given the second template parameter being `Virtual` is,

```
template <typename T, typename D = Virtual>
class InitializeSEQ
{
    virtual std::size_t initialize_state (SingleParticle<T>) = 0;
    virtual void initialize_param (Particle<T> &, void *);
    virtual void pre_processor (Particle<T> &);
    virtual void post_processor (Particle<T> &);

    std::size_t operator() (Particle<T>, void *);
};
```

The existence of the *non-virtual* member function `operator()` and the form of its signature ensures that an object of its derived class can be used just as `init_f`. It is implemented as if,

```
std::size_t operator() (Particle<T> &particle, void *param)
{
    this->initialize_param(particle, param);
    this->pre_processor(particle);
    std::size_t acc = 0;
    for (std::size_t i = 0; i != particle.size(); ++i) {
        acc += this->initialize_state(
                SingleParticle<T>(i, particle));
    }
    this->post_processor(particle);

    return acc;
}
```

Different class templates listed above differ at how they implement the loop. For example, `InitializeOMP` uses OpenMP to parallelize this loop.

The user can derive from this class and use the virtual functions to provide application specific behaviors of this operator. For example, the simulation of Normal random variates can now be re-implemented as,

```
class init_c : public InitializeSEQ<T>
{
    public :

    init_c () : mean_(0), sd_(1) {}

    std::size_t initialize_state (SingleParticle<T> sp)
    {
        std::normal_distribution<double> rnorm(mean_, sd_);
        for (std::size_t k = 0; k != K; ++k)
            sp.state(k) = rnorm(sp.rng());
    }

    void initialize_param (Particle<T> &, void *param)
    {
        const Param *p = static_cast<const Param *>(param);
        mean_ = p->mean;
        sd_ = p->sd;
    }

    private :

    double mean_, sd_;
};

// In the main function, replace
// sampler.init(init_f); with
sampler.init(init_c());
```

At a first glance, it takes quite a few more lines than the original implementation of `init_f`. However, by replacing `InitializeSEQ` with `InitializeOMP`, without changing anything else, the sampler will be using OpenMP for parallelization during the initialization step.

There are also other benefits of this implementation. First, if OpenMP is not available in users' C++ environment (e.g., using the popular Clang [156] compiler), one can use the same implementation with other parallel programming models. For instance, to use Intel TBB instead of OpenMP, only `InitializeOMP` needs to be changed to `InitializeTBB`.

Second, this implementation is also scalable. A few changes allows it to use MPI for parallelization on distributed memory computers. All that needs to be done is to wrap the value collection type with the adapter class `StateMPI`,

```
typedef StateMPI<StateMatrix<RawMajor, K, double> > T
```

In summary, with almost identical implementations, we can build programs running on single threaded sequential mode, on multicore processors with various parallel programming models or on a distributed memory computer with MPI.

6.5 UPDATING

The addition of methods that update the particles is more flexible than initialization. There are two kinds of updating methods. One is simply called `move` in vSMC, and is performed before the possible resampling at each iteration. The other is called `mcmc`, and is performed after the possible resampling. They are often MCMC type moves. Multiple `move`s or `mcmc`s are also allowed. In fact a vSMC sampler consists of a queue of `move`s and a queue of `mcmc`s.

All these are implemented using user defined callbacks similar to the `init_f` function in the last section, with a slight different signature,

```
std::size_t move_f (std::size_t, Particle<T> &);
```

This is the same for both `move`'s and `mcmc`'s. The first argument is the iteration number, counting from zero for the initialization step. The second argument is passed by the sampler, which is `sampler.particle()`.

To add `move_f` into the queue of `move`'s, call

```
sampler.move(move_f, false);
```

The second argument, a boolean value, indicates whether the new `move` should be appended to the existing (possibly empty) queue (if it is set to `false`); or the queue should be cleared before set a new one. The queue of the `mcmc`'s is manipulated similarly.

6.5.1 *Example: Updating the weights in the SMC2 algorithm*

Recall Algorithm 5.2, the updating of weights should be performed before possible resampling at each iteration. And the change of the weights are calculated with the incremental weights,

$$W_t^{(i)} \propto W_{t-1}^{(i)} w_t(\theta_{t-1}^{(i)}, \theta_t^{(i)})$$
$$w_t(\theta_{t-1}^{(i)}, \theta_t^{(i)}) = p(\boldsymbol{y}|\theta_{t-1}^{(i)}, \mathcal{M})^{\alpha(t/T)-\alpha([t-1]/T)}.$$

This is quite generic for different applications. All we need here is the calculation of the log-likelihood function. It is natural to write a function template for it,

```
template <typename T>
std::size_t smc_move (std::size_t iter, Particle<T> &particle)
{
    // Calculate $\alpha_{\mathrm{diff}} = \alpha(t/T) - \alpha([t-1]/T)$
    const double alph_diff =
        particle.value().alpha(iter) -
        particle.value().alpha(iter - 1);

    // Calculate the logarithm of the incremental weights
    // $\tilde{w}_t = \alpha_{\mathrm{diff}} \log p(\boldsymbol{y}|\theta_t, \mathcal{M})$
    std::vector<double> log_inc_w(particle.size());
    for (std::size_t i = 0; i != particle.size(); ++i) {
        log_inc_w[i] =
            alpha_diff * particle.value().log_likelihood(i);
    }

    particle.weight_set().add_log_w(log_inc_weight.begin());

    return 0;
}

// in main function
sampler.move(smc_move<T>, false);
```

The assumption about the value collection type `T` is,

1. It provides access to $\alpha(t/T)$ in the same way as the `GMM` class in Section 6.3.3.
2. It provides access to the log-likelihood in the same way as the `GMM` class.

The first part of the function template `smc_move` calculates $\alpha(t/T) - \alpha([t-1]/T)$. The second part calculates the logarithm of the incremental weights for each particle. The last part manipulates the weights.

Weights are manipulated through a object of type `WeightSet`. There are other ways to manipulate them, such as,

```
std::vector<double> weight(particle.size());
particle.weight_set().set_equal_weight();
particle.weight_set().set_weight(weight.begin());
particle.weight_set().mul_weight(weight.begin());
particle.weight_set().set_log_weight(weight.begin());
particle.weight_set().add_log_weight(weight.begin());
```

The `set_equal_weight` member function sets all weights to be equal, i.e., $1/N$. The `set_weight` and `set_log_weight` member functions set the values of weights and logarithm weights, respectively. The `mul_weight` and `add_log_weight` member functions multiply the weights or add to the logarithm weights by the given values, respectively. All these member functions accept general input iterators as their arguments.

6.5.2 *Example: The MCMC move in GMM*

In this example, we show an implementation of the MCMC moves in the GMM example (Section 5.5.1). We will only detail the implementation of random walk block on $\mu_{1:r}$. The others are similar. Recall that, we perform a Normal random walk on the mean parameters. The MCMC algorithm's implementation can be summarized as the following steps,

1. Calculate the value of the target density for the parameter values, say f.

2. Propose new values according to the proposal distribution. In our implementation, this proposal step are carried in place, meaning that the particle values are updated when new values are proposed.
3. Calculate the value of the target density for the proposed parameter values, say f'.
4. Generate a uniform random variate on the $[0, 1]$ interval, say u,
5. Accept the proposed values if $u < f'/f$. Otherwise, restore the old values.

The implementation of these five steps are straightforward,

```
template <std::size_t R>
class GMM_MCMC_Mu : public MoveOMP<GMM<R> >
{
    public :

    std::size_t move_state (
            std::size_t iter, SingleParticle<GMM<R> > sp)
    {
        // Step 1.
        double log_target =
            sp.particle().value().log_likelihood(sp.id()) +
            sp.particle().value().log_prior(sp.id());

        // Step 2.
        double backup[R];
        std::normal_distribution<double> rnorm(
                0, sp.particle().value().mu_scale());
        for (std::size_t j = 0; j != R; ++j) {
            std::size_t mu_idx = GMM<R>::mu_idx(j);
            backup[mu_idx] = sp.state(mu_idx);
            sp.state(mu_idx) += rnorm(sp.rng());
        }

        // Step 3.
        double log_target_proposed =
            sp.particle().value().log_likelihood(sp.id()) +
            sp.particle().value().log_prior(sp.id());

        // Step 4.
        std::uniform_distribution<double> runif(0, 1);
        double log_u = std::log(runif(sp.rng()));

        // Step 5.
        double log_prob = log_target_proposed - log_target;
        if (log_u > log_prob) {
            for (std::size_t j = 0; j != R; ++j) {
                std::size_t mu_idx = GMM<R>::mu_idx(j);
                sp.state(mu_idx) = backup[mu_idx];
            }
            return 0;
        }
        return 1;
    }
};
```

First, we derived our class from a class template called `MoveOMP`. It is similar to the `InitializeSEQ` class template introduced in Section 6.4.2. It provides OpenMP parallelization.

Second, we used a few new features of the `SingleParticle` class template. Recall that, it is created from a reference to a `Particle<T>` object and an index of the individual particle. The `Particle<T>` object can be obtained, by a constant reference, through

```
sp.particle();
```

and the index can be obtained through

```
sp.id();
```

These are used in the calculation of the values of the log-likelihood and the log-prior densities.

Otherwise, the implementation is a straightforward translation of the mathematical representation of the algorithm. The whole algorithm has three blocks of random walks. Say we implemented the other two similarly as `GMM_MCMC_Lamba` and `GMM_MCMCM_Omega`, then in the body of the program we can add them to the sampler by,

```
sampler
    .mcmc(GMM_MCMC_Mu(), false)
    .mcmc(GMM_MCMC_Lambda(), true)
    .mcmc(GMM_MCMC_Omega(), true);
```

Note that the `mcmc` member function call return a reference the sampler itself. Therefore we can chain these calls. When we call,

```
sampler.iterate(IterNum);
```

the sampler will iterate `IterNum` steps and at each step, all three of these random walks will be applied to the particle system.

6.6 MONITORING

Before initializing the sampler or after a certain time point, one can add monitors to the sampler. The concept is similar to BUGS's `monitor` statement, except it does not monitor the individual values but rather the importance sampling estimates.

Consider approximating $\mathbb{E}[h(X)]$, where $h(X) = (h_1(X), \dots, h_m(X))$ is an m-vector function. The importance sampling estimate can be obtained by AW where A is an N by m matrix where $A(i, j) = h_j(X^{(i)})$ and $W = (W^{(i)}, \dots, W^{(N)})^T$ is the N-vector of normalized weights. To compute this importance sampling estimate, one needs to define the following evaluation function (or other kinds of callable objects),

```
void monitor_eval (std::size_t iter, std::size_t m,
        const Particle<T> &particle, double *res);
```

and adds it to the sampler by calling,

```
sampler.monitor("variable.name", m, monitor_eval);
```

When the function `monitor_eval` is called, `iter` is the iteration number of the sampler, `m` has the same value as the one users passed to `Sampler<T>::monitor`; and thus one does not need global variables or other similar techniques to access this value. The output pointer `res` points to an $N \times m$ output array of row major order. That is, after the calling of the function, the value of `res[i * dim + j]` should be $h_j(X^{(i)})$.

Implementation of the path sampling estimator (Section 5.2.4) can be viewed as a special kind of monitor. In addition to the evaluation of $h(X^{(i)})$, where $h(X) = \mathrm{d}\log\gamma_\alpha(X)/\,\mathrm{d}\alpha$ in this special case, the interval length of the numerical integration as in Equation (5.28) also needs to be obtained. The vSMC library provides special support for path sampling. First one needs to define a function, say `path_eval` with the following signature,

```
double path_eval (std::size_t iter,
        const Particle<T> &particle, double *res);
```

It is not unlike the `monitor_eval` function above. It only differs by,

1. The output array `res` is always of length N, and thus there is no argument to pass the dimension of the monitor
2. It returns a value, which should be α_t.

To use it, one can add it to the sampler by,

```
sampler.path_sampling(path_eval);
```

And the estimate can be obtained by

```
sampler.path_sampling();
```

6.6.1 *Example: Path sampling in the SMC2 algorithm*

In Algorithm 5.2, the path sampling integrands is simply the log-likelihood, and $\alpha_t = \alpha(t/T)$. Therefore the implementation of a generic `path_eval` is straightforward. Again, we assume that the value collection type T provides access to $\alpha(t/T)$ and the log-likelihood in the same way as the `GMM` class. We can implement the function template as the following,

```
double path_eval (std::size_t iter,
        const Particle<T> &particle, double *res)
{
    for (std::size_t i = 0; i != particle.size(); ++i)
        res[i] = particle.value().log_likelihood(i);

    return particle.value().alpha(iter);
}
```

Other SMC algorithms such Algorithm 5.3 may have different path sampling estimator expressions. But the implementation is similar.

6.6.2 *Example: Adaptive specification of proposal scales*

Now we demonstrate the implementation of a slightly more complex monitor and the use of it for the purpose of adaptive specification of proposal scales. Consider the GMM example, as outlined in Section 5.3.3. We can use the moments of the parameters to set the proposal scales adaptively.

First, we need to create a monitor that records the first two raw moments of each parameter. We can make this problem more general as estimating the first M raw moments. We use a parallelized monitor, `MonitorEvalOMP` for the implementation of the evaluation function. It is not unlike the `MoveOMP` class template introduced earlier. The main difference is that now we need to define the following member function,

```
void monitor_state(std::size_t, std::size_t,
        ConstSingleParticle<GMM<R> > csp, double *res)
```

where the first argument is the iteration number and the second is the dimension of the monitor. The third, a `ConstSingleParticle` type object, is similar to `SingleParticle` except that now one does not have write access to the particles. In other words, one cannot change the particle values through it. The last, the output parameter `res` is of length the dimension of the monitor. The function call needs only to store the values of $h(X^{(i)})$ for a single particle. The complete implementation is given as below,

```
template <std::size_t R, std::size_t M>
class GMM_Moments : MonitorEvalOMP<GMM<R> >
{
    public :

    static const std::size_t Dim = R * Order * 3;

    static std::size_t mu_idx (std::size_t j, std::size_t m)
    { return j * Order + (m - 1); }

    static std::size_t lambda_idx (std::size_t j, std::size_t m)
    { return j * Order + (m - 1) + R * Order; }
```

```
static std::size_t omega_idx (std::size_t j, std::size_t m)
{ return j * Order + (m - 1) + R * Order * 2; }

static double mu_scale (const double *res);
static double lambda_scale (const double *res);
static double omega_scale (const double *res);

void monitor_state(std::size_t, std::size_t,
        ConstSingleParticle<GMM<R> > csp, double *res)
{
    // Record the first raw moment
    for (std::size_t j = 0; j != R; ++j) {
        res[mu_idx(j, 1)] = csp.state(GMM<R>::mu_idx(j));
        res[lambda_idx(j, 1)] = csp.state(GMM<R>::lambda_idx(j));
        res[omega_idx(j, 1)] = csp.state(GMM<R>::omega_idx(j));
    }

    // Record the second and up to Order $M$ raw moments
    // Using the simple recursion $X^m = X^1 X^{m-1}$
    for (std::size_t m = 2; m <= Order; ++m) {
        for (std::size_t j = 0; j != R; ++j) {
            res[mu_idx(j, m)] =
                res[mu_idx(j, 1)] * res[mu_idx(j, m - 1)];
            res[lambda_idx(j, m)] =
                res[lambda_idx(j, 1)] * res[lambda_idx(j, m - 1)];
            res[omega_idx(j, m)] =
                res[omega_idx(j, 1)] * res[omega_idx(j, m - 1)];
        }
    }
    }
};
```

In addition to the monitor_state member function, we also provide a few utilities in this class. First, similar to the GMM class, we use functions to return the index of a given order of moment for a specific parameter of a certain components inside the output parameter res. This makes the implementation more readable. Second, we also provide functions, whose definitions are trivial and not shown here, that calculate the proposal scales for each random walk given an array of moments estimates. We can add this monitor to the sampler by,

```
sampler.monitor("moments", GMM_Moments<R, 2>::Dim,
        GMM_Moments<R, 2>());
```

We choose only to estimate the first two raw moments.

Now we only need a method to set the proposal scales using this monitor. Since this should be set before the updating of weights and possible resampling, we can construct a move for this job.

```
template <std::size_t R>
class GMM_AdaptiveScale
{
    public :

    GMM_AdaptiveScale (const Sampler<GMM<R> > *sampler_ptr) :
        sampler_ptr_(sampler_ptr) {}

    std::size_t operator() (std::size_t, Particle<GMM<R> > &particle)
    {
        double res[GMM_Moments<R, 2>::Dim];
        for (std::size_t j = 0; j != GMM_Moments<R, 2>::Dim; ++j)
            res[j] = sampler_ptr_->monitor("moments").record(j);
        particle.value().mu_scale() =
            GMM_Moments<R, 2>::mu_scale(res);
        particle.value().lambda_scale() =
            GMM_Moments<R, 2>::lambda_scale(res);
        particle.value().omega_scale() =
            GMM_Moments<R, 2>::omega_scale(res);
    }

    private :

    const Sampler<GMM<R> > *sampler_ptr_;
}
```

And in the `main` function, we change the move queue to

```
sampler.move(GMM_AdaptiveScale(&sampler), false);
sampler.move(smc_move<GMM<R> >, true);
```

Note that, we initialize the move with a pointer of the sampler itself, and use this pointer to access the record in the monitor named `"moments"`. There are many ways to retrieve the importance sampling estimates from a monitor. The one we used here is

```
res[i] = sampler_ptr_->monitor("moments").record(j);
```

which returns the importance sampling estimate of $\mathbb{E}[h_j(X)]$ for the latest generation of the particle system.

6.7 PERFORMANCE

One of the main motivation behind the creation of vSMC is to ease the parallelization with different programming models. The same implementation can be used to build different samplers based on what kinds of parallel programming models are supported on users' platforms. In this section we compare the performance of various SMP parallel programming models and OpenCL parallelization. We use the GMM with SMC2 algorithm as shown in Section 5.5.1 for benchmarking. Many major parts of its implementation have been shown through this chapter. For a complete documentation on its implementation with vSMC, see [166].

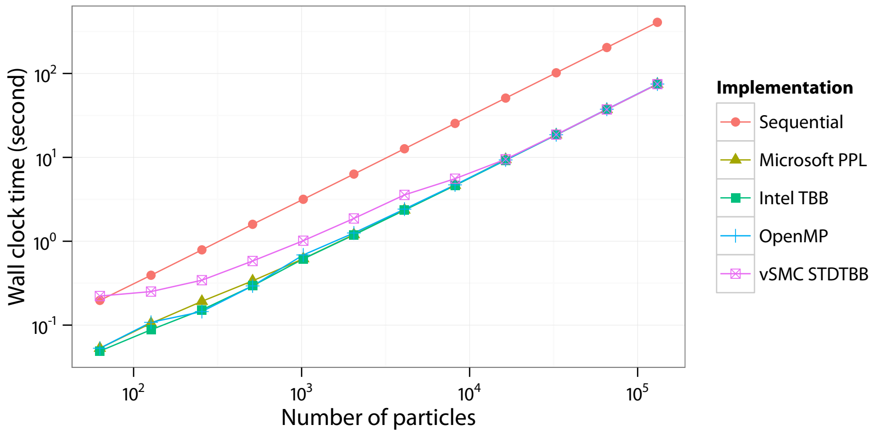
6.7.1 *Using the SMP module*

We consider five different parallel programming models supported by Intel C++ Composer XE 2013: sequential, Intel TBB, Intel Cilk Plus, OpenMP and C++11 `<thread>`. The program was built and run on a Ubuntu 12.10 workstation with an Xeon W3550 (3.06GHz, 4 cores, 8 hardware threads through hyper-threading) CPU. A four components model and 100 iterations with a prior annealing scheme is used for all implementations. A range of numbers of particles are tested, from 2^3 to 2^{17}.

For different number of particles, the wall clock time and speedup are shown in Figure 6.1. For 10,000 or more particles, the differences are minimal among all the programming models. They all have roughly 550% speedup. With smaller number of particles, vSMC's C++11 parallelization is less efficient than other industry strength programming models. However, with 1000 or more particles, which is less than typical applications, the difference is not very significant.

6.7.2 *Using the OpenCL module*

The implementation of the same algorithm using OpenCL is quite similar to those using the SMP module. OpenCL implementations are also compared on the same workstation, which has an NVIDIA Quadro 2000 graphic card. OpenCL programs can be compiled to run on both CPU and GPU. For CPU implementation, there are Intel OpenCL [80] and AMD APP OpenCL [1] platforms. We use the Intel TBB implementation as a baseline for comparison. The same OpenCL implementation are used for all the CPU and GPU runtimes. Therefore they are not particularly



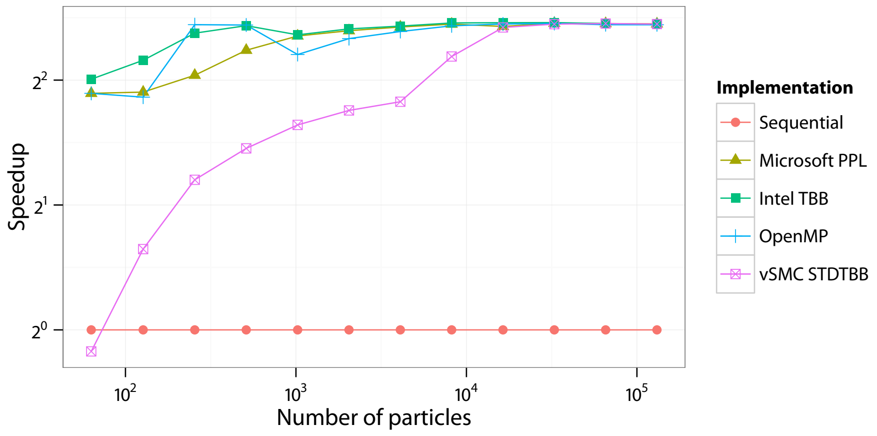
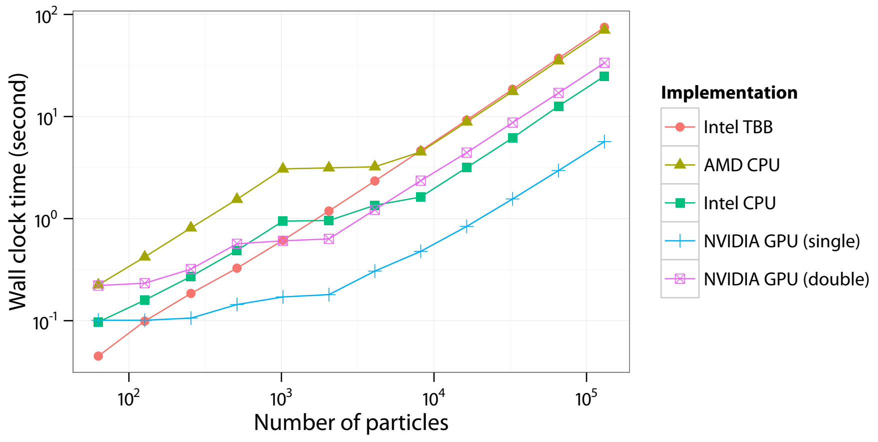


Figure 6.1 Performance of C++ implementations of Bayesian modeling for Gaussian mixture model (Linux; Xeon W3550, 3.06GHz, 4 cores, 8 threads). The top plots the wall clock time against the number of particles. The bottom plots the the speedup relative the sequential implementation against the number of particles.



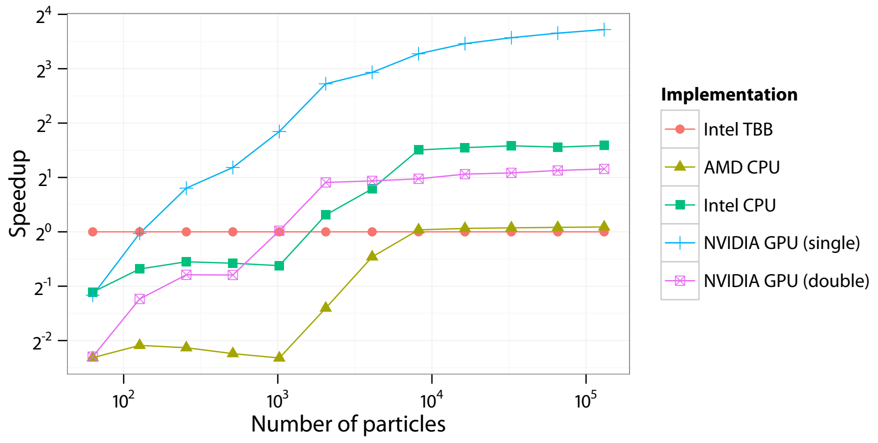


Figure 6.2 Performance of OpenCL implementations of Bayesian modeling for Gaussian mixture model (Linux; Xeon W3550 GPU, 3.06GHz, 4 cores, 8 threads; NVIDIA Quadro 2000). The top plots the wall clock time against the number of particles. The bottom plots the the speedup relative the sequential implementation against the number of particles.

optimized for any of them. For the GPU implementation, in addition to double precision, we also tested a single precision configuration. Unlike modern CPUs, which have the same performance for double and single precision floating point operations (unless SIMD instructions are used, the performance gain of which can vary considerably among different applications), GPUs penalize double precision performance heavily.

For different number of particles, the wall clock time and speedup are plotted in Figure 6.2. With smaller number of particles, the OpenCL implementations have a high overhead when compared to the Intel TBB implementation. With a large number of particles, AMD APP OpenCL has performance similar to that of the Intel TBB implementation. Intel OpenCL is about 40% faster than the Intel TBB implementation. This is due to more efficient vectorization and compiler optimizations. The double precision performance of the NVIDIA GPU has a 220% speedup and the single precision performance has nearly 1600% speedup. As a rough reference for the expected performance gain, the CPU has a theoretical peak performance of 24.48 GFLOPS (floating point operations per seconds, measured in the unit 10^9). The GPU has a theoretical peak performance of 60 GFLOPS in double precision and 480 GFLOPS in single precision. This represents 245% and 1960% speedup compared to the CPU, respectively.

6.7.3 *Performance and productivity*

Performance alone is not enough for a software to be useful. The productivity, the efforts needed to develop new algorithms, should also be taken into consideration. Due to the low level natural of C++, it certainly takes more effort to develop an algorithm using vSMC than, say LibBi or BiiPS. However, SMC does provide a some advantages. First, some application of SMC algorithms may not fit into the framework of those softwares. The framework of SMC is general enough for them to be implemented with relative ease.

Second, one can choose various parallel programming models while using the same implementation, as we have seen in Section 6.4.2. This can be particularly useful in a few scenarios,

1. Many parallel programming models do not coexist in the same program well. Some of them such as Intel TBB and Intel Cilk Plus has explicit support for OpenMP. Much less so can be said for the others. Often, some other part of the program may also be parallelized with a particular programming model familiar to users. In this case, using vSMC one can often freely choose the same one for the SMC algorithm's parallelization. See [166] for all the programming models supported by the library.
2. Often an algorithm is first developed on a desktop or laptop with only multicore processors. Later it may be deployed on larger computers to process bigger data. With vSMC it is possible to use the implementation on a SMP system, with little modifications, for the larger computer. In [166] there is a full fledged example showing the use MPI.

Overall, we found the productivity of vSMC is at a similar level of SMCTC. For example, the particle filter example in [90] can be implemented in vSMC using roughly the same number of lines of code. Considering that there is a significant performance gain through parallelization, as seen in Section 6.7.1, we believe the effort of using a C++ library is adequate.

The library also support OpenCL parallelization. The performance is impressive as seen in Section 6.7.2. It is widely believed that OpenCL programming is tedious and hard. Limited by the scope of this chapter, the OpenCL implementation (distributed with the vSMC source) is not documented in this chapter. Overall the OpenCL implementation has about 800 lines including both host and device code. It is not an enormous increase of effort when compared to the 500 lines SMP implementation. Less than doubling the code base but gaining more than 15 times performance speedup, we consider the programming effort is relatively small. Moreover, the GPU used in the examples is relatively lower end and outdated. With better hardware, the same implementation has the potential to gain hundreds of times performance speedup.

In addition, the OpenCL language is essentially a variant of the C programming language. For the intended users of vSMC, those with some knowledge of C++, writing OpenCL kernels (the part of the program that are executed on the device, such as GPUS) is not a difficult task. What often makes OpenCL programming difficult is the management of the devices. It involves the understanding an array of layers of the underlying hardware. There are examples[3] where a major part of the program is irrelevant to the algorithm itself. This is not the case when using vSMC. The library provides facilities to manage OpenCL platforms and devices as well as common operations. The implementations of SMC algorithms using OpenCL, compared to using the SMP module, only requires a marginal addition of efforts to manage the OpenCL platform.

6.8 DISCUSSIONS

For C++ proficient researchers that are interested in developing new algorithms, the vSMC library can be appealing for a few reasons. First, it provides an easy to use interface that can be used to implement standard algorithms with minimal efforts. Second, it is extensible. Limited by the scope, in this chapter we did not introduce the more technical part of the library that allows users to write non-standard algorithms. However, it is sufficient to note here that many parts of the library can be replaced by user implementations. For example, users can provide new resampling algorithms or non-standard numerical integration scheme for approximating the path sampling estimator, while reusing the library to perform other steps of the algorithms.

For users more interested in the application of SMC algorithms, basic C++ knowledge is sufficient to start using the library. Apart from a framework for the implementation of generic SMC algorithms, the library also provides utilities such as templates for the implementation of common Metropolis random walks.

[3]See `https://developer.apple.com/library/mac/samplecode/OpenCL_FFT` for examples

Through the examples, we have shown that the implementation of parallelized samplers is not more difficult than that of an serialized one. The performance is more or less close to ideal situations. This may not always be the case in reality. However, through all the examples in Chapter 5, we have found that there are always considerable speedup compared to serialized implementations. The OpenCL module further provides superior performance compared to the SMC module.

There are at least two interested directions of the future development of the library. The first is to provide an easier to use interface. The BUGS software and others certainly contributed to the popularity of MCMC algorithms among statisticians. In this thesis, we advocate the use of SMC algorithms for the purpose of Bayesian model comparison. It will almost certainly help to provide a easier to use software that enables researchers to develop new algorithms. The second is to include some important parallel programming models that are currently absent due to technical difficulties. One of them is the popular CUDA framework [122].

In summary, we believe the vSMC library provides an adequate balance among performance, ease of use, flexibility and extensibility.

7 CONCLUSIONS

This thesis is concerned with the use SMC for the purpose of Bayesian model comparison. A generic framework was developed. Practical implementation tools for generic SMC algorithms are also presented.

It is found that, compared to MCMC approach to Bayesian model comparison, the SMC approach is often more robust. Both the standard and the path sampling estimators are more stable when compared to the generalized harmonic mean estimator used in the MCMC setting. Though there are also estimators used for some specific MCMC algorithms such as the Gibbs sampling, that is more stable than the generalized harmonic mean estimator, they often require knowledge of the models that are often absent in reality. In contrast, the SMC algorithm and its estimators are more generic and therefore applicable in more areas of interest.

There are considerable performance gain of the adaptive SMC algorithms, in particular the CESS-based adaptive specification of distributions. Unlike adaptive MCMC algorithms, such strategies have little computational cost. In addition, it is also generic in the sense that it does not depend on a specific form of the intermediate distributions. It is recommended that such strategies should be employed for complex models, where the characteristics of the posterior distribution is hardly known and it is difficult to manually specify a smooth path from the prior towards the posterior.

In summary, the SMC framework for Bayesian model comparison presented in this thesis has the potential to solve many realistic problems which are previously difficult with the MCMC algorithms or requires significantly less efforts to optimize the algorithm.

7.1 CONTRIBUTIONS

In Chapter 5, the algorithms presented can accurately approximate the Bayes factor with little or no human tuning for many applications of interest. Some theoretical results of the use of path sampling estimator within this framework was developed as extension to the results of the standard estimator. A novel adaptive algorithm for specification of the placement of distributions in the SMC2 algorithm is introduced. It provides better (and more sensible) results than using the ESS as a criterion of how to introduce a new distribution. This method can be extended to other algorithms such as SMC3 straightforwardly. The performance of the presented algorithms is studied in detail through various empirical experiments.

In Chapter 6, a C++ library is introduced. It provides a tool for the implementation of generic SMC algorithms. Compared to some established softwares, it has either a higher level of flexibility in the sense of enabling the implementation of general algorithms instead of particular models; or higher performance through the use of parallel computing.

7.2 FUTURE DIRECTIONS

Though many convincing results have been shown in Chapter 5, theoretical development is also needed. In particular, the better performance of CESS-based adaptive scheme has only been shown empirically. It might be of interest to establish if it is better in some sense when compared to some commonly used deterministic scheme and under what conditions.

Some algorithms have potential applications in scenarios different than those shown in this thesis. For example, the SMC3 algorithm may be used for Bayesian model expansion. It may be of interest to see if the approach presented in this thesis has any significant advantage when compared to alternatives in terms of accuracy and computational efficiency.

In this thesis, most examples use a geometric annealing scheme to specify the intermediate distributions. However, the adaptive strategies proposed are not limited to this setting. There are other forms of the sequence of distributions, such as the data tempering mentioned before. The adaptive strategies proposed. It is of interest to see if strategies studied in this thesis can benefit more general situations.

The work presented in Chapter 6 can be useful for many researchers. However, it still requires considerable expertise in C++. It is of interest to provide a easier to use interface on top of the library. Some popular parallel programming models are not included in this library, such as CUDA [122]. Further work is needed to include them in the presented framework so the library can be more useful to those familiar with them.

[1] Advanced Micro Devices, Inc. *AMD Accelerated Parallel Processing OpenCL Programming Guide*. Version 2.8. 2012. URL: `http://developer.amd.com/tools-and-sdks/heterogeneous-computing/amd-accelerated-parallel-processing-app-sdk`.

[2] Hirotugu Akaike. "A new look at the statistical model identification". *IEEE Transactions on Automatic Control* 19.6 (1974), pp. 716–723.

[3] Hirotugu Akaike. "Information theory and an extension of the maximum likelihood principle". In: *Second international symposium on information theory*. Akademiai Kiado, 1973, pp. 267–281.

[4] Andrei Alexandrescu. *Modern C++ Design: Generic Programming and Design Patterns Applied*. Boston, USA: Addison Wesley, 2004.

[5] David M. Allen. "The relationship between variable selection and data agumentation and a method for prediction". *Technometrics* 16.1 (1974), pp. 125–127.

[6] David H. Anderson. *Compartmental Modelling and Tracer Kinetics*. Vol. 50. Lecture Notes in Biomathematics. New York, USA: Springer-Verlag, 1983.

[7] Christophe Andrieu, Arnaud Doucet, and Roman Holenstein. "Particle Markov chain Monte Carlo methods". *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 72.3 (2010), pp. 269–342.

[8] Christophe Andrieu, Arnaud Doucet, and Vladislav Tadic. "One-Line Parameter Estimation in General State-Space Models Using a Pseudo-Likelihood Approach". In: *System Identification*. Vol. 16. 1. 2012, pp. 500–505.

[9] Christophe Andrieu and Eric Moulines. "On the ergodicity properties of some adaptive MCMC algorithms". *The Annals of Applied Probability* 16.3 (2006), pp. 1462–1505.

[10] Christophe Andrieu, Éric Moulines, and Pierre Priouret. "Stability of stochastic approximation under verifiable conditions". *SIAM Journal on control and optimization* 44.1 (2005), pp. 283–312.

[11] Christophe Andrieu and Johannes Thoms. "A tutorial on adaptive MCMC". *Statistics and Computing* 18.4 (2008), pp. 343–373.

[12] Whit Armstrong. *C++ version of BUGS*. 2012. URL: `https://github.com/armstrtw/CppBugs`.

[13] Yves F. Atchadé, Gareth O. Roberts, and Jeffrey S Rosenthal. "Towards optimal scaling of Metropolis-coupled Markov chain Monte Carlo". *Statistics and Computing* 21.4 (2010), pp. 555–568.

[14] James O. Berger and José M. Bernardo. "Estimating a product of means: Bayesian analysis with reference priors". *Journal of American Statistical Association* 84.405 (1989), pp. 200–207.

[15] James O. Berger and José M. Bernardo. "On the development of reference priors". In: *Bayesian Statistics 4*. Ed. by José M. Bernardo et al. Oxford University Press, 1992, pp. 35–60.

[16] James O. Berger and José M. Bernardo. "Ordered group reference priors with application to the multinomial problem". *Biometrika* 79.1 (1992), pp. 25–37.

[17] James O. Berger and Mohan Delampady. "Testing precise hypotheses". *Statistical Science* 2.3 (1987), pp. 317–335.

[18] James O. Berger and Luis R. Pericchi. "Objective Bayesian methods for model selection: introduction and comparison". *Model Selection* 38 (2001), pp. 135–207.

[19] José M. Bernardo. "Reference posterior distributions for Bayesian inference". *Journal of Royal Statistical Society B* 41.2 (1979), pp. 113–147.

[20] José M. Bernardo and Adrian F. M. Smith. *Bayesian Theory*. Wiley Series in Probability and Statistics. John Wiley & Sons, Inc., 1994.

[21] Alexandros Beskos, Ajay Jasra, and Alexandre Thiery. *On the convergence of adaptive sequential Monte Carlo methods*. Mathematics e-print 1306.6462. ArXiv, 2013.

[22] Leo Breiman and Philip Spector. "Submodel Selection and Evaluation in Regression. The X-Random Case". *International Statistical Review* 60.3 (1992), pp. 291–319.

[23] Kenneth P. Burnham and David R. Anderson. *Model Selection and Multimodel Inference: A Practical Information-theoretic Approach*. 2nd ed. New York: Springer-Verlag, 2002.

[24] Ben Calderhead and Mark Girolami. "Estimating Bayes factors via thermodynamic integration and population MCMC". *Computational Statistics & Data Analysis* 53.12 (2009), pp. 4028–4045.

[25] Francois Caron et al. *BiiPS: Bayesian Inference with Interacting Particle Systems*. Version 0.7.2. 2012. URL: `https://alea.bordeaux.inria.fr/biips/doku.php`.

[26] George Casella and Christian P. Robert. "Post-processing accept-reject samples: recycling and rescaling". *Journal of Computational and Graphical Statistics* 7.2 (1998), pp. 139–157.

[27] L. Chen et al. "PFLib: an object oriented MATLAB toolbox for particle filtering". In: *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*. Vol. 6567. Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series. 2007.

[28] Siddhartha Chib. "Marginal likelihood from the Gibbs output". *Journal of the American Statistical Association* 90.432 (1995), pp. 1313–1321.

[29] Siddhartha Chib and Ivan Jeliazkov. “Marginal likelihood from the Metropolis-Hastings output”. *Journal of the American Statistical Association* 96.453 (2001), pp. 270–281.

[30] Nicolas Chopin. “A sequential particle filter method for static models”. *Biometrika* 89.3 (2002), pp. 539–552.

[31] Nicolas Chopin. “Central limit theorem for sequential Monte Carlo methods and its application to Bayesian inference”. *The Annals of Statistics* 32.6 (2004), pp. 2385–2411.

[32] Gerda Claeskens and Nils Lid Hjort. *Model Selection and Model Averaging*. Cambridge Series in Statistical and Probablistic Mathematics. Cambridge, UK: Cambridge University Press, 2008.

[33] M. Clyde. “Bayesian model averaging and model search strategies”. In: *Bayesian Statistics 6*. Oxford University Press, 1999, pp. 157–185.

[34] T. M. Cover and J. A. Thomas. *Elements of Inforamtion Theory*. New York, USA: John Wiley and Sons, 1991.

[35] Bruno De Finetti. *Theory of probability: A critical introductory treatment*. Vol. 1. John Wiley & Sons, 1974.

[36] Bruno De Finetti. *Theory of probability: A critical introductory treatment*. Vol. 2. John Wiley & Sons, 1975.

[37] Pierre Del Moral. *Feynman-Kac Formulae: Genealogical and Interacting Particle Systems with Applications*. New York: Springer-Verlag, 2004.

[38] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. “On adaptive resampling strategies for sequential Monte Carlo methods”. *Bernoulli* 18.1 (2012), pp. 252–278.

[39] Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. “Sequential Monte Carlo samplers”. *Journal of Royal Statistical Society B* 68.3 (2006), pp. 411–436.

[40] Randal Douc, Olivier Cappé, and Eric Moulines. “Comparison of resampling schemes for particle filtering”. In: *Proceedings of the 4th International Symposium on Imange and Signal Processing and Analysis*. 2005, pp. 1–6.

[41] David Draper. “Assessment and Propagation of Model Uncertainty”. *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (1995), pp. 45–97.

[42] T. H. Fan and J. O. Berger. “Behavior of the posterior distribution and inferences for a normal mean with t prior distributions”. *Statistics and Decisions* 10 (1992), pp. 99–120.

[43] Y. Fan, D.S. Leslie, and M. P. Wand. “Generalised linear mixed model analysis via sequential Monte Carlo sampling”. *Electronic Journal of Statistics* 2 (2008), pp. 916–938.

[44] Paul Fearnhead and Benjamin M. Taylor. “An adaptive Sequential Monte Carlo sampler”. *Bayesian analysis* 8.2 (2013).

[45] Ronald A Fisher. *Statistical Methods and Scientific Inference*. Hafner Publishing Co., 1956.

[46] Chris Fonnesbeck et al. *Bayesian inference in Python*. Version 2.3. 2013. URL: `https://github.com/pymc-devs/pymc`.

[47] W. G. Frankle and M Laruelle. “Neuroreceptor imaging in psychiatric disorders”. *Annals of Nuclear Medicine* 16 (2002), pp. 437–46.

[48] N. Friel, M. Hurn, and J. Wyse. “Improving power posterior estimation of statistical evidence”. *ArXiv* 1209.3198 (Sept. 2012), pp. 1–24.

[49] Sanjiv Sam Gambhir. “Molecular imaging of cancer with positron emission tomography”. *Nature Reviews Cancer* 2 (2002), pp. 683–693.

[50] Eric Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. New York, USA: Addison-Wesley, 1995.

[51] Seymour Geisser. “The predictive sample reuse method with applications”. *Journal of the American Statistical Association* 70.350 (1975), pages.

[52] Alan E. Gelfand and Dipak K. Dey. “Bayesian model choice: Asymptotics and exact calculations”. *Journal of Royal Statistical Society B (Statistical Methodlogoy)* 56.3 (1994), pp. 501–514.

[53] Alan E. Gelfand and Adrian F. M. Smith. “Sampling-based approaches to calculating marginal densities”. *Journal of the American Statistical Association* 85.410 (1990), pp. 398–409.

[54] A. Gelman, G. Roberts, and W. Gilks. “Efficient Metropolis jumping rules”. In: *Bayesian Statistics 5*. Oxford University Press, 1995, pp. 559–607.

[55] A. Gelman and K. Shirley. “Inference from Simulations and Monitoring Convergence”. In: *Handbook of Markov Chain Monte Carlo*. Ed. by S. Brooks et al. CRC Press, 2011, pp. 163–174.

[56] Andrew Gelman and Xiao-Li Meng. “Simulating normalizing constants: From importance sampling to bridge sampling to path sampling”. *Statistical Science* 13.2 (1998), pp. 163–185.

[57] A. Gelman et al. *Bayesian Data Analysis*. 2nd ed. Chapman & Hall/CRC Texts in Statistical Science. New York, USA: Taylor & Francis, 2003.

[58] Stuart Geman and Donald Geman. “Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images”. *Journal of Applied Statistics* 20.5 (1993), pp. 25–62.

[59] John Geweke. “Bayesian inference in econometric models using Monte Carlo integration”. *Econometrica: Journal of the Econometric Society* 57.6 (1989), pp. 1317–1339.

[60] Charles J. Geyer and Leif T. Johnson. *mcmc: Markov Chain Monte Carlo*. R package version 0.9-2. 2013. URL: `http://CRAN.R-project.org/package=mcmc`.

[61] W. Gilks and G. Roberts. “Strategies for improving MCMC”. In: *Markov Chain Monte Carlo in Pratice*. Ed. by W. Gilks et al. New York, USA: Chapman and Hall, 1996, pp. 89–114.

[62] Gopi Goswami. *SMC: Sequential Monte Carlo (SMC) Algorithm*. R package version 1.1. 2011. URL: `http://CRAN.R-project.org/package=SMC`.

[63] Peter J. Green. "Reversible jump Markov chain Monte Carlo computation and Bayesian model determination". *Biometrika* 82.4 (1995), pp. 711–732.

[64] Peter J. Green and David I. Hastie. *Reversible jump MCMC*. Tech. rep. 2009. URL: `http://www.maths.bristol.ac.uk/~peter/papers/rjmcmc_20090613.pdf`.

[65] Peter J. Green and Antonietta Mira. "Delayed rejection in reversible jump Metropolis-Hastings". *Biometrika* 88.4 (2001), pp. 1035–1053.

[66] P.D. Grünwald, I.J. Myung, and M.A. Pitt. *Advances in Minimum Description Length: Theory and Applications*. Neural information processing series. Bradford Book, 2005.

[67] Roger N. Gunn, Steve R. Gunn, and Vincent J. Cunningham. "Positron emission tomography compartmental models". *Journal of Cerebral Blood Flow & Metabolism* 21.6 (2001), pp. 635–652.

[68] Heikki Haario, Eero Saksman, and Johanna Tamminen. "Adaptive proposal distribution for random walk Metropolis algorithm". *Computational Statistics* 14.3 (1999), p. 375.

[69] Heikki Haario, Eero Saksman, and Johanna Tamminen. "An adaptive Metropolis algorithm". *Bernoulli* 7.2 (2001), p. 223.

[70] David Hastie. "Towards automatic reversible jump Markov chain Monte Carlo". PhD thesis. 2005.

[71] W. Keith Hastings. "Monte Carlo sampling methods using Markov chains and their applications". *Biometrika* 57.1 (1970), pp. 97–109.

[72] Dimitri van Heesch. *Doxygen – Generating Documentation from Source Code*. Version 1.8.4. 2013. URL: `http://www.stack.nl/~dimitri/doxygen/index.html`.

[73] W.D. Hillis and G.L. Jr. Steele. "Data parallel algorithms". *Communications of the ACM* 29 (1986), pp. 1170–1184.

[74] Susan E Hills and Adrian FM Smith. "Diagnostic plots for improved parameterization in Bayesian inference". *Biometrika* 80.1 (1993), pp. 61–74.

[75] U. Hjorth. *Computer Intensive Statistical Methods: Validation Model Selection and Bootstrap*. Chapman & Hall, 1994.

[76] J.P. Hobert, C.P. Robert, and C. Goutis. "Connectedness conditions for the convergence of the Gibbs sampler". *Statistics & Probability Letters* 33.3 (1997), pp. 235–240.

[77] Clifford M. Hurvich and Chin-ling Tsai. "Regression and time series model selection in small samples". *Biometrika* 76.2 (1989), pp. 297–307.

[78] Robert B Innis et al. "Consensus nomenclature for in vivo imaging of reversibly binding radioligands". *Journal of Cerebral Blood Flow and Metabolism* 27 (2007), pp. 1533–1539.

[79] Intel Cooperation. *Intel Cilk Plus Language Specification*. Version 1.1. 2011. URL: `http://cilkplus.org/`.

[80] Intel Cooperation. *Intel SDK for OpenCL Applications 2013*. 2013. URL: `http://software.intel.com/en-us/vcsource/tools/opencl-sdk`.

[81] John A. Jacquez. *Compartmental Analysis in Biology and Medicine*. 3rd. University of Michigan Press, 1996.

[82] Ajay Jasra, David A. Stephens, and Christopher C. Holmes. "On population-based simulation for static inference". *Statistics and Computing* 17.3 (2007), pp. 263–279.

[83] Ajay Jasra, David A. Stephens, and Christopher C. Holmes. "Population-based reversible jump Markov chain Monte Carlo". *Biometrika* 94.4 (2007), pp. 787–807.

[84] Ajay Jasra et al. "Inference for Lévy-Driven Stochastic Volatility Models via Adaptive Sequential Monte Carlo". *Scandinavian Journal of Statistics* 38.1 (2010), pp. 1–22.

[85] Ajay Jasra et al. "Interacting sequential Monte Carlo samplers for trans-dimensional simulation". *Computational Statistics & Data Analysis* 52.4 (2008), pp. 1765–1791.

[86] E.T. Jaynes. *Papers on Probability, Statistics, and Statistical Physics*. Ed. by R.D. Rosenkrantz. Kluwer Academic, 1989.

[87] Harold Jeffreys. "An invariant form for the prior probability in estimation problems". *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences* 186.1007 (1946), pp. 453–461.

[88] Harold Jeffreys. *Theory of Probability*. Clarendon Press. 1961.

[89] Ci-Ren Jiang, John A. D. Aston, and Jane-Ling Wang. "Smoothing dynamic positron emission tomography time courses using functional principal components". *NeuroImage* 47.1 (2009), pp. 184–193.

[90] Adam M. Johansen. "SMCTC: Sequential Monte Carlo in C++". *Journal of Statistical Software* 30.6 (2009), pp. 1–41.

[91] Adam M. Johansen, Pierre Del Moral, and Arnaud Doucet. "Sequential Monte Carlo samplers for rare events". In: *Proceedings of the 6th International Workshop on Rare Event Simulation*. 2006, pp. 256–267.

[92] Adam M. Johansen, Arnaud Doucet, and Manuel Davy. "Particle methods for maximum likelihood estimation in latent variable models". *Statistics and Computing* 18.1 (2008), pp. 47–57.

[93] Adam M. Johansen et al. "Convergence of the SMC implementation of the PHD filte". *Methodology and Computing in Applied Probability* 8.2 (2006), pp. 265–291.

[94] Bi Jun et al. "Parallel resampling particle filter algorithm". *Journal of Computational Information Systems* 7.6 (2011), pp. 1838–1845.

[95] Robert E. Kass. “Bayes factors in practice”. *The Statistician* 42.5 (1993), pp. 551–560.

[96] Robert E. Kass and Adrian E. Raftery. “Bayes factors”. *Journal of the American Statistical Association* 90.430 (1995), pp. 773–795.

[97] Robert E. Kass and Suresh K. Vaidyanathan. “Approximate Bayes factors and orthogonal parameters, with application to testing equality of two binomial proportions”. *Journal of Royal Statistical Society B* 54.1 (1992), pp. 129–144.

[98] P.E. Kinahan and J.G. Rogers. “Analytic 3D image reconstruction using all detected events”. *IEEE Transactions on Nuclear Science* 36.1 (1989), pp. 964–968.

[99] Genshiro Kitagawa. “Monte Carlo filter and smoother for non-Gaussian nonlinear state space models”. *Journal of Computational and Graphical Statistics* 5.1 (1996), pp. 1–25.

[100] Augustine Kong, Jun S. Liu, and Wing Huang Wong. “Sequential imputations and Bayesian missing data problems”. *Journal of the American Statistical Association* 89.425 (1994), pp. 278–288.

[101] Kronos OpenCL Working Group. *The OpenCL Specification*. Version 1.2. 2012. URL: `http://www.khronos.org/opencl/`.

[102] S. Kullback and R. A. Leibler. “On information and sufficiency”. *The Annals of Mathematical Statistics* 22.1 (1951), pp. 79–86.

[103] A. A. Lammertsma and S. P. Hume. “Simplified Reference Tissue Model for PET Receptor Studies”. *Neuroimage* 4.3 (1996), pp. 153–8.

[104] Anthony Lee et al. “On the utility of graphics cards to perform massively parallel simulation of advanced Monte Carlo methods”. *Journal of Computational and Graphical Statistics* 19.4 (2010), pp. 769–789.

[105] Sangyeol Lee and Alex Karagrigoriou. “An asymptotically optimal selection of the order of a linear process”. *Sankhyā: The Indian Journal of Statistics, Series A* (2001), pp. 93–106.

[106] E. L. Lehmann. *Theory of point estimation*. New York, USA: John Wiley & Sons, 1983.

[107] Faming Liang and Wing Hung Wong. “Real-parameter evolutionary Monte Carlo with applications to Bayesian mixture models”. *Journal of the American Statistical Association* 96.454 (2001), pp. 653–666.

[108] Jun S. Liu and Rong Chen. “Sequential Monte Carlo methods for dynamic systems”. *Journal of the American Statistical Association* 93.443 (1998), pp. 1032–1044.

[109] Jun S. Liu, Wing H. Wong, and Augustine Kong. “Covariance Structure and Convergence Rate of the Gibbs Sampler with Various Scans”. *Journal of the Royal Statistical Society. Series B (Methodological)* 57.1 (1995), pp. 157–169.

[110] David Lunn et al. *The BUGS Book: A Practical Guide to Bayesian Analysis*. New York, USA: CRC Press, 2012.

[111] Enzo Marinari and Giorgio Parisi. “Simulated tempering: a new Monte Carlo scheme”. *EPL (Europhysics Letters)* 19.6 (1992), pp. 451–458.

[112] Andrew D. Martin, Kevin M. Quinn, and Jong Hee Park. “MCMCpack: Markov Chain Monte Carlo in R”. *Journal of Statistical Software* 42.9 (2011), pp. 1–22. URL: `http://www.jstatsoft.org/v42/i09/`.

[113] Michael McCool, Arch D. Robison, and James Reinders. *Structured Parallel Programming: Patterns for Efficient Computation*. MA, USA: Morgan Kaufmann, 2012.

[114] Robert McCulloch and Peter E. Rossi. “A Bayesian approach to testing the arbitrage pricing theory”. *Journal of Econometrics* 49.1-2 (1991), pp. 141–168.

[115] K. L. Mengersen and R. L. Tweedie. “Rates of convergence of the Hastings and Metropolis algorithms”. *The Annals of Statistics* 24.1 (1996), pp. 101–121.

[116] Nicholas Metropolis et al. “Equation of state calculations by fast computing machines”. *The Journal of Chemical Physics* 21.6 (1953), pp. 1087–1092.

[117] L. M. Murray. *Bayesian State-Space Modelling on High-Performance Hardware Using LibBi*. Mathematics e-print 1306.3277. ArXiv, 2013.

[118] L. M. Murray, A. Lee, and P. E. Jacob. *Rethinking resampling in the particle filter on graphics processing units*. Mathematics e-print 1301.4019. ArXiv, 2013, pp. 1–9.

[119] Peter Neal and Gareth Roberts. "Optimal scaling of random walk metropolis algorithms with non-Gaussian proposals". *Methodology and Computing in Applied Probability* 13.3 (2011), pp. 583–601.

[120] Radford M. Neal. "Annealed importance sampling". *Statistics and Computing* 11.2 (2001), pp. 125–139.

[121] Michael A. Newton and Adrian E. Raftery. "Approximate Bayesian inference with the weighted likelihood bootstrap". *Journal of Royal Statistical Society B* 56.1 (1994), pp. 3–48.

[122] NVidia. *CUDA C Programming Language*. Version 5.5. 2013. URL: `http://www.nvidia.com/object/cuda_home_new.html`.

[123] Man-Suk Oh and James O. Berger. "Adaptive importance sampling in Monte Carlo integration". *Journal of Statistical Computation and Simulation* 41.3-4 (1992), pp. 143–168.

[124] A. O'Hagan. "Outliers and credence for location parameter inference". *Journal of the American Statistical Association* 85.409 (1990), pp. 172–176.

[125] Jyh-Ying Peng et al. "Dynamic positron emission tomography data-driven analysis using sparse Bayesian learning". *IEEE Transactions on Medical Imaging* 27.9 (2008), pp. 1356–1369.

[126] Gareth W Peters. "Topics in sequential Monte Carlo samplers". MA thesis. 2005.

[127] Michael E. Phelps. "Positron emission tomography provides molecular imaging of biological processes". In: *Proceedings of the National Academy of Sciences*. Vol. 97. 16. 2000, pp. 9226–9233.

[128] Martyn Plummer et al. "CODA: Convergence diagnosis and output analysis for MCMC". *R News* 6.1 (2006), pp. 7–11. URL: `http://CRAN.R-project.org/doc/Rnews/`.

[129] R Core Team. *R: A Language and Environment for Statistical Computing*. Version 2.15.2. R Foundation for Statistical Computing. Vienna, Austria, 2013. URL: `http://www.r-project.org`.

[130] Adrian E. Raftery, David Madigan, and Jennifer A. Hoeting. "Bayesian model averaging for linear regression models". *Journal of the American Statistical Association* 92.437 (1997), pp. 179–191.

[131] Adrian E. Raftery et al. "Estimating the integrated likelihood via posterior simulation using the harmonic mean identity". In: *Bayesian Statistics 8*. Oxford University Press, 2006, pp. 1–45.

[132] Sylvia Richardson and Peter J. Green. "On Bayesian analysis of mixtures with an unknown number of components". *Journal of Royal Statistical Society B* 59.4 (1997), pp. 731–792.

[133] Christian P. Robert. "Convergence control methods for Markov chain Monte Carlo algorithms". *Statistical Science* 10.3 (1995), pp. 231–253.

[134] Christian P. Robert. *The Bayesian Choice: From Decision-theoretic Foundations to Computational Implementation*. 2nd ed. New York: Springer, 2007.

[135] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods*. Springer, 2004.

[136] G. O. Roberts, A. Gelman, and W. R. Gilks. "Weak convergence and optimal scaling of random walk Metropolis algorithms". *The Annals of Applied Probability* 7.1 (1997), pp. 110–120.

[137] Gareth O. Roberts and Jeffrey S Rosenthal. "Optimal scaling for various Metropolis-Hastings algorithms". *Statistical Science* 16.4 (2001), pp. 351–367.

[138] Gareth O. Roberts and Richard L. Tweedie. "Exponential convergence of Langevin distributions and their discrete approximations". *Bernoulli* 2.4 (1996), pp. 341–363.

[139] John K. Salmon et al. "Parallel random numbers: As easy as 1, 2, 3". *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis* (2011), pp. 1–12.

[140] Christian Schäfer and Nicolas Chopin. "Sequential Monte Carlo on large binary sampling spaces". *Statistics and Computing* (2011), pp. 1–22.

[141] Gideon Schwarz. "Estimating the dimension of a model". *The Annals of Statistics* 6.2 (1978), pp. 461–464.

[142] G. A. F. Seber and C. J. Wilde. *Nonlinear Regression*. Wiley Series in Probability and Statistics. New York, USA: John Wiley & Sons, 2003.

[143] Jun Shao. "An asymptotic theory for linear model selection". *Statistica Sinica* 7.2 (1997), pp. 221–242.

[144] Jun Shao. "Linear model selection by cross-validation". *Journal of the American Statistical Association* 88.422 (1993), pp. 486–494.

[145] Chris Sherlock and Gareth Roberts. "Optimal scaling of the random walk Metropolis on elliptically symmetric unimodal targets." *Bernoulli* 15.3 (2009), pp. 774–798.

[146] A N Shiryaev. *Probability*. New York: Springer-Verlag, 1995.

[147] Chor-Yiu Sin and Halbert White. "Information criteria for selecting possibly misspecified parametric models". *Journal of Econometrics* 71.1 (1996), pp. 207–225.

[148] David Spiegelhalter, Andrew Thomas, and Nicky Best. *BUGS – Bayesian Inference Using Gibbs Sampling*. Version 0.5. 1996. URL: `http://www.mrc-bsu.cam.ac.uk/bugs/`.

[149] M. Stone. “An asymptotic equivalence of choice of model by cross-validation and Akaike’s criterion”. *Journal of the Royal Statistical Society. Series B (Methodological)* 39.1 (1977), pp. 44–47.

[150] M. Stone. “Cross-validation: A review”. *Series Statistics* 9.1 (1978), pp. 127–139.

[151] M. Stone. “Cross-validatory choice and assessment of statistical predictions”. *Journal of the Royal Statistical Society. Series B (Methodological)* 36.2 (1974), pp. 111–147.

[152] Bjarne Stroustrup. *The C++ Programming Language*. 3rd ed. New York, USA: Addison-Wesley, 2003.

[153] Bjarne Stroustrup. *The Design and Evolution of C++*. New York, USA: Addison-Wesley, 1994.

[154] Nariaki Sugiura. “Further analysts of the data by Akaike’ s information criterion and the finite corrections”. *Communications in Statistics - Theory and Methods* 7.1 (1978), pp. 13–26.

[155] K Takeuchi. “Distribution of informational statistics and a criterion of model fitting”. Japanese. *Suri-Kagaku (Mathematics Sciences)* (153 1976), pp. 12–18.

[156] The LLVM Developer Group. *clang: A C Language Family Frontend for LLVM*. Version 3.2. 2013. URL: `http://clang.llvm.org/`.

[157] The MathWorks, Inc. *MATLAB – The Language of Technical Computing*. Version 2013b. 2013. URL: `http://www.mathworks.com/products/matlab/`.

[158] Luke Tierney. “Markov chains for exploring posterior distributions”. *The Annals of Statistics* 22.4 (1994), pp. 1701–1728.

[159] Luke Tierney and Joseph B. Kadane. “Accurate Approximations for Posterior Moments and Marginal Densities”. *Journal of the American Statistical Association* 81.393 (1986), pp. 82–86.

[160] Federico E. Turkheimer, Rainer Hinz, and Vincent J. Cunningham. “On the undecidability among kinetic models: From model selection to model averaging”. *Journal of Cerebral Blood Flow & Metabolism* 23.4 (2003), pp. 490–498.

[161] Todd L. Veldhuizen. *C++ Templates are Turing Complete*. Tech. rep. Compuer Science, Indiana University, 2003.

[162] Nick Whiteley. *Stabability properties of some particle filters*. Mathematics e-print 1306.6779. ArXiv, 2013.

[163] Darrell Whitley. “A genetic algorithm tutorial”. *Statistics and Computing* 4.2 (1994), pp. 65–85.

[164] Yuhong Yang. “Consistency of cross validation for comparing regression procedures.” *Annal of Statistics* 35.6 (2007), pp. 2450–2473.

[165] Bin Yu and Per Mykland. “Looking at Markov samplers through CUSUM path plots: a simple diagnostic idea”. *Statistics and Computing* 8.3 (1998), pp. 275–286.

[166] Y. Zhou. *vSMC: Parallel sequential Monte Carlo in C++*. Mathematics e-print 1306.5583. ArXiv, 2013.

[167] Y. Zhou, J. A. D. Aston, and A. M. Johansen. “Bayesian model comparison for compartmental models with applications in positron emission tomography”. *Journal of Applied Statistics* 40.5 (2013), pp. 993–1016.

[168] Y. Zhou, A. M Johansen, and J. A. Aston. *Towards automatic model comparison: an adaptive sequential Monte Carlo approach*. Mathematics e-print 1303.3123. ArXiv, 2013.

[169] Yan Zhou, Adam M. Johansen, and John A. D. Aston. “Bayesian model selection via path-sampling sequential Monte Carlo”. In: *Proceedings of IEEE Statistical Signal Processing Workshop*. 2012.

A MONTE CARLO METHODS

A.1 DISCRETE TIME MARKOV CHAIN

A Markov chain can be defined in terms of *transition kernels*. In general, consider a measurable space, say $(E, \mathcal{E})$, a transition kernel K is a function defined on $E \times \mathcal{E}$ such that $K(x, \cdot)$ is a probability measure for every $x \in E$ and $K(\cdot, A)$ is measurable for every $A \in \mathcal{E}$.

A discrete time Markov chain, denoted by (X^t) is a sequence of random variables $X^0, X^1, \dots, X^t, \dots$ such that conditional on $(x^{t-1}, \dots, x^0)$, X^t has the same distribution as it has conditional on x^{t-1}. Clearly a transition kernel is such a conditional distribution. In the context of MCMC algorithms, we are mostly concerned with *time homogeneous* Markov chains. A Markov chain (X^t) is said to be time homogeneous if it satisfies the following (*weak*) *Markov property*: For every initial distribution of X^0, μ, and every $(t+1)$ samples $(X^0, \dots, X^t)$, and some measurable function φ,

$$\mathbb{E}_\mu[\varphi(X^{t+1}, X^{t+2}, \dots)|x^0, \dots, x_n] = \mathbb{E}_{x^t}[\varphi(X^1, X^2, \dots)], \tag{A.1}$$

where $\mathbb{E}_\mu$ denotes the expectation with respect to the law of the chain given the initial distribution of X^0 and $\mathbb{E}_{x^t}$ denote the expectation with respect to the law of the chain conditional on $X^t = x^t$.

In the remaining of this section, we define a few properties of Markov chains that are particularly relevant in the study of MCMC algorithms.

A.1.1 *Irreducibility*

A Markov chain (X^t) with transition kernel K on $(E, \mathcal{E})$ is said to be ψ*-irreducible*, for a given measure ψ, if for every $A \in \mathcal{E}$ such that $\psi(A) > 0$, there exists t such that

$K^t(x, A) > 0$ for all $x \in E$. This chain is said to be *strongly ψ-irreducible* if $t = 1$ for every $A \in \mathcal{E}$ such that $\psi(A) > 0$.

An equivalent way of saying irreducibility is that for $x \in E$, $A \in \mathcal{E}$, $P_x(\tau_A \leq \infty) > 0$, where $\tau_A = \inf\{t \geq 1; X^t \in A\}$ is the smallest value of t that the chain enters the set A, namely the *first hitting time* at A and P_x denote the law of the chain conditional on the initial state x. In other words, the probability of reach any set A in finite many steps is positive.

Two related concepts, *atom* and *small set*, are useful for formally defining aperiodicity and ergodicity later. A Markov chain (X^t) is said to have an *atom* $\alpha \in \mathcal{E}$ if there exists an associated nonzero measure η such that, $K(x, A) = \eta(A)$ for all $x \in \alpha$ and $A \in \mathcal{E}$. A set C is said to be *small* if there exists $m \in \mathbb{N}$ and a nonzero measure η_m such that $K^m(x, A) \geq \eta_m(A)$, for all $x \in C$ and $A \in \mathcal{E}$.

A.1.2 *Cycles and aperiodicity*

A ψ-irreducible chain (X^t) has a *cycle of length d* if there exists a small set C, an integer M, and a probability distribution η_M such that d is the greatest common denominator of

$$\{m \geq 1;\ \text{There exists } \varepsilon_m > 0 \text{ such that } C \text{ is small for } \eta_m \geq \varepsilon_m \eta_M\}.$$

A chain is *aperiodic* if $d = 1$. If there exists a small set C and an associated measure η_1 such that $\eta_1(C) > 0$, that is it is possible to go from C to C in a single step, the chain is said to be *strongly aperiodic*.

A sufficient condition for aperiodicity is that the kernel is positive in a neighborhood of a state x. Then the chain can stay in this neighborhood for an arbitrary time. If a chain is not aperiodic, then the return from one state to its own neighborhood will requires a forced passage through another part of the space, which is clearly an undesired property for an MCMC algorithm. It will be shown that for the algorithms discussed in this chapter, they are aperiodic.

A.1.3 *Recurrence*

For a Markov chain (X^t) on $(E, \mathcal{E})$, a set $A \in \mathcal{E}$ is said to be *recurrent* if for all $x \in A$, $\mathbb{E}_x[N_A] = \infty$ where $N_A = \sum_{t=1}^{\infty} \mathbb{I}_A(X^t)$ is the number of passages through A. The Markov chain is said to be recurrent if there exists a measure ψ such that the chain is ψ-irreducible and for all $A \in \mathcal{E}$ such that $\psi(A) > 0$, A is recurrent.

A sufficient condition for a ψ-irreducible chain to be recurrent is that there exists a small set C with $\psi(C) > 0$ such that $P_x(\tau_C < \infty) = 1$ for all $x \in C$ where τ_C is the first hitting time at C.

A stronger property is called the *Harris recurrence*. A set $A \in \mathcal{E}$ is said to be Harris recurrent if for all $x \in A$, $P_x(N_A = \infty) = 1$. The ψ-irreducible Markov chain is said to be Harris recurrent if for all $A \in \mathcal{E}$ such that $\psi(A) > 0$, A is Harris recurrent.

Informally, Harris recurrence says that starting from *everywhere* in the space E, every part of the space will be visited for infinite instances with probability one. This is important for MCMC algorithms in the sense that Harris recurrence guarantees unique limiting distribution (up to a multiplicative factor).

A.1.4 *Invariant measure*

A σ-finite measure π is *invariant* for a Markov chain with transition kernel K if,

$$\pi(A) = \int K(x, A)\pi(\mathrm{d}\, x) \tag{A.2}$$

for all $A \in \mathcal{E}$. When an invariant *probability* measure exists for a ψ-irreducibility chain, the chain is said to be *positive*. A positive chain is always recurrent. Also the invariant measure is unique for a recurrent chain, up to a multiplicative factor. It is trivial to see that, for a invariant probability measure π of a Markov chain (X^t), if $X^0 \sim \pi$, then $X^t \sim \pi$ for all $t \geq 1$. Thus this distribution is also often referred to as the *stationary* measure.

A related concept is the *reversibility*. A stationary Markov chain (X^t) is said to be *reversible* if the distribution of X^{t+1} conditional on $X^t = x$ is the same as the distribution of X^t conditional on $X^{t+1} = x$. Intuitively, this says that the direction of time is irrelevant. The chain has the same stationary if it travels backward in time. A sufficient condition for a Markov chain to have an invariant probability distribution π and be reversible is the existence of the *detailed balance* condition,

$$\pi(x)K(x, y) = \pi(y)K(y, x). \tag{A.3}$$

A.1.5 *Ergodicity*

As stated in the beginning of this section, MCMC algorithms relies on the limiting π of a Markov chain (X^t), which has the property that, if $X^t \sim \pi$, then $X^{t+1} \sim \pi$. In other words, we are interested in the convergence of the distribution $P^t_\mu = \mu K^t$ where μ is the initial distribution of X^0. More importantly, we are interested in the independence of initial condition μ of its limiting behavior when $n \to \infty$. In the following we establish that the invariant distribution π is such a limiting distribution.

For a Harris recurrent and positive Markov chain (X^t) with transition kernel K and invariant distribution π, an atom α is *ergodic* if

$$\lim_{t\to\infty} |K^t(\alpha, \alpha) - \pi(\alpha)| = 0, \tag{A.4}$$

where $K(\alpha, \alpha) = \int_\alpha K(x, \alpha)\pi(\mathrm{d}\,x)$ and $K^t = K \circ K^{t-1}$. For more general situations, the convergence is established through the *total-variation norm*, defined for two measure μ_1 and μ_2 on the space $(E, \mathcal{E})$,

$$\|\mu_1 - \mu_2\|_{TV} = \sup_{A\in\mathcal{E}} |\mu_1(A) - \mu_2(A)|. \tag{A.5}$$

Two important results are that, if a Markov chain (X^t) is Harris recurrent, positive and aperiodic, with transition kernel K and invariant distribution π, then

$$\lim_{t\to\infty} \|\mu K^t(x, \cdot) - \pi\|_{TV} = 0 \tag{A.6}$$

for every initial distribution μ. And this total-variation norm decreases in t. From here, it becomes clear why the recurrence and aperiodicity discussed before are important for MCMC algorithms. Note that the above results also implies that, for bounded function φ,

$$\lim_{t\to\infty} |\mathbb{E}_\mu[\varphi(X^t)] - \mathbb{E}_\pi[\varphi(X)]| = 0, \tag{A.7}$$

where the first expectation is with respect to P^n_μ, and the second is for a random variable distributed with π. This result establishes the validity of using dependent samples from MCMC algorithms for the approximation of integration with respect to π.

However, the above results only states that the Markov chain will converge. It does not imply how fast the chain converges to its limiting distribution. Two stronger form of convergence is *geometric* and *uniform* ergodicity.

A Markov chain (X^t) with transition kernel K on $(E, \mathcal{E})$ and invariant distribution π, is said to be *geometrically ergodic*, if there exists $r > 1$ such that for all $x \in E$,

$$\sum_{t=1}^{\infty} r^n \|\mu K^t(x, \cdot) - \pi\|_{TV} < \infty. \tag{A.8}$$

This implies that,

$$\|\mu K^t(x, \cdot) - \pi\|_{TV} \le r^{-t} M(x) \tag{A.9}$$

where $M(x) = \sum_{t=1}^{\infty} r^t \|\mu K^t(x, \cdot) - \pi\|_{TV}$. In other words, the chain converges at least at a speed of a geometric sequence. We emphasize that $M(x)$ is a function of the initial value x.

A stronger form of convergence, *uniform* ergodicity requires that the convergence speed is the same for all $x \in E$, or in other words $M(x)$ is bounded. That is, the above convergence holds for a finite constant $M = \sup_x M(x)$.

B SEQUENTIAL MONTE CARLO FOR BAYESIAN COMPUTATION

B.1 PROOF OF PROPOSITION 5.1

We begin by making some identifications which allow the connection between the SMC sampler algorithm presented above and Feynman-Kac formula to be made explicit as the proof relies on approaches pioneered in [37]. Throughout this appendix we write $\eta K(\cdot) = \int \eta(dx)K(x, \cdot)$ for any compatible measure η and Markov kernel K and $\eta(\varphi) = \int \eta(dx)\varphi(x)$ for any η-integrable function φ.

A Feynman-Kac formula describes the law of a Markov chain on $\{(E_t, \mathcal{E}_t)\}_{t\geq 0}$ (with initial distribution $\hat{\eta}_0$ and transitions M_t) evolving in the presence of a (time-varying) potential (described by G_t) such that the marginal law of the t^{th} coordinate is:

$$\hat{\eta}_t(A) = \frac{\int_{E_1\times\ldots\times E_{t-1}\times A} \hat{\eta}_0(d\tilde{x}_0)\prod_{i=1}^t M_i(\tilde{x}_{i-1}, d\tilde{x}_i)G(\tilde{x}_i)}{\int_{E_1\times\ldots\times E_t} \hat{\eta}_0(d\tilde{x}_0^{'})\prod_{i=1}^t M_i(\tilde{x}_{i-1}^{'}, d\tilde{x}_i^{'})G(\tilde{x}_i^{'})}$$

for any measurable set A. It is convenient to define the operator

$$\hat{\Phi}_t(\eta)(\mathrm{d}\,\tilde{x}_t) = G_t(\tilde{x}_t)\eta M_t(\mathrm{d}\,\tilde{x}_t)/\eta M_t(G_t)$$

which allows us to write, recursively, $\hat{\eta}_t = \hat{\Phi}_t(\hat{\eta}_{t-1})$ and to define the intermediate distributions $\eta_t = \hat{\eta}_{t-1}M_t$ such that $\hat{\eta}_t(\mathrm{d}\,\tilde{x}_t) = G_t(\tilde{x}_t)\eta_t(d\tilde{x}_t)/\eta_t(G_t)$.

If $\mathcal{X}$ denotes the space upon which an SMC sampler with MCMC proposal K_t at time t and sequence of target distributions π_t operates, then we obtain π_t as the final coordinate marginal of the Feynman-Kac distribution at time t if we identify $E_t = \mathcal{X}^t$ and

$$M_t(\tilde{x}_{t-1}, d\tilde{x}_t) = \delta_{\tilde{x}_{t-1}}(d\tilde{x}_{t,1:t-1})K_t(\tilde{x}_{t,t-1}, d\tilde{x}_t)$$
$$G_t(\tilde{x}_t) = \pi_t(\tilde{x}_{t,t-1})/\pi_{t-1}(\tilde{x}_{t,t-1}).$$

To provide symmetry between the simulation system and the ideal system which it targets, it is convenient to let $\tilde{X}_t^i$ denote the extended sample corresponding to X_t^i at iteration t together with the full collection of values which its ancestors took during previous iterations (i.e., $\tilde{X}_t^i$ corresponds to the particle system obtained by sampling according to M_t above rather than K_t at each iteration). It is then convenient to write the particle approximation at time t as

$$\hat{\eta}_t^N(d\tilde{x}_t) = \sum_{i=1}^N \frac{G_t(\tilde{X}_t^i)}{\sum_{j=1}^N G_t(\tilde{X}_t^j)} \delta_{\tilde{X}_t^i}(d\tilde{x}_t).$$

We refer the reader to [37] for further details of the connection between such particle systems and the Feynman-Kac formula.

In order to proceed, we prove the following more general result to which Proposition 5.1 is a direct corollary.

PROPOSITION B.1. *Under the regularity conditions given in [37, sec. 9.4,], a weighted sum of integrals obtained from successive generations of the particle approximation of a Feynman-Kac flow $\{\hat{\eta}_t\}_{t=0}^T$, with the application of multinomial resampling at every iteration, obeys a central limit theorem in the following sense, for a collection of finite weights $\beta_t \in \mathbb{R}$ and bounded measurable functions $\xi_t : E_t \to \mathbb{R}$ (where, in the historical process case described above it is required that $\xi_t(\tilde{x}_t) = \xi_t(\tilde{x}_{t,t})$):*

$$\lim_{N\to\infty} \sqrt{N} \sum_{t=0}^T \beta_t(\hat{\eta}_t^N(\xi_t) - \hat{\eta}_t(\xi_t)) \xrightarrow{D} \mathcal{N}(0, V_T(\xi_{0:T})) \tag{B.1}$$

where V_t, $0 \leq t \leq T$ is defined by the following recursion,

$$\begin{aligned} V_0(\xi_0) =& \beta_0 \int \hat{\eta}_0(x_0)(\xi_0(x_0) - \eta_0(\xi_0))^2 dx_0 \\ V_t(\xi_{0:t}) =& V_{t-1}\left(\xi_{0:t-2}, \xi_{t-1} + \frac{\beta_t}{\beta_{t-1}} \frac{M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)])}{\hat{\eta}_{t-1}M_t(G_t)}\right) \\ &+ \beta_t^2 \hat{\eta}_t\left(\frac{G_t(\cdot)(\xi_t(\cdot) - \hat{\eta}_t(\xi_t))^2}{\hat{\eta}_t(G_t)}\right). \end{aligned} \tag{B.2}$$

The strategy of the proof is to decompose the error as that propagated forward from previous times and that due to sampling at the current time, just as in [37].

First note that the term $\hat{\eta}_t^N(\xi_t) - \hat{\eta}_t(\xi_t)$ can be rewritten as

$$\hat{\eta}_t^N(\xi_t) - \hat{\eta}_t(\xi_t) = \hat{\eta}_t^N(\xi_t) - \hat{\Phi}_t(\hat{\eta}_{t-1}^N)(\xi_t) + \hat{\Phi}_t(\hat{\eta}_{t-1}^N)(\xi_t) - \hat{\eta}_t(\xi_t) \tag{B.3}$$

and the weighted sum,

$$T_t^N(\xi_{0:t}) = \sqrt{N}\sum_{j=0}^{t} \beta_j(\hat{\eta}_j^N(\xi_j) - \hat{\eta}_j(\xi_j)) \tag{B.4}$$

can therefore be written as

$$\begin{aligned} T_t^N(\xi_{0:t}) &= T_{t-1}^N(\xi_{0:t-1}) + \sqrt{N}\beta_t(\hat{\eta}_t^N(\xi_t) - \hat{\eta}_t(\xi_t)) \\ &= \bar{T}_t^N(\xi_{0:t}) + \chi_t^N(\xi_t) \end{aligned} \tag{B.5}$$

where

$$\bar{T}_t^N(\xi_{0:t}) = T_{t-1}^N(\xi_{0:t-1}) + \sqrt{N}\beta_t(\hat{\Phi}_t(\hat{\eta}_{t-1}^N)(\xi_t) - \hat{\eta}_t(\xi_t)) \tag{B.6}$$

$$\chi_t^N(\xi_t) = \sqrt{N}\beta_t(\hat{\eta}_t^N(\xi_t) - \hat{\Phi}_t(\eta_{t-1}^N)(\xi_t)) \tag{B.7}$$

Lemma B.1 shows that error propagation leads to controlled normal errors; Lemma B.2 shows that the act of sampling during each iteration also produces a normally-distributed error and Lemma B.3 shows that these two normal errors can be combined leading by induction to Proposition B.1.

LEMMA B.1. *Under the conditions of Proposition B.1, if*

$$T_{t-1}^N(\xi_{0:t-1}) \xrightarrow{D} \mathcal{N}(0, V_{t-1}(\xi_{0:t-1})),$$

then

$$\bar{T}_t^N(\xi_{0:t}) \xrightarrow{D} \mathcal{N}(0, \bar{V}_t(\xi_{0:t})) \tag{B.8}$$

where

$$\bar{V}_t = V_{t-1}\left(\xi_{0:t-2}, \xi_{t-1} + \frac{\beta_t}{\beta_{t-1}}\frac{M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)])}{\hat{\eta}_{t-1}M_t(G_t)}\right). \tag{B.9}$$

Proof. We begin by re-expressing the difference of interest in a more convenient form:

$$\begin{aligned} \hat{\Phi}(\hat{\eta}_{t-1}^N)(\xi_t) - \hat{\eta}_t(\xi_t) &= \frac{1}{\hat{\eta}_{t-1}^N M_t(G_t)}\{\hat{\eta}_{t-1}^N M_t(G_t\xi_t) - \hat{\eta}_{t-1}^N M_t(G_t)\hat{\eta}_t(\xi_t)\} \\ &= \frac{1}{\hat{\eta}_{t-1}^N M_t(G_t)}\{(\hat{\eta}_{t-1}^N - \hat{\eta}_{t-1})M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)])\} \end{aligned} \tag{B.10}$$

where the final equality is a simple consequence of the fact that for any integrable test function φ:

$$\hat{\eta}_{t-1}M_t(G_t\varphi) = \eta_t(G_t)\hat{\eta}_t(\varphi) \Rightarrow \hat{\eta}_{t-1}M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)]) = \eta_t(G_t)\underbrace{\hat{\eta}_t(\xi_t - \hat{\eta}_t(\xi_t))}_{=0}.$$

Substituting this representation into Equation (B.6),

$$\begin{aligned}\bar{T}_t^N(\xi_{0:t}) &= T_{t-1}^N(\xi_{0:t-1}) + \frac{\sqrt{N}\beta_t}{\hat{\eta}_{t-1}^N M_t(G_t)}\{(\hat{\eta}_{t-1}^N - \hat{\eta}_{t-1})M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)])\} \\ &= T_{t-1}^N\left(\xi_{0:t-2}, \xi_{t-1} + \frac{\beta_t}{\beta_{t-1}}\frac{M_t(G_t[\xi_t - \hat{\eta}_t(\xi_t)])}{\hat{\eta}_{t-1}^N M_t(G_t)}\right) \qquad \text{(B.11)}\end{aligned}$$

The proof is completed by using the result [37, sec. 7.4.3], that if G_t is essentially bounded below then,

$$\frac{1}{\hat{\eta}_{t-1}^N M_t(G_t)} \xrightarrow{P} \frac{1}{\hat{\eta}_{t-1}M_t(G_t)}$$

together with the induction hypothesis. ∎

LEMMA B.2. *Under the conditions of Proposition B.1,*

$$\lim_{N\to\infty} \chi_t^N(\xi_t) \xrightarrow{D} \mathcal{N}(0, \hat{V}_t(\xi_t)) \qquad \text{(B.12)}$$

where

$$\hat{V}_t(\xi_t) = \beta_t^2\hat{\eta}_t((\xi_t - \hat{\eta}_t(\xi_t))^2) \qquad \text{(B.13)}$$

Proof. Consider first the particle system before reweighting with the potential function G_t:

$$\sqrt{N}\beta_t\sum_{j=1}^N \frac{\xi_t(\tilde{X}_t^{(j)}) - \hat{\eta}_{t-1}^N M_t(\xi_t)}{N} = \sum_{j=1}^N U_{t,j}^N \qquad \text{(B.14)}$$

where $U_{t,j}^N = \frac{\beta_t}{\sqrt{N}}\{\xi_t(\tilde{X}_t^{(j)}) - \hat{\eta}_{t-1}^N M_t(\xi_t)\}$. Define, recursively, the σ-algebras $\mathcal{H}_t^N = \mathcal{H}_{t-1}^N \vee \sigma(\{\tilde{X}_t^{(j)}\}_{j=1}^N)$, $\mathcal{H}_{t-1} = \sigma\left(\cup_{N=0}^{\infty}\mathcal{H}_{t-1}^N\right)$ and the increasing (in j) sequence of σ-algebras $\mathcal{H}_{t,j}^N = \mathcal{H}_{t-1} \vee \sigma(\{\tilde{X}_t^{(l)}\}_{l=1}^j)$. It is clear that

$$\mathbb{E}[U_{t,j}^N|\mathcal{H}_{t,j-1}^N] = \mathbb{E}[U_{t,j}^N|\mathcal{H}_{t-1}] = 0 \qquad \text{(B.15)}$$

and so the sequence $U_{t,j}^N$, $j = 1, \ldots, N$ comprises a collection of $\mathcal{H}_{t,j}^N$-martingale increments. Further it can be verified that these martingale increments are square integrable,

$$\begin{aligned}\mathbb{E}[(U_{t,j}^N)^2|\mathcal{H}_{t,j-1}^N] &= \mathbb{E}[(U_{t,j}^N)^2|\mathcal{H}_{t-1}] \\ &= \frac{\beta_t^2}{N}\{\hat{\eta}_{t-1}^N M_t(\xi_t^2) - [\hat{\eta}_{t-1}^N)(\xi_t)]^2\} < c_t\frac{\beta_t^2}{N}\end{aligned}$$

where $c_t < \infty$ exists by the boundedness of ξ_t. The conditional Linderberg condition is also clearly satisfied. That is, for any $0 < u \leq 1$ and $\varepsilon > 0$,

$$\lim_{N\to\infty} \sum_{j=1}^{\lfloor Nu \rfloor} \mathbb{E}[(U_{t,j}^N)^2 \mathbb{I}_{(\varepsilon,\infty)}(|U_{t,j}^N|)|\mathcal{H}_{t,j}^N] \xrightarrow{P} 0.$$

Thus we have

$$\begin{aligned}\sum_{j=1}^N \mathbb{E}[(U_{t,j}^N)^2|\mathcal{H}_{t,j-1}^N] &= \frac{\beta_t^2}{N}\sum_{j=1}^N \{\hat{\eta}_{t-1}^N M_t(\xi_t^2) - [\hat{\eta}_{t-1}^N M_t(\xi_t)]^2\} \\ &= \beta_t^2\{\hat{\eta}_{t-1}^N M_t(\xi_t^2) - [\hat{\eta}_{t-1}^N M_t(\xi_t)]^2\}\end{aligned}$$

and we can invoke the martingale central limit theorem [146, pp. 543],

$$\lim_{N\to\infty} \chi_t^N(\xi_t) \xrightarrow{D} \mathcal{N}(0, \check{V}_t(\xi_t)) \tag{B.16}$$

where the asymptotic variance, $\check{V}_t(\xi_t)$, may be written as the limit of the sequence defined by

$$\check{V}_t^N(\xi_t) = \beta_t^2\{\hat{\eta}_{t-1}^N M_t(\xi_t^2) - [\hat{\eta}_{t-1}^N M_t(\xi_t)]^2\} \tag{B.17}$$

and as (again, see [37, sec. 7.4])

$$\check{V}_t^N(\xi_t) \xrightarrow{P} \beta_t^2\{\hat{\eta}_{t-1} M_t(\xi_t^2) - [\hat{\eta}_{t-1} M_t(\xi_t)]^2\}$$

the proof is completed using Slutzky's lemma and applying [31, Lemma A2] which yields that:

$$\lim_{N\to\infty} \mathcal{X}_t^N \xrightarrow{D} \mathcal{N}(0, \hat{V}_t(\xi_t))$$

with

$$\begin{aligned}\hat{V}_t(\xi_t) &= \check{V}_t\Big(\frac{G_t(\cdot)}{\hat{\eta}_{t-1}M_t(G_t)}(\xi_t(\cdot) - \hat{\eta}_t(\xi_t))\Big)\\ &= \beta_t^2\hat{\eta}_t\Big(\frac{G_t(\cdot)}{\hat{\eta}_{t-1}M_t(G_t)}(\xi_t(\cdot) - \hat{\eta}_t(\xi_t))\Big)\end{aligned}$$

∎

Lemma B.3. *Under conditions of Proposition B.1, and the inductive assumption of Lemma B.1, $T_t^N(\xi_{0:t})$ is asymptotically normal with variance stated as in Proposition B.1.*

Proof. Consider the characteristic function,

$$\begin{aligned}\varphi(T_t^N(\xi_{0:t}))(s) &= \mathbb{E}[\exp(isT_t^N(\xi_{0:t}))]\\ &= \mathbb{E}[\exp(is\bar{T}_t^N(\xi_{0:t}))\exp(is\chi_t^N(\xi_t))]\\ &= \mathbb{E}[\exp(is\bar{T}_t^N(\xi_{0:t}))\,\mathbb{E}[\exp(is\chi_t^N(\xi_t))|\mathcal{H}_{t-1}^N]]\\ &= \mathbb{E}[(A_t - \exp(-s^2\hat{V}_t(\xi_t)/2))B_t] + \exp(-s^2\hat{V}_t(\xi_t)/2)\,\mathbb{E}[B_t]\end{aligned}$$

where $A_t = \mathbb{E}[\exp(is\chi_t^N(\xi_t))|\mathcal{H}_{t-1}^N]$ and $B_t = \exp(is\bar{T}^N(\xi_{0:t}))$. The first term can easily be shown to converge a.s. to zero as $N \to \infty$ by the asymptotic normality of ξ_t^N and the conditional independence of the particles at iteration t given $\mathcal{H}_{t-1}^N$. The second term is the product of two Gaussian characteristic functions and thus we have that T_t^N also follows a Gaussian distribution (see detail in Lemma 10 in [93], for details). ∎

Using Lemma B.1 to B.3, the proof of Proposition B.1 follows by mathematical induction and a trivial base case (the first iteration is simple importance sampling).

C vSMC: A C++ LIBRARY FOR PARALLEL SMC

C.1 CLASSES OF PARALLEL COMPUTERS

There are a few types of parallel computers. Here we introduce the four types of hardware parallelism that are most commonly seen. Parallel computers can be nested. In a multicore CPU, each core can perform instruction level parallelism. On the other hand, a distributed system can be formed by multiple multicore CPUs.

C.1.1 *Instruction level*

Modern CPUs all implement the so called SIMD instructions, short for *single instruction, multiple data*. The CPU can execute a single instruction on different data in a single cycle. However, unlike the higher level parallelism discussed later, SIMD often has strict requirement on the arrangement of the data. In addition, the implementation often requires using low level assembly language or intrinsics functions.

Though vSMC does not directly implement this level of parallelism, it can be used by the user nonetheless. In addition, many operations within vSMC can be performed using libraries that are implemented with SIMD parallelization, such as Intel MKL. Also note that, most modern C++ compilers perform SIMD optimizations on simple loop and some of them, such as Clang [156] performs SIMD optimizations for non-loop structures. This kind of optimization is also called *vectorization*.

C.1.2 *Multicore processors and symmetric multiprocessing*

In the late 1990s, computer CPUs are advanced by increasing the clock speed. However, this strategy soon hit some bottlenecks, mainly the control of heat and power. The industry started to develop multicore processors. Each CPU has several cores, each running at a modest clock rate. By executing different threads on different cores, the CPU can process the same amount of work with less time without increasing the clock rate.

When a computer has multiple CPUs and each of them has the same speed to access the memory, the system is often called *symmetric multiprocessing* (SMP). Most higher end workstations are SMP systems. The programming tools are usually the same for SMP and multicore processors.

The vSMC library support various SMP programming models. In addition, vSMC allows the same user implementation source code to be compiled into different parallel samplers using different programming models.

C.1.3 *Distributed computing*

Distributed computing usually refers to the form of computing where both memory and computing processors are spread among computing nodes. It can take different forms, such as grids and clusters. The *de facto* programming model for distributed computing is MPI. This is also supported by vSMC. In addition, the library also allows easy integration of MPI and various SMP programming models.

C.1.4 *Massive parallel computing*

In recent years, there is a new trend of using specialized massive parallel devices, such as GPUs for scientific computing. Modern GPUs often have hundreds or thousands co-processors. The main difference between GPU and CPU is that, CPU has more logic control units, and thus is more suited for general programs. In contrast, GPU are better at applying the same arithmetic operations on a collection of data. It performs the best if each computing unit are executing *exactly the same* instructions. In addition, it is often much more efficient if there are a large amount data to be processed. Another significant feature of these devices is that they provide much higher *local* memory bandwidth and can use various technologies to reduce local data latency than traditional CPU.

Massive parallel computing is extremely suitable for the SMC algorithms, which can have a large number of particles, while each of them needs to be updated using the same MCMC kernel.

There are two major programming models for general purpose GPU programming (GPGPU), Nvidia's CUDA framework and the OpenCL [101] standard. The vSMC library provides direct support for the OpenCL programming model.

C.2 PARALLEL PATTERNS

In a more micro level, parallelism can be implemented with different patterns. The term *pattern* in computer science, introduced and popularized by [50], is a way of codifying best practices for software engineering. We found patterns are more useful to statisticians for reasoning the parallel structure of a given algorithm. This section is not an exhaustive discussion of parallel patterns. Instead, we choose some of the most commonly seen in practice, in particular those relevant to Monte Carlo algorithms.

C.2.1 *Map*

This is perhaps the simplest form of parallelism. A function, called *elemental function*, is replicated for each element of a data collection concurrently. The elemental function must have no side-effects in order for the map to be implementable in parallel while achieving deterministic results. In particular, it cannot modify global data that other instances of that function depend on.

In SMC algorithm, the updating of particle values is clearly implementable using a map pattern. The operation of the kernel $K(x_{t-1}, x_t)$ depends only on the history of the particle that it will be used to update, but not other particles at a given generation.

C.2.2 *Fork-join*

This pattern lets control flows fork into multiple parallel flows that rejoin later. The major difference between fork-join and map is that fork-joint does not necessarily apply the same function on different data. Instead, usually different functions are applied to different or the same data. There are different programming models that implement this pattern. The OpenMP `parallel region` fork control into multiple threads that all execute the *same* statements and use other constructs to determine which thread does what. The Intel Cilk Plus [79] `spawn` fork a new thread to execute the calling function on a new thread and it is later joined with the callee.

The fork-join pattern are often used by programming models to implement other patterns and is widely used in practice itself. One example is numerical integrations, especially for adaptive schemes. Whenever a new segment of the integral interval is chosen, the program can fork a new thread to compute the results. And after all segments are computed, the program can join all threads and sum up the final result.

C.2.3 *Reduction*

This pattern uses an associative operator to combine every element in a collection into a single element. Given the associativity of the operator, many different orderings are possible and hence multiple threads can be used to parallelize the computation. This is most often used for parallelization of computations such as summations.

For example, the computation of ESS, CESS, normalizing of weights, etc., are all parallelized using the reduction pattern within vSMC.

C.2.4 *Pipeline*

A pipeline connects tasks in a *producer-consumer* relationship. A few computation units are active at the same time. The first one consume the data, and produce new data to be used by the second, and so on. As the data flows into the pipeline, each unit has its own work to do and thus computations are carried out in parallel while the data dependencies are correctly maintained.

There are several applications of pipeline in Monte Carlo computing. For example, an MCMC algorithm often needs to compute various convergence statistics, say $h(X_{0:t})$. Often, this statistic can be written as $h(X_{0:t}) = h(X_t, h(X_{0:t-1}))$. Instead of compute it after all iterations, one can use one thread to update the MCMC chain and another one to compute the statistics, using the pipeline pattern. In this case, the Markov kernel that update the states is the *producer* and the thread that update the statistics is the *consumer*.

C.3 MODERN C++

The C++ programming language [152] was first created to support object-oriented programming (OOP) on top of the C programming language [153]. The features, such as templates, come to the language fairly late. However, it was found that the C++ template feature provides a complete sub-language [161]. This leads to various new metaprogramming techniques. Many of them are documented in [4] and characterize the modern usage of C++. In this section, we introduce two of these techniques. They are widely used inside the vSMC library and the contents here should ease the reading of the following sections for those less familiar with them. However, we assume the reader has at least some working knowledge of C++, including concepts such as OOP.

C.3.1 *Templates*

C++ is a static strong type language. It requires the user to declare variables, functions, and most other kinds of entities using specific types. However, a lot of code looks the same for different types, especially for implementation of algorithms. The C++ template technique allows one to write generic code to solve a class of problems, while the involved types can be seamlessly replaced at compile time.

Templates are useful for a few reasons. It reduces duplication of the same code for multiple types. Though conventional OOP also supports polymorphism behaviors, they rely on runtime decisions. In contrast, templates rely on compile time decisions and are more type safe. Templates emphasize that the same operation can be applied to many types. It allows unlimited extension of existing functionality. It is possible to have any combination of the allowed operations on a certain type and the allowed types of a certain operations. More specifically, given a collection of operations and a collection of types, each operation may support any subset of the types and each type can support any subset of the operations. Such combinatorial behavior is difficult to implement using conventional OOP, where a collection of types have a common interface that provides a fixed set of operations.

There are two main types of templates in C++, function template and class template.

Function template

The following lines define a simple function template,

```
template <typename T>
inline T max (const T &a, const T &b)
{
    return a < b ? b : a;
}
```

This template definition specifies a *family* of functions that returns the maximum of two values, which are passed as function parameters `a` and `b`. The type of these parameters is left open as *template parameter* `T`.

In this template definition, the assumption about `T` is that the operator `<` is properly defined. It does not matter whether `T` is a fundamental type or a class type with this operator defined by the user. The actual types are deduced at compile time when the function template is used.

Class template

Class template is similar to function template. A class template define a family of classes. For example,

```
template <typename T>
class Stack
{
    public :

    void push (const T &val) {elems_.push(val);}
    void pop () {elems_.pop_back();}
    T top () const {return elems_[0];}
    bool empty () const {return elems_.empty();}

    private :

    std::vector<T> elems_;
};
```

defines a `Stack` class template. For simplicity, some edge cases and exceptional situations such as calling `top` on an empty `Stack` is not handled here. This class template can be used as the following,

```
Stack<std::string> sstack;
sstack.push_back("foo");
sstack.push_back("bar");
sstack.pop();
if (!sstack.empty())
    std::cout << sstack.top() << std::endl;
```

As we can see, to use a class template, one *explicitly* supply the type of the template parameter. Unlike function template, there is no template parameter deduction here.

C.3.2 *Callable objects*

vSMC is a framework for constructing generic SMC samplers. It relies on the user to write callback functions to perform application specific operations, such as updating particles. In this section, we introduce the few forms of callback that are supported by the library. Collectively, they are also called *callable objects*, meaning that they support the function calling syntax though they may not be functions.

A callable object, say `callable`, is similar to a function in the sense that it has a return type and a parameter list as its signature. It can be used with the syntax,

```
callable( /* arguments */ );
```

However, the object may or may not be a function. There are three ways to define a callable objects, function pointer, functor and C++11 lambda expression. Function pointer is the main way of passing callback in C. The other two are introduced later.

The library also use type erasures, introduced later in this section, to enforce certain interfaces. The benefits of techniques introduced below increases the productivity and flexibility of the library compared to conventional techniques of passing callback through function pointer.

Functors

Consider the simple problem, sort a vector $\{x_i\}_{i=1}^N$ according to the values $y_i = f(x_i)$. We may define a function template to solve this problem,

```
template <typename F>
void sort_f (std::size_t N, const double *input,
        double *output, const F &f)
{
    for (std::size_t i = 0; i != N; ++i)
        output[i] = f(input[i]);
    std::sort(output, output + N);
}
```

The function template `sort_f` expects input and output as pointers. In addition, it expects a callable object `f`, which accepts a variable of type `double` as its input and return a number that can be assigned to a variable of type `double`.

One way to define such a callable object is to use *functor*, a class type with `operator()` properly defined. For example,

```
struct F
{
    double operator() (double x) const { return x * x; }
};
sort_f(input, output, F());
```

Here we created this object in the function call of `sort_f`. It can also be used as,

```
F f;
double y = f(3); // y <- 9
```

Lambda expressions

Another way, introduced in C++11, is a new feature called *lambda expression*. It is also called *local function* or *closure* in other programming languages. It allows us to define callable object *on site*. Here is an example,

```
sort_f(input, output, [] (double x) { return x * x; });
```

The full declaration of a lambda expression is as the following,

```
[ /* capture */ ] ( /* parameters */ ) /* mutable */
/* exception specification */
/* attribute specification */
-> /* return type */ { /* body * };
```

The `/* mutable */` part can be either empty or the keyword `mutable`, which allows the body to modify captured parameters (explained soon). The exception specification is similar to a normal function and the attribute specification is a new feature for all functions in C++11, that specifies things like parameter passing conventions among other things, which we will not go into details. The part `-> /* return type */` specifies the return type of the lambda expression. If omitted, it is deduced from the body. And if the body does contain any `return` statement, it is deduced to be `void`. If the expression takes no arguments, the parameter list can also be omitted.

The `/* capture */` specifies which symbols visible at the scope of the definition of the lambda expression will be visible inside the body. There are a few forms,

`[a, &b]` captures `a` by value and `b` by reference.

`[this]` captures the `this` pointer by value.

`[=]` captures all automatic variables *used* in the body by value.

`[&]` captures all automatic variables *used* in the body by reference.

`[]` captures nothing.

Type erasures

In the example above, the function template sort_f does not actually enforce the signature of the function or functor, in contrast to the definition of it that takes a function pointer as an argument. For example, the following is perfect valid C++,

```
int h (int x) { return x * x };
sort_f(input, output, &h);
```

while it may not be what one wants. The use of h with sort_f is perhaps an typo. The type erasure in C++11, std::function, provides a solution to this problem. A *type erasure* can convert various types of objects into a single type. Below is a basic usage of std::function,

```
#include <functional> // the header that defines std::function

std::function<double (double)> f;
F f_obj;
f = f_obj;  // Correct
f = &h;     // ERROR: h does not has the required signature.
```

Now we can redefine the function sort_f as,

```
double sort_f (std::size_t N, const double *input,
        double *output,
        const std::function<double (double)> &f)
{ /* same as before */ }
```

The vSMC library makes extensive use of the type erasure to enforce certain callback interfaces. When C++11 features are not available, the Boost library provides the same functionality through boost::function. See [166] for details of how vSMC choose between C++11 and Boost libraries.