Computational Bayesian Methods for Communications and Control

Soren John Henriksen, BE(Hons), ME

February 2012

A thesis submitted to embody the research carried out to fulfil the requirements for the degree of:

Doctor Of Philosophy
in Electrical Engineering
at The University of Newcastle
New South Wales, Australia
Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text. I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying subject to the provisions of the Copyright Act 1968.

Soren John Henriksen, BE(Hons),ME

February 2012
Acknowledgements

I would like to thank Prof. Brett Ninness and A/Prof. Steve Weller for their supervision and invaluable help in the work presented here. Also of great assistance have been my fellow postgraduate students and colleagues at the Signal Processing Microelectronics group. Finally, I must thank my family and friends for their perseverance through the duration of this project.
Contents

Abstract xiii

1 Introduction 1
   1.1 Overview 2
   1.2 Key Contributions 3
   1.3 Publications 4

2 Stochastic Sampling 5
   2.1 Introduction 5
   2.2 Characterising Stochastic Models 5
      2.2.1 Possible Implementation Techniques 7
   2.3 Integration Methods 8
      2.3.1 Uniform Sampling 8
      2.3.2 Polynomial Fitting 9
      2.3.3 Random Sampling 9
      2.3.4 Importance Sampling 11
      2.3.5 Volume of a Ball 13
      2.3.6 Weighting by Target Distribution 14
   2.4 Sampling Methods 16
      2.4.1 Inversion 16
      2.4.2 Rejection Sampling 17
      2.4.3 Composition 18
      2.4.4 Ratio of Uniforms 19
      2.4.5 Squeezing 19
   2.5 Laws of Large Numbers 20
   2.6 Markov Chain Monte-Carlo Methods 21
      2.6.1 Markov Chains 22
      2.6.2 The Metropolis Algorithm 23
      2.6.3 Metropolis-Hastings 24
   2.7 Bootstrap Resampling 25
   2.8 Sequential Monte Carlo 26
      2.8.1 Weighting 28
4.2.1 Suboptimal Demodulation ........................................ 60
4.3 Multiple Access Channels ........................................... 61
  4.3.1 CDMA Signal Model ........................................ 62
  4.3.2 Matched Filter Model over Additive White Gaussian Noise Channel 63
  4.3.3 Matrix Form ........................................... 64
4.4 Detection Decision Schemes ........................................ 65
  4.4.1 Maximum Likelihood .................................... 65
  4.4.2 Maximum a Posteriori .................................. 66
4.5 CDMA Detection Methods ........................................... 66
4.6 MCMC Methods ................................................... 67
  4.6.1 Metropolis–Hastings .................................... 68
  4.6.2 Gibbs Sampling ......................................... 69
4.7 MCMC Implementation Details .................................... 71
  4.7.1 MU Detection via the Gibbs Sampler ..................... 71
  4.7.2 Multi–User Detection via the Metropolis Algorithm ....... 72
  4.7.3 Metropolis and Gibbs Sampling Compared ................. 74
4.8 Convergence Analysis ............................................. 74
  4.8.1 Strong Law of Large Numbers .......................... 75
  4.8.2 Limiting Distribution ................................... 78
4.9 Simulation Study ................................................... 81
  4.9.1 Single-User Bound ..................................... 81
  4.9.2 Algorithm Performance ................................ 82
4.10 Hold-Up Effects .................................................. 83
  4.10.1 High SNR Solution .................................... 85
  4.10.2 Convergence of Modified Algorithm .................... 85
  4.10.3 Maintaining Soft Outputs ............................... 86
  4.10.4 Performance of the Modified Algorithm ................ 87
  4.10.5 Soft-Output Performance ............................... 89
4.11 Convergence Rate Analysis ....................................... 89
  4.11.1 Metropolis–Hastings with Independent Proposal ........ 90
  4.11.2 Independent Proposal Choice ........................... 96
4.12 Computationally Efficient Method ................................ 99
4.13 Conclusions ....................................................... 101

5 Application in System Identification .......................... 103
5.1 Introduction ....................................................... 103
5.2 Systems Models .................................................. 103
  5.2.1 General Probabilistic Model ............................ 104
  5.2.2 State Space Models .................................... 105
  5.2.3 Transfer Function Models ............................... 106
5.3 Parameter Estimation ............................................. 107
## 5.3 Maximum Likelihood

- 5.3.1 Maximum Likelihood ...................................................... 107
- 5.3.2 Prediction Error Decomposition ........................................... 108
- 5.3.3 Asymptotic Convergence .................................................... 110
- 5.3.4 Quantifying Estimation Error ................................................ 110
- 5.3.5 Error in Functions of Parameters ............................................ 111

## 5.4 A Bayesian Approach to Estimation and Error Quantification ............ 113

## 5.5 A Markov Chain Monte–Carlo Solution ....................................... 114

- 5.5.1 MCMC Sampler over Uncountable Spaces ................................. 116

## 5.6 Markov Model ................................................................. 118

## 5.7 MCMC Convergence ............................................................ 120

- 5.7.1 Invariant Density ............................................................. 120
- 5.7.2 Distributional Convergence ................................................... 122
- 5.7.3 Sample Averages .............................................................. 124

## 5.8 Convergence Rate ............................................................. 127

- 5.8.1 Convergence of Sample Averages ............................................. 130
- 5.8.2 Choice of Proposal Distribution ............................................. 131

## 5.9 Conclusion ................................................................. 132

## 6 System Identification Implementation ........................................ 133

### 6.1 Simulation Study ............................................................ 134

- 6.1.1 System Model .............................................................. 134
- 6.1.2 Metropolis–Hastings Algorithm ............................................. 136
- 6.1.3 First-Order System Example ................................................ 137
- 6.1.4 Marginal Density Estimation ................................................ 137

### 6.2 Density Function Representation ............................................ 139

### 6.3 Convergence Behaviour ..................................................... 142

- 6.3.1 Density norms .............................................................. 142
- 6.3.2 Convergence of First-Order System ........................................ 142
- 6.3.3 Effect of Smoothing on Convergence ....................................... 144

### 6.4 Comparing the Asymptotic Normal Estimate .................................. 146

---

### 6.5 Non-Gaussian Noise .......................................................... 147

- 6.5.1 Uniform Measurement Noise ................................................ 147
- 6.5.2 Truncated Gaussian Measurement Noise ................................... 149

### 6.6 Measurement Noise Estimation ............................................... 150

- 6.6.1 Gaussian Example .......................................................... 151
- 6.6.2 Uniform Example ............................................................ 153

### 6.7 MCMC Error Bounds .......................................................... 153

### 6.8 Proposal ................................................................. 154

- 6.8.1 Random Walk .............................................................. 156
- 6.8.2 Random Walk Variance ..................................................... 156
- 6.8.3 Adaption ................................................................. 158
Abstract

As available computing power increases, there are now opportunities to use computational numerical methods to solve engineering problems that were once intractable. This thesis presents the application of stochastic simulation methods to areas of telecommunications and system identification.

While there are many randomised algorithms available for solving problems of optimisation and integration, the focus in this work is on those based on Markov chain Monte–Carlo methods, for which it is possible to prove convergence results. This thesis provides an introduction to the Markov chain theory that is used in this area, through to the point of proving convergence properties of the algorithms developed and showing the conditions required.

A method is presented for multi-user detection in code division multiple-access communications systems. The Metropolis algorithm is used to build a soft-input, soft-output detector, which provides the maximum a posteriori estimate of the symbols sent, together with the probability of error. This demonstrates the ability of stochastic algorithms to be used in high speed applications.

For the second application of this thesis, the parameter estimation of dynamic systems models is considered. Not only does this provide a means of obtaining maximum a posteriori estimates of parameters in nonlinear and non-Gaussian model structures, but also provides full probability density functions of the parameters given the observed measurements. By additionally incorporating the Particle Filter, a wide class of model structures may be used. The steps required to achieve this in a computationally efficient manner are described, including a parallel implementation using Graphic Processing Units.
Chapter 1

Introduction

Through the 1940s and 1950s, there was considerable effort expended in finding numerical solutions to difficult mathematical problems. Under the wartime imperative, researchers at Los Alamos were researching numerical solutions to model the dynamic behaviour of nuclear detonation[21], and conveniently at the same time, the first general–purpose electronic computers were being constructed and employed on this task. By 1953, a team led by Nicholas Metropolis devised an algorithm to randomly sample configurations of particles according to a Boltzmann distribution[34]. This algorithm became known as the Metropolis Monte Carlo method, and has since found application in many other areas of science, engineering and economics.

The Metropolis algorithm is one instance of the broader class of stochastic simulation methods. Within these methods, deterministic results are estimated by evaluating functions of samples drawn from specially crafted random number generators. This approach is often chosen on account of the direct numerical computation of the result being computationally intractable for the problem at hand.

Since the birth of the stochastic sampling methods in the 1950s, computing power has increased dramatically. While this means that the original problems posed can now be directly computed numerically, it also opens up new problems to the stochastic methods. Estimation problems that even a decade ago did not have a solution are now within the reach of effective estimation through stochastic simulation approaches. In the area of systems and control, they have the potential to open up the use of model structures and control strategies that have previously been infeasible to work with.

An objection raised against randomised computation methods is that the result is not reliably correct, on account of the random process by which it was obtained. However, given sufficient algorithm iterations, for many practical problems the sample averages of the Metropolis Monte Carlo method can be proven to eventually converge to the true result. The proof process is facilitated by the observation that the Metropolis algorithm may be modelled as a Markov chain, and consequently the extensive theoretical framework developed for Markov chains may be applied.

This thesis commences by presenting a background on stochastic simulation meth-
ods and the operation of the associated algorithms. Following this, the Markov chain theoretical framework is established, later to be used as the underpinnings for algorithm convergence proofs and convergence rate analysis. Following this, particular applications of the general Metropolis algorithm are developed for applications in multi-user detection, and in system identification.

Two distinct applications of the Metropolis algorithm are presented in this thesis in order to highlight the similarities and differences among the range of applications. The first example involves a Markov state over a discrete space, while the second is over an uncountable space. There are differences in how these must be treated, from both a theoretical and a practical standpoint, but there are also common traits in implementation and performance.

Much of the current theory and practice in the area of system identification is based on an assumption of relatively limited computer processing capability. Many of the areas of research originate from a decade or more ago, when computing capabilities were far less, and it was necessary to find closed-form solutions to computational problems. However now that numerical operations are cheap, it is a natural question to ask whether this can be used to extend the capabilities of the system identification field.

In recognising that the form of parameter estimation methods ought to reflect the computational resources available, future trends of computing are also considered. With the difficulties present in further increasing processor clock speeds, there is an expected future direction toward greater parallel computing. This is evidenced by the recent trend toward greater core count in microprocessors. Graphics Processing Units (GPU) are also demonstrating the huge computational power that may be available to algorithms that are amenable to parallel execution. The final part of this thesis demonstrates the natural fit that stochastic simulation methods have to parallel computing.

1.1 Overview

The body of this thesis is composed of five chapters. The contents of these are:

- Chapter 2 describes the use of stochastic simulation as a means of computing the computationally intensive integrals needed in parameter estimation tasks.

- In Chapter 3 some basic theoretical results for Markov chains are presented, over both countable and non-countable spaces.

- Following the presentation of the Markov chain theory, a concrete application of parameter estimation is presented in the area of telecommunications, and an MCMC (Markov Chain Monte Carlo) algorithm over countable spaces is developed to perform multi-user detection.

- Chapter 5 extends the use of the Metropolis–Hastings algorithm to non-countable spaces, with an application in dynamic system identification, and demonstration of
algorithm convergence.

- The final chapter presents the results of using the Metropolis–Hastings algorithm for system identification tasks, and showing methods to overcome issues of the computational costs of these algorithms.

1.2 Key Contributions

- This thesis brings together a complete picture of the Markov chain theory needed to prove the convergence of algorithms based on the Markov chain Monte–Carlo methods. While this theory already exists, some components of needed theory are not easily accessible for people not specialist in the discipline.

- The proof of Harris recurrence in Theorem 5.7.2 fills an apparent gap in the literature.

- Chapter 4 presents insights into the comparative behaviours of the Metropolis–Hastings algorithm and the Gibbs Sampler, with some conclusions on this in Section 4.7.3.

- The performance and computational burden of MCMC applied to multi-user detection has been shown in Section 4.9.

- In Section 4.10 a basis is found for problems with stochastic methods in high signal-to-noise ratio detection problems. A modified algorithm involving importance sampling is also presented to overcome this problem.

- By constructing a hypothetical algorithm in Section 4.11, intuition from convergence results obtained for independence chains are applied to the practical case.

- A method was developed to greatly improve the per-iteration execution times for certain applications of the Metropolis algorithm. (Section 4.12)

- It has been shown in Chapter 6 how MCMC methods may be used to provide parameter estimates not currently available by other means, but with relatively short computational time.

- Insight is shown into the important aspect of the choice of proposal distribution in the Metropolis–Hastings algorithm (Section 6.8)

- In Section 6.8.6, a new proposal adaption scheme to independently scale multiple variables is described.

- The required execution time for system identification MCMC tasks is profiled, including the dependence on system model order.
• It is shown in Section 6.11 how the Metropolis–Hastings algorithm may be coupled with the Kalman Filter and the Particle filter to extend the scope of model structures that may be identified.

• In Section 6.13, a new parallel implementation of a combined particle filter and MCMC algorithm is developed to take advantage of the computational resources of a Graphics Processing Unit.

1.3 Publications

The following publications are a direct result of the research leading to this thesis:


• Soren John Henriksen, Adrian Wills, Thomas Schon, and Brett Ninness. Parallel implementation of particle MCMC methods on a GPU. In *Sysid 2012*, Jul 2012.


Chapter 2

Stochastic Sampling

2.1 Introduction

In order to set the background for the use of computational Bayesian methods, this chapter considers the sampling from sequences of random numbers. Initially consideration is given to the task of estimating parameters of a stochastic model based on observed samples. This task is shown to fundamentally depend on integration over a potentially high-dimensional space. Sampling methods are then introduced as a means to compute this integration problem, and in particular, Markov chain Monte–Carlo methods are explained as a means of drawing samples from the desired distribution.

2.2 Characterising Stochastic Models

Consider the stochastic process shown in Figure 2.1. The system $G_\theta$ produces a sequence of observed output values $Y = \{y_1, ..., y_N\}$, with each $y_i \in \mathbb{R}$.

$$G_\theta \rightarrow y_i$$

*Figure 2.1: Parameterised stochastic system.*

The output $Y$ is considered to be drawn from a random process with a defined probability density function,

$$Y \sim p(Y \mid \theta). \quad (2.1)$$

The model of the system $G_\theta$ is explicitly parameterised by a parameter vector $\theta \in \mathcal{X}$. Given the model structure, the value of $\theta$ will determine the distribution of measured $Y$. For computational purposes, typically $\mathcal{X} \subseteq \mathbb{R}^n$.

After observing the output $Y$ drawn from (2.1), a common requirement is to form an estimate $\hat{\theta}$ of the underlying parameters in the model that generated the output. For
example, this may be a maximum likelihood estimate with,

\[ \hat{\theta} = \arg \max_{\theta} p(Y \mid \theta). \] (2.2)

However, such an estimate doesn’t provide any degree of certainty in the precision of the value of \( \hat{\theta} \) obtained. For this reason, it can be desirable to determine the marginal probability density for one component \( \theta^i \) of \( \theta \) as,

\[ p(\theta^i \mid Y). \] (2.3)

According to Bayesian principles\[4\], this posterior density function contains all of the available knowledge that can be deduced of \( \theta^i \) given the observed outputs, together with any prior knowledge of \( \theta \).

More generally, rather than characterising individual parameters, the desired probability density may be of an arbitrary function \( f : \mathcal{X} \rightarrow \mathcal{W} \) of the parameter vector \( \theta \),

\[ p(f(\theta) \mid Y). \] (2.4)

While such probability density functions would clearly be desirable to obtain, it isn’t immediately obvious how they may be calculated. In the cases where it is possible to evaluate \( p(Y \mid \theta) \), the probability density of (2.1), for a given \( Y \) and \( \theta \), the conditional density \( p(\theta \mid Y) \) may be computed via Bayes’ rule,

\[ p(\theta \mid Y) = \frac{p(Y \mid \theta)p(\theta)}{p(Y)}. \] (2.5)

Here, \( p(\theta) \) represents the a-priori probability density of \( \theta \), and \( p(Y) \) is the probability density of the observed output \( Y \) independent of the parameter vector \( \theta \). While not known a-priori, \( p(Y) \) may be computed by integrating across all possible values of \( \theta \) with,

\[ p(Y) = \int p(Y \mid \theta)p(\theta)d\theta. \] (2.6)

Fortunately, this integration is not necessary in practice, as being constant with respect to \( \theta \), it may be treated as a normalising factor and obtained by integration instead over a lower dimensional space.

In order to calculate a marginal probability density, or the probability density of any function \( f(\theta) \), inherently an integration is required over the so-called nuisance parameters\[41, p71\]. In the general form it may be computed as,

\[ \mathbf{P}(f(\theta) \in A \mid Y) = \frac{1}{p(Y)} \int p(Y \mid \theta)\mathbf{1}(f(\theta) \in A)p(\theta)d\theta, \] (2.7)

where \( \mathbf{P}(f(\theta) \in A) \) is the probability that \( f(\theta) \) falls in some arbitrary set \( A \subset \mathcal{X} \), and \( \mathbf{1}(\cdot) \)
is an indicator function taking the value 1 when true, or 0 when false. The computation of
this probability necessitates an integral over the whole \( n \) dimensional parameter space \( X \),
where \( n \) is the size of the parameter vector \( \theta \). This demonstrates that the computation of
marginal probability densities is fundamentally an integration over a multi-dimensional
space.

2.2.1 Possible Implementation Techniques

While the marginal density integral (2.7) is across a potentially high dimensional space, it
could be argued that by exploiting the structure of the integrand, the dimensional order
of the integral could be reduced. Such a reduction could greatly improve the feasibility
of performing the integration.

To this end, in some situations, it may be possible to easily compute the bounds of a
set \( B \), which is the pre-image of \( A \) under \( f \), with

\[
B = \{ \theta \in X : f(\theta) \in A \}. \tag{2.8}
\]

For example, the region \( B \) for one component, \( P(f(\theta^i) \in A_j \mid Y) \) for some \( A_j \subset \mathbb{R} \), of a
marginal density calculation on the \( i \)th parameter will be,

\[
B = \{ \theta \in X : f(\theta^i) \in A_j \}. \tag{2.9}
\]

In this case, the integral need only be calculated over the set \( B \), with

\[
P(f(\theta) \in A \mid Y) = \frac{1}{p(Y)} \int_{\theta \in B} p(Y \mid \theta)p(\theta)d\theta. \tag{2.10}
\]

The most useful application of this is in computing marginal densities of parameters \( \theta^i \),
where the integral may be performed over a space of dimension \( n - 1 \) for \( \theta \) of dimension
\( n \). While this is an improvement, it isn’t sufficient to greatly influence the ability to
perform the integration.

In order to avoid computing the denominator multi-dimensional integral in (2.6),
it’s necessary to find another method of calculating \( p(Y) \). If computing an estimate of
a complete marginal probability density function, this simply involves normalising the
resulting density function.

In practice, this is achieved by computing a histogram. If the image \( X \) under \( f \) is
contained in the set \( A_c \), and \( A_c \) is partitioned into \( k \) disjoint subsets \( A_i \), then by the total
law of probability,

\[
\sum_i P(f(\theta) \in A_i \mid Y) = 1. \tag{2.11}
\]

When this is applied to the integration results from (2.7), the constant term \( p(Y) \) may
be deduced.
2.3 Integration Methods

Integration is a broad and well-studied area that incorporates many different approaches[14]. At first glance, we may contrast symbolic, or exact, methods against computational approximations. Symbolic methods have the advantage of requiring minimal numerical computation, and the promise of a precise result. However, for many applications there is simply no known method to form an exact anti-derivative function, particularly over higher dimensional spaces. Even in some cases where an exact expression is available, the numerical problems in evaluating that expression can involve greater loss of precision than what would occur using an approximate method[14, p17].

Exact solutions are not considered in this thesis, due to the unavailability of symbolic methods for the integrals encountered. However, in specialisations where a symbolic method is available, it may prove to be the most effective option.

There are many approaches available just in numerical integration. This section describes some basic techniques to estimate the integral

$$\int_{\mathcal{X}} f(x) dx,$$

with $\mathcal{X} \subseteq \mathbb{R}^n$ and $f : \mathcal{X} \rightarrow \mathbb{R}$. Additionally, $f$ is assumed to be bounded on $\mathcal{X}$, and $\mathcal{X}$ assumed to be an orthotope on $\mathbb{R}^n$ as,

$$\mathcal{X} = \prod_{i=1}^{n} [a_i, b_i].$$

Integration using the following methods may easily be extended across other subsets $\mathcal{X} \subseteq \mathbb{R}^n$, but this restriction is made to improve clarity.

2.3.1 Uniform Sampling

Perhaps the most intuitive integration technique is that of uniform sampling. This involves evaluating $f(x)$ at each of the points on a regular lattice spanning $\mathcal{X}$ and calculating a sum.

A crude method of evaluating the area of a unit circle could be to integrate the function,

$$f(x, y) = 1 \left( (x - \frac{1}{2})^2 + (y - \frac{1}{2})^2 < \frac{1}{2} \right),$$

across a unit square, as shown in Figure 2.2.

Here $f(x)$ is evaluated over 100 points evenly distributed across the region $[0, 1]^2$. The evaluation shows that 80 of the 100 points fall inside the circle, resulting in an estimate of $\int f(x) dx \approx 0.80$.

While uniform sampling is quite feasible in low dimensions, the number of function evaluations rises exponentially with the dimension. For $m$ points per dimension, the total
2.3. Integration Methods

number of evaluations is $m^n$. It may help to increase the lattice point spacing along some axes, but this increases the error in the approximation, and even for low values of $m$, $m^n$ grows very quickly for even a moderate number of dimensions\[14\].

2.3.2 Polynomial Fitting

In order to improve on the approach of sampling from a simple random grid, a polynomial function can be fitted to the evaluated points, and the integral of the polynomial computed. This results in the Newton-Cotes formulae, the best-known of which is for the 2nd order case, and is known as Simpson’s Rule. However for accurate modelling of functions that are not polynomial in shape, it becomes necessary to divide the region into a grid in much the same manner as the uniform sampling case. This results in a similar order of computational complexity as the dimension increases.

2.3.3 Random Sampling

In order to avoid some of the drawbacks of uniform grid sampling, the integration sampling points may be chosen at locations other than a uniform lattice. This may take the form of a non-uniform lattice, or of randomly chosen points.

The random sampling methods rely on the properties of the expectation of functions of a random variable. Consider the sampling points $x_i$ from independent realisations from a random variable $X$. The expected value of $f(X)$ is,

$$E\{f(X)\} = \int_{\mathcal{X}} f(x)p_X(x) \, dx.$$

(2.15)

If $X$ is uniformly distributed across $\mathcal{X}$, then for $x \in \mathcal{X}$, $p_X(x) = \tilde{p}$ is a constant, with

$$\int_{\mathcal{X}} p_X(x) \, dx = 1.$$

(2.16)
By defining a normalising constant \( K_X \) as
\[
K_X = \bar{p} = \frac{1}{\int_X dx},
\] (2.17)
the expected value of \( f(X) \) may then be expressed as
\[
E\{f(X)\} = K_X \int_X f(x)dx.
\] (2.18)
However, the expected value of \( f(X) \) may also be estimated via a sample mean from realisations of the sampler, with
\[
\hat{f}_k(x) = \frac{1}{k} \sum_{i=1}^k f(x_i).
\] (2.19)
By the strong law of large numbers (see Theorem 2.5.2), for sufficiently large \( k \), this will converge to the true expectation, and hence the true value of the integral,
\[
\lim_{k \to \infty} \frac{1}{k} \sum_{i=1}^k f(x_i) = E\{f(X)\}.
\] (2.20)
So with sufficient random samples, the integral of \( f(x) \) may be computed accurately via the expected value of the samples.

For the circle example, in order to compute the area of the circles, samples may be uniformly generated in the enclosing square, and the total proportion of those inside the circle counted.

Random sampling avoids the rigid \( m^n \) complexity of uniform deterministic sampling. However, in its place, the total number of samples, \( k \), that are needed may easily grow with \( k \propto e^n \). To illustrate this, the uniform lattice of Figure 2.2 is really just one possible realisation of the random sampling depicted in Figure 2.3. Both examples have the same number of points, and are equally likely realisations under uniform random sampling. In this situation, there is an advantage to random sampling, due to the symmetry of the
deterministic pattern, but in itself it doesn’t overcome the curse of dimensionality.

The convergence rate of this approach can be measured by the variance of the estimate itself. The variance of the samples in (2.19) is,

$$\text{var} \left( \tilde{f}(x) \right) = \frac{1}{k} \int \left( \tilde{f}(x) - E\{f(X)\} \right)^2 dx. \quad (2.21)$$

As the variance is proportional to $\frac{1}{k}$, the precision of the estimate only improves with $\frac{1}{\sqrt{k}}$. This indicates relatively slow convergence, so it is important that the constant term in (2.21) is not excessively large[40, p120].

Where the required number of samples is not known a priori, the quality of the estimate of the integral in (2.19) may be estimated via a sample variance calculation, and the total number of samples needed may be estimated by (2.21). One feature of random sampling approaches is that the variance of their estimates is a smooth function of the number of samples. This means they may be terminated early while still proving a useful, albeit less accurate result.

### 2.3.4 Importance Sampling

Considering the random sampling pattern in Figure 2.3, we could claim the samples near the centre of the circle, or near the corners of the square as “less valuable”. Through those regions, $f(x)$ is not changing, and so there isn’t such need for fine-grained sampling.

Importance sampling involves sampling non-uniformly across $\mathcal{X}$, and placing the samples in areas where they are of more benefit in improving the estimate. In this case, those areas are the localities where $f$ has non-zero value. By doing so, the accuracy of the estimate improves for a given number of samples, making importance sampling a variance reduction technique.

While this is a useful approach, there are two potential problems in applying this. One is that it may not be easy to draw samples from a distribution that has a useful shape. The other is that the evaluation of the integral from the expectation, as in (2.15), no longer follows. Instead, the effect of a weighting density $w$ needs to be considered. To do this, the expectation of a function $f$ of samples from two different densities $p(\cdot)$ and $w(\cdot)$, may be related with,

$$E_p[f] = \int_{\mathcal{X}} f(x)p(x)dx \quad (2.22)$$

$$= \int_{\mathcal{X}} \frac{f(x)}{w(x)}p(x)w(x)dx \quad (2.23)$$

$$= E_w \left[ \frac{fp}{w} \right] \quad (2.24)$$

for all densities $w(\cdot)$ such that $w(x) > 0 \ \forall x$. This relationship motivates the definition.
of a new function $f^*$ to be evaluated when sampling from $w(\cdot)$, defined as

$$f^*(x) \triangleq \frac{f(x)p(x)}{w(x)}. \quad (2.25)$$

In this integration example, $p(x)$ represents the uniform density across the region $\mathcal{X}$, and so the integral estimate is given by,

$$\int_{\mathcal{X}} f(x)dx = E_w \left[ \frac{f}{w} \right]. \quad (2.26)$$

This expectation may again be calculated as in (2.19) by appealing to the law of large numbers.

$$\hat{E}_w \left[ \frac{f}{w} \right] = \frac{1}{k} \sum_{i=1}^{k} \frac{f(x_i)}{w(x_i)}. \quad (2.27)$$

The terms collected in the summation are divided by a weighting term from the probability density of the importance distribution. Samples from areas where there is a lower probability of being drawn are weighted up by a corresponding amount.

Clearly for importance sampling to be effective, it’s necessary that it be both easy to draw independent samples from $w(x)$ and also to be cheap to evaluate the probability density function $w(x)$ for $x \in \mathcal{X}$. It’s also important that the support of $w$ contain the support of $f$, or there will be regions that simply cannot be sampled.

In order to compare the behaviour of random sampling to that of importance sampling, we can compare the variance of the samples obtained. For samples drawn $x \sim p(x)$ the variance is,

$$\text{var}_p(f(x)) = \int f(x)^2 p(x) \, dx - E_p[f]^2. \quad (2.28)$$

In comparison, the variance of $f^*$ when sampled with $x \sim w(x)$ may be calculated, and expressed in the form of (2.28) to obtain,

$$\text{var}_w(f^*(x)) = \int f^*(x)^2 w(x) \, dx - E_w[f^*]^2 \quad (2.29)$$

$$= \int \frac{f(x)^2 p(x)^2}{w(x)^2} w(x) \, dx - E_p[f]^2 \quad (2.30)$$

$$= \int \frac{f(x)^2 p(x)}{w(x)} p(x) \, dx - E_p[f]^2. \quad (2.31)$$

The difference between these two results may be taken to obtain the change in variance through implementing importance sampling,

$$\text{var}_w(f^*(x)) - \text{var}_p(f(x)) = \int f(x)^2 \left( \frac{p(x)}{w(x)} - 1 \right) p(x) \, dx. \quad (2.32)$$
In order to achieve a variance reduction, this expression must be negative. As all of the terms are positive, this is achieved by driving the \( \frac{p(x)}{w(x)} \) ratio towards zero. Given that \( p(x) \) is fixed, and in this case uniform, the design choice is to vary \( w(x) \). Being a probability density function, it is still necessary to observe that

\[
\int w(x) \, dx = 1,
\]

so \( w(x) \) cannot be arbitrarily large everywhere. Instead, \( w(x) \) must be chosen to be large where it is of most value to do so. By inspection of (2.32), this will happen precisely when \( f^2(x)p(x) \) is large. For the case when \( f(x) > 0 \), this motivates the idea of somehow choosing a sampler with a pdf of

\[
w(x) \propto f(x)p(x).
\]

From the definition of expectation and (2.33), the constant of proportionality is obtained and,

\[
w(x) = \frac{f(x)p(x)}{E_p[f]}.
\]

Now, using (2.31) to calculate the variance of such a sampler,

\[
\text{var}_w(f^*(x)) = E_p[f^2] \int f(x)p(x) \, dx - E_p[f]^2
\]

\[
= 0.
\]

While clearly it would not be possible to choose such a sampler in practice, it demonstrates that in order to reduce the variance, a basic design guideline is that \( w(x) \) should be chosen to approximate the shape of \( f(x)p(x) \).

### 2.3.5 Volume of a Ball

A potential shortcoming of sampling methods as a means of integration in higher dimensions can be illustrated by considering the volume of an n-ball that fits inside a unit hypercube. This involves integrating the size of the set

\[
\left\{ x \in \mathbb{R}^{n+1} : \|x\| < \frac{1}{2} \right\}
\]

A 2-ball, as illustrated in Figure 2.4, is enclosed area of a circle, and has a known area of

\[
V = \pi r^2
\]

This circle has volume \( \pi/4 \), and fills the greater part of the enclosing square of unit volume. Although the area of this circle is known analytically, it can also be computed
by sampling uniformly across the square, and counting the proportion of samples that fall within the circle. This is the basic form of stochastic numerical integration described in Section 2.3.3. One can visualise sampling in the circle of Figure 2.4 and having a reasonable proportion of samples fall inside and outside the circle.

There is a natural extension to n-balls of higher dimension. As the dimension increases, for even n, the volume of an n-ball is

\[ V = \frac{\pi^{n/2}}{\left(\frac{n}{2}\right)!} r^n \]  

Table 2.1 shows the ball to cube ratio of volume for increasing dimension.

<table>
<thead>
<tr>
<th>Dimension</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0.785</td>
</tr>
<tr>
<td>4</td>
<td>0.308</td>
</tr>
<tr>
<td>8</td>
<td>0.015</td>
</tr>
<tr>
<td>16</td>
<td>3.6\times10^{-6}</td>
</tr>
<tr>
<td>32</td>
<td>1.0\times10^{-15}</td>
</tr>
</tbody>
</table>

*Table 2.1: Volume ratios of enclosed n-spheres*

This shows that in 8 dimensions, using uniform sampling across the hypercube, only 1.5% of samples would even fall within the ball, let alone define the shape of its surface. Given that an n-ball is one of the larger regular shapes we can visualise to fit in a hypercube, it shows that the intuition of size can easily be lost in higher dimensions. It can be very difficult for a sampling algorithm to even find the region in the parameter space \( \mathcal{X} \) where the important detail is.

### 2.3.6 Weighting by Target Distribution

When evaluating the integral,

\[ \int_{\mathcal{X}} g(x)dx, \]  

(2.41)
2.3. Integration Methods

A special case of importance sampling is when the sampling distribution can be chosen such that the weighting function \( w(x) \) is equal to the desired integrand up to a scaling constant, as,

\[
w(x) = k_w g(x),
\]

(2.42)

with \( k_w \) a normalising factor such that \( \int w(x) \, dx = 1 \). In this case the resulting estimator in (2.24) for uniform \( p(\cdot) \) is somewhat simplified with,

\[
\int_{\mathcal{X}} g(x) \, dx = \int_{\mathcal{X}} \frac{g(x)}{k_w g(x)} w(x) \, dx = E_w \left[ \frac{1}{k_w} \right].
\]

(2.43)

(2.44)

The integral may be calculated simply by estimating the expected value of the sample distribution. This choice of weighting function also makes intuitive sense, as the samples will generally be drawn from regions of \( \mathcal{X} \) where \( g(x) \) has its larger values.

An obvious challenge with utilising this approach is finding a method to efficiently draw independent samples from a distribution with a probability density proportional to \( g(\cdot) \). While this is not a trivial task, there are methods that may be used to approximate this for a substantial class of functions. In particular, this thesis will focus on the Markov Chain Monte-Carlo methods, as described in Section 2.6.

For the motivation example of marginal density computation in Section 2.2, the desired integral in (2.7) may be expressed as,

\[
\mathbf{P}(f(\theta) \in A_i) = \int g(\theta) 1(f(\theta) \in A_i) \, d\theta,
\]

(2.45)

for an arbitrary set \( A_i \), with

\[
g(\theta) = \frac{1}{P(Y)p(Y \mid \theta)p(\theta)}.
\]

(2.46)

Using \( g \) as the weighting function, the estimate in (2.27) may be represented as,

\[
\hat{\mathbf{P}}(f(\theta) \in A_i) = \frac{1}{k} \sum_{j=1}^{k} 1(f(x_j) \in A_i), \quad x_j \sim G.
\]

(2.47)

Hence the estimate for the probability density of a function of \( \theta \) may be obtained simply by constructing a histogram from samples drawn from \( G \). However, the challenge still remains in the ability to draw these samples in the first place.
2.4 Sampling Methods

The generation of pseudo-random sequences of numbers is a well-studied area\cite{40}. The methods generally involve first generating uniformly distributed values, and then applying a transformation on those values to obtain the desired distribution properties. Ripley\cite{40} presents this as five basic principles that are used in constructing specific distributions: inversion, rejection, composition, ratio of uniforms, and squeezing. These are the basic tools that can be applied to the task of generating random values with a probability density matching some given function $g(x)$.

2.4.1 Inversion

The process of inversion exploits the knowledge that if the Cumulative Distribution Function (CDF) of $X$ is $G$, then $G(X)$ is uniformly distributed on $(0, 1)$. Hence, by inverting $G$, a mapping can be made from a uniform distribution to the desired distribution of $G$. This inverse is called a quantile function, and to ensure it always exists, is defined as,

$$G^{-1}(p) = \inf\{x \in \mathbb{R} : p \leq G(x)\}. \quad (2.48)$$

The inversion algorithm involves first drawing a sample from the uniform distribution $u \sim U(0, 1)$, and then computing $x = G^{-1}(u)$ to obtain an independent sample that is distributed according to $G$. This process is illustrated in Figure 2.5. The only limitation with this method is the computation of the quantile function. This is often problematic even in one dimension due to the lack of a closed-form expression\cite{14}, but the equivalent in a multi-dimensional setting is significantly more difficult and computationally challenging. There are methods to construct the equivalent of quantile functions in a multivariate setting\cite{44}\cite{20}, but these are no less computationally costly than the original problem they are being applied to solve. Intuitively, the CDF $G(x)$ is itself an integration of $p(x)$ over $\mathcal{X}$, and the computation of a quantile function, if it were to exist, would involve this integration, together with the computation of an inverse. However, if
2.4. Sampling Methods

there were an efficient way to integrate $p(x)$, then the original problem could be solved by doing so directly.

2.4.2 Rejection Sampling

The rejection sampling methods\cite{49}, otherwise known as acceptance/rejection, or simply acceptance methods are a general approach to sample from arbitrary distributions. They are useful if we wish to sample from a density $g(x)$, but only have a means of sampling from another density $h(x)$. The first step is to find some constant $c$ such that

$$g(x) \leq ch(x) \forall x,$$  \hspace{1cm} (2.49)

where the minimum such $c$ will be shown to provide the optimum efficiency. If such a suitable constant may be found, the sampling algorithm proceeds as following,

**Algorithm 2.4.1** (Rejection Sampling).

1. sample $x_h \sim h(x)$
2. independently sample $u \sim U(0, 1)$
3. if $\frac{g(x_h)}{ch(x_h)} < u$ return to step 1
4. provide $x = x_h$ as a sample.

Steps 2 and 3 are an acceptance/rejection process where the sample $x_h$ is accepted with an acceptance probability $\alpha$ defined by,

$$\alpha(x) = \frac{g(x)}{ch(x)}.$$  \hspace{1cm} (2.50)

Potential samples $x_h$ drawn from $H$ are rejected in proportion to how much the value of $x_h$ is overrepresented in $h$, compared to $g$. Provided both the density functions $g$ and $h$ may be evaluated with low computational cost, this is quite a versatile and simple algorithm to implement.

To demonstrate that Algorithm 2.4.1 does indeed draw samples from $g(x)$, consider first the probability of accepting a sample $x_h$ on any given iteration of the algorithm,

$$P\{x_h \text{ accepted} \} = \int_{-\infty}^{\infty} h(\nu) \frac{g(\nu)}{ch(\nu)} d\nu$$  \hspace{1cm} (2.51)

$$= \frac{1}{c} \int_{-\infty}^{\infty} g(\nu) d\nu$$  \hspace{1cm} (2.52)

$$= \frac{1}{c}.$$  \hspace{1cm} (2.53)

For each possible proposal $\nu$, this represents the product of the probability of proposing $\nu$ and that of independently accepting such a proposal. The joint probability that $x_h$ be
accepted and also that \( x_h \leq x \) is obtained simply by restricting the range of the integral to,

\[
\mathbb{P}\{x_h \text{ accepted and } x_h \leq x\} = \int_{-\infty}^{x} h(\nu) \frac{g(\nu)}{ch(\nu)} d\nu
\]

\[
= \frac{1}{c} \int_{-\infty}^{x} g(\nu) d\nu.
\] (2.55)

Applying the law of conditional probability

\[
\mathbb{P}\{x_h \leq x \mid x_h \text{ accepted}\} = \frac{\mathbb{P}\{x_h \text{ accepted and } x_h \leq x\}}{\mathbb{P}\{x_h \text{ accepted}\}}
\]

\[
= \int_{-\infty}^{x} g(\nu) d\nu.
\] (2.57)

So an accepted value, \( x \), from Algorithm 2.4.1, will have a probability density of \( g(x) \) as desired.

The efficiency of rejection sampling may be measured by the expected number of samples drawn from \( h \) for a single sample to be calculated. This is given by,

\[
E[n_r] = \frac{1}{E[\alpha]},
\] (2.58)

where \( \alpha \) is the acceptance probability and \( n_r \) the ratio of rejected samples. For maximum efficiency, the aim is to maximise \( \alpha(x) \) subject to the constraint (2.49). The algorithm will operate efficiently if it is possible to sample from an \( h \) with shape similar to \( g \), with the efficiency diminishing as the difference increases. Unfortunately, as the dimension of \( x \) increases, it becomes increasingly difficult to construct a sampler that will provide a high acceptance ratio. As such, it is normally only useful in univariate contexts.

Note also that the motivation here for considering rejection sampling is to use it in conjunction with the importance sampling in Section 2.3.4 to reduce the number of samples required compared to uniform random sampling. However, rejection sampling provides no benefit in this situation, as if a suitable \( H \) exists for rejection sampling, importance sampling could instead be performed directly on the samples from \( H \).

2.4.3 Composition

In some cases it’s useful to express the desired pdf \( g(x) \) as a composition of a set of physically realisable sampling densities \( \{g_i(x)\} \) with,

\[
g(x) = \sum_{i=1}^{n_c} p_i g_i(x)
\] (2.59)

where \( \{p_i\} \) is a known distribution. In this case, the algorithm for sampling from \( G \) is to first sample from \( \{p_i\} \) in order to choose which sampler \( g_i \) to use. The resulting sample is
then drawn from the chosen \( g_i \). The utility of this technique depends entirely on it being possible to find a set of realisable densities \( g_i(x) \) for which (2.59) holds. This is not true for the general case addressed by this thesis, so composition isn’t utilised in this form.

### 2.4.4 Ratio of Uniforms

For a univariate density, the ratio of uniforms involves sampling uniformly a point \((u, v)\) from a region in a two-dimensional space. The resulting sample is calculated from the quotient.

\[
x = \frac{u}{v}
\]  

The density \( g(x) \) of \( x \) is determined by the region over which the uniform distribution is taken[30]. For example, a Cauchy distribution is achieved when a unit circle is chosen for the region of \((U, V)\). In that instance, sampling \((u, v)\) bounded to a circle may easily be achieved by rejection sampling out of a unit square.

Again, while this is a useful technique, it isn’t sufficiently general to be of use in the application of this thesis.

### 2.4.5 Squeezing

The performance of rejection sampling and related methods may be markedly improved by using a process of pre-testing or squeezing. The aim is to reduce the number of evaluations of \( g(x) \) that need to be accomplished. In rejection sampling, an acceptance ratio is calculated with (2.50),

\[
\alpha(x) = \frac{g(x)}{ch(x)},
\]

which may be costly to compute. However, it may be cheap to compute useful bounds \( \alpha_l(x) \) and \( \alpha_h(x) \) on the range of possible values of \( \alpha(x) \),

\[
\alpha_l(x) \leq \alpha(x) \leq \alpha_h(x).
\]

As \( \alpha(x) \) is only used for a comparison against a sampled value \( u \sim U(0, 1) \), it may be sufficient to instead compare \( u \) against the bounds. The comparison step in Algorithm 2.4.1
may be replaced with,

1. if $\alpha_h(x) < u$ reject sample.

2. if $\alpha_l(x) > u$ accept sample.

3. otherwise evaluate $\alpha(x)$ as normal.

If the bounds are sufficiently cheap to calculate, and prevent a significant proportion of evaluations of $\alpha(x)$, then computational savings are made. In some applications there may even be a role in calculating a series of bounds, each with increasing computational complexity.

However, in the application of interest, the acceptance ratio $\alpha(x)$ is not exceptionally expensive to calculate compared to any reasonable bounds, and so squeezing in itself is not able to make rejection sampling a viable method.

### 2.5 Laws of Large Numbers

An important area of results from probability theory is in the laws of large numbers. These laws provide a theoretical basis for the effect of accumulating a large number of samples from a given probability distribution.

For a sequence of random variables $[X_t]$, we may consider the sum,

$$S_n = \sum_{i=1}^{n} X_i.$$  \hfill (2.63)

For example, such a sum may occur when counting the number of times “heads” appears in tossing a coin $n$ times. A theorem that states a form of convergence of $\frac{1}{n}S_n$ is considered to be a law of large numbers\[26\]. The most commonly mentioned laws of large numbers are those that relate to independently and identically distributed sequences, as presented in Theorems 2.5.1 and 2.5.2.

**Theorem 2.5.1 (Weak Law of Large Numbers).** Let $[X_t]$ be independent and identically distributed with,

$$\mu = E\{X_t\}, \quad \sigma^2 = \text{var}\{X_t\} \leq \infty.$$  \hfill (2.64)

Then for any $\epsilon > 0$,

$$P \left( \left| \frac{1}{n} \sum_{i=1}^{n} X_i - \mu \right| > \epsilon \right) \to 0 \quad \text{as } n \to \infty$$  \hfill (2.65)
Theorem 2.5.2 (Strong Law of Large Numbers). Let \([X_t]\) be independent and identically distributed with,

\[\mu = E\{X_t\}, \quad \sigma^2 = \text{var}\{X_t\} \leq \infty. \tag{2.66}\]

Then,

\[\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} X_i = \mu \text{ a.s.} \tag{2.67}\]

Proof. See Theorem 20.1 of [26]

The almost sure convergence in (2.67) means that any region over which convergence does not hold is of measure zero.

While there is often a focus on the strong law of large numbers for independent sequences the same results also apply for a broader range of sequences[37]. Chapter 3 will demonstrate a strong law of large numbers for the samples from a range of Markov chain Monte–Carlo methods.

2.6 Markov Chain Monte-Carlo Methods

In Section 2.3.6 it was shown that in order to integrate arbitrary multivariate functions,

\[\int_X g(x)dx, \tag{2.68}\]

it is useful to be able to draw independent samples with a probability density function equal to \(g(x)\). However, the standard range of approaches to drawing independent samples from a distribution, as described in Section 2.4, do not offer a computationally attractive way to do this. An alternative is to look to methods which will provide a sequence of samples \(\{x_t\}\), which although not independent, are still distributed in proportion according to \(g(x)\). The Markov Chain Monte Carlo (MCMC) methods are range of sampling algorithms for which it may be shown that the output sequence \(\{x_t\}\) holds the conditions necessary for the strong law of large numbers to apply. Consequently, even though the samples are dependent, the estimate from (2.19) is still assured of converging to the correct result with sufficient iterations of the algorithm.

The origins of these algorithms date back into the 1950s with work to sample from the Boltzmann distribution. The seminal paper of Metropolis, Rosenbluth, Rosenbluth, Teller and Teller[34] details what was to become known as the Metropolis algorithm. Together with later extensions by Hastings[23], this forms the basis of the algorithms presented in this thesis.
2.6.1 Markov Chains

The distinction between Monte Carlo integration in the general sense, and MCMC, is that MCMC uses samples from a Markov chain rather than a sequence of independent samples. A Markov chain consists of a sequence of random variables \( \{X_0, X_1, X_2, ..., X_t, ...\} \), with the realisation of each \( X_t \) being known as the state of the chain at time \( t \). Each successive state depends on the prior state via a probability distribution,

\[
P(X_{t+1} | X_t).
\] (2.69)

The Markov property requires that the entire dependence is on the current state, and if conditioned on that current state, the next state is otherwise independent of prior states, with

\[
P(X_{t+1} | X_t) = P(X_{t+1} | X_t, X_{t-1}, X_{t-2}, ...).\] (2.70)

Example 2.6.1. As an example, consider the ring of states in Figure 2.7 as a game where a player may move around from one circle to another. Starting at one of the circles, at each subsequent time step, the player throws a (possibly loaded) coin, and based on the result either moves one circle clockwise with probability \( p \), or otherwise remains in the same location. Each time, the new state depends only on the present state, and the toss of the coin. While the new state does ultimately depend on prior states, all of that dependence information is captured in the knowledge of the present state.

In this example, it’s also intuitively clear that if we were to compute the long-term averages of the time spent in each state, then it would eventually converge to be equal for each state. This demonstrates a form of strong law of large numbers, the conditions for which will be developed in Chapter 3.

The approach of the MCMC algorithms is to form the same type of Markov chain, but to cleverly choose the state transition probability \( P(X_{t+1} | X_t) \) such that the time spent in each state is proportional to the desired \( g(x) \), and yet the required ergodic properties also still hold.
2.6.2 The Metropolis Algorithm

The Metropolis algorithm, as published by Metropolis et al. in 1953[34], forms the basis of a number of MCMC methods. Like the rejection sampling method, it uses a proposal and acceptance approach, however both the acceptance probability, and the behaviour on rejection are different.

In each step of the algorithm, a proposal $\xi_{k+1}$ for the next sample is drawn from a proposal density $\gamma$ with,

$$\xi_{k+1} \sim \gamma(\cdot \mid x_k).$$  

(2.71)

The choice of $\gamma$ is up to the discretion of the algorithm designer. This proposal distribution is somewhat arbitrary, and is subject only to mild constraints. Unless specific structure is known about $g(x)$, a random walk from the present state may be the generic choice. The proposal will generally depend on the present state, but is subject to a symmetry constraint that,

$$\gamma(\xi \mid x) = \gamma(x \mid \xi),$$  

(2.72)

for all $x$ and $\xi$. This symmetry requirement was later removed by Hastings’ generalisation of the algorithm[23].

Once the proposal $\xi$ is generated, it’s then subject to an accept/reject decision with an acceptance probability of,

$$\alpha(\xi_{k+1} \mid x_k) = \min \left\{ 1, \frac{g(\xi_{k+1})}{g(x_k)} \right\},$$  

(2.73)

where $g(x)$ is the desired limiting density of the chain state. If the proposal is accepted, it becomes the new state, otherwise the old state is retained for the next iteration of the algorithm. An important difference when compared to rejection sampling is that every iteration of the Metropolis algorithm emits a sample, regardless of acceptance or rejection. This method is summarised in Algorithm 2.6.1.

Algorithm 2.6.1 (Metropolis Sampling).

1. sample $\xi_{k+1} \sim \gamma(\cdot \mid x_k)$
2. independently sample $u \sim U(0, 1)$
3. compute $\alpha(\xi_{k+1} \mid x_k) = \min \left\{ 1, \frac{g(\xi_{k+1})}{g(x_k)} \right\}$
4. if $\alpha(\xi_{k+1} \mid x_k) > u$ accept $\xi$ with $x_{k+1} = \xi_{k+1}$, otherwise $x_{k+1} = x_k$
5. provide $x_{k+1}$ as a sample.
6. return to step 1 for next iteration.
The computation requirements are quite mild. It requires two pseudo-random numbers to be generated per iteration, and one new evaluation of the target density function $g(x)$. The denominator $g(x_k)$ term may generally be cached from the previous iteration.

The disadvantage of the Metropolis sampler, compared to rejection sampling, is that the samples are not independent. An increase in correlation results in a slower convergence rate under the strong law of large numbers\cite{37}. There are two mechanisms by which the samples $\{x_k\}$ from the Metropolis sampler are correlated:

- The proposal may depend on the previous state.
- When the proposal is rejected, the previous state is retained for the new state.

To an extent, design decisions that reduce the effect from one of these mechanisms will promote the effect of the other. Generally it is the correlation in the proposal distribution that allows the achievement of high acceptance ratios. As a result, correlation is a necessary cost of using the Metropolis type algorithms. The benefit is that for equivalent results, far lower computational cost is required than independent samplers, such as independent rejection sampling.

### 2.6.3 Metropolis-Hastings

A limitation of the Metropolis algorithm is the requirement in (2.72) that the proposal distribution be symmetric. Hastings’ generalisation of the Metropolis algorithm in \cite{23} removed this constraint and altered the acceptance ratio to maintain the correct limiting distribution of the chain.

When relaxing the symmetry constraint (2.72), the acceptance probability in (2.73) becomes instead,

$$
\alpha(\xi_{k+1} \mid x_k) = \min \left\{ 1, \frac{g(\xi_{k+1})\gamma(x_k \mid \xi_{k+1})}{g(x_k)\gamma(\xi_{k+1} \mid x_k)} \right\}.
$$

(2.74)

The difference is the additional multiplicative term to adjust for bias that would otherwise be introduced by the asymmetric proposal. Clearly when the symmetry constraint (2.72) is included, the terms cancel back to the same as the basic Metropolis algorithm.

In application of this algorithm, a symmetric proposal scenario can be quite a natural choice. Consider the obvious “random walk” proposal density implied by

$$
\xi_k = x_k + e_k
$$

(2.75)
in the special, but common, case where the probability density $p_e(\cdot)$ of the process generating $e_k$ is symmetric in that

$$
p_e(x) = p_e(-x).
$$

(2.76)
2.7. Bootstrap Resampling

Then clearly
\[ \gamma(\xi \mid x) = p_e(\xi - x) = p_e(x - \xi) = \gamma(x \mid \xi). \]  
(2.77)

For such a case, with a symmetric proposal density, then
\[ \gamma(x_k \mid \xi_{k+1}) = \gamma(\xi_{k+1} \mid x_k), \]  
(2.78)
and so extra term in the acceptance ratio is
\[ \frac{\gamma(x_k \mid \xi_{k+1})}{\gamma(\xi_{k+1} \mid x_k)} = 1. \]  
(2.79)

This confirms that the Metropolise–Hastings algorithm reduces to the original Metropolis algorithm when using a symmetric proposal density.

Although the Metropolis–Hastings algorithm may be identified as currently popular in the statistics community, it is also widely used in physics, chemistry and biology, as profiled in [27] where it is listed at first place in a survey of great algorithms of scientific computing.

### 2.7 Bootstrap Resampling

An alternate approach to sampling from a desired parameter distribution can be provided by the Bootstrap[13] method, which is itself a variant of earlier resampling methods, such as the jackknife[12]. Efron introduced the bootstrap method in 1979 as a means of evaluating the accuracy of statistical parameter estimators.

Returning to the motivating example in (2.1), we consider the parameter estimation problem of the system,
\[ Y = f(U, \theta) + \epsilon. \]  
(2.80)

To utilise the bootstrap, some additional restrictions need to be placed on the range of problems that can be approached. The output vector \( Y \) consists of \( N \) real-valued samples, \( Y \triangleq \{y_1, \ldots, y_N\} \). Each of these \( N \) values \( \{y_i\} \) are additively corrupted by an independent and identically distributed noise process with realisations \( \{\epsilon_i\} \).

If there is a computationally efficient method available to calculate a maximum likelihood estimate \( \hat{\theta} \),
\[ \hat{\theta} = \arg \max_\theta p(Y \mid \theta), \]  
(2.81)
then the bootstrap may be used to provide estimates of probability densities of arbitrary functions of \( \theta \).

The first step in using the Bootstrap method involves evaluating the maximum like-
likelihood estimate $\hat{\theta}$ from (2.81). From this, a set of error residuals may be computed,

$$\{\hat{\epsilon}_t\} = Y - f(U, \hat{\theta}).$$  \hspace{1cm} (2.82)

These are a measure of the error in the parameter estimation, but are not immediately useful in determining information about $\hat{\theta}$. To do that, the Bootstrap uses Monte-Carlo process that involves resampling from $\{\hat{\epsilon}_t\}$ and recomputing the maximum likelihood estimate $\hat{\theta}$ with the resampled residuals.

For each iteration $k$ of the algorithm, a set of $N$ samples $\{r_t\}_k$ are drawn randomly with replacement from the set of $N$ residuals $\{\hat{\epsilon}_t\}$. From this, a new realisation of $Y$ is fabricated with

$$\tilde{Y}_k = Y + \{r_t\}_k.$$  \hspace{1cm} (2.83)

The justification for this is that the empirical values of the residuals are a good estimator of the population distribution of the random process. The aim of the Bootstrap is to generate fake new data that can be used in the place of multiple experiments under different noise realisations.

Given the new realisation of $Y$, a new estimate for $\theta$ is estimated with

$$\tilde{\theta}_k = \arg \max_{\theta} p(\tilde{Y} | \theta).$$ \hspace{1cm} (2.84)

The sequence $\{\tilde{\theta}_k\}$ is an approximation to a set of samples drawn from the distribution of $p(Y | \theta)$, from which the importance sampling of Section 2.3.4 may be used to derive results for arbitrary functions of $\theta$.

Using Bayes Theorem (2.5), and applying importance sampling on $\{\tilde{\theta}_k\}$, probability estimates may be achieved with

$$\hat{p}(f(\theta) | Y) = K_s \sum_k f(\tilde{\theta}_k)p(\tilde{\theta}_k),$$  \hspace{1cm} (2.85)

where $K_s$ is a normalising constant.

### 2.8 Sequential Monte Carlo

An important group of sampling algorithms are those that fall under the name of *Sequential Monte Carlo* (SMC)[6, 10]. These methods have developed from the work of Gordon *et al.*[17] in the early 1990s, and are now applied to many areas of signal processing. The names *Sequential Importance Resampling* and *Particle Filters* are applied to methods that fall within this category.

As the name suggests, these are sequential methods, which involve characterising the state of a stochastic process as it evolves with time. Unlike the other algorithms
presented in this chapter, the sampling algorithm is closely connected with the evolution of a hidden state within the model. Consequently it’s necessary that the system model include this state, and in place of the very general system model in (2.1) a more specific state-space model structure is used,

\[ x_{t+1} = f(x_t, u_t, v_t, \theta) \]  
\[ y_t = g(x_t, u_t, e_t, \theta). \]

This is a Markov process where the hidden state variable \( x_t \in \mathbb{R}^{n_x} \) is driven by an input \( u_t \in \mathbb{R}^{n_u} \), and described by the model parameters \( \theta \in \mathbb{R}^n \). The measured output \( y_t \in \mathbb{R}^N \) depends on the present state and inputs. There is also a random state noise \( v_t \), and output noise \( e_t \).

Under this formulation, the most natural application of Sequential Monte Carlo is in the estimation of the probability density of the state \( p(x_t | Y_t) \) as a function of time, where \( Y_t = \{y_0, \ldots, y_t\} \) are the observed measurements up until time \( t \). It may also be extended to include estimation of the parameters \( p(\theta | Y_t)[45, 53] \), and with the inclusion of an additional smoothing step, estimates of the densities conditioned on all of \( Y = \{y_t\} \) may also be made.

It’s convenient to think of the particle filter as an extension of the Kalman filter to cover non-linear systems and non-Gaussian noise. In linear, Gaussian systems, \( p(x_t | Y_t) \) follows a Gaussian distribution, and so may simply be represented by an expected value \( \hat{x}_t \), and a covariance \( P \). However, in the more general case, it is necessary to use a non-parametric representation of the probability density.

The Particle Filter method involves representing \( p(x_t | Y_t) \) by a set of \( M \) random samples \( \{x_t^i\} \) drawn from that distribution. Each iteration of the particle filter involves moving and weighting the “particles” to update the estimated state probability density function. Consequently, at any point in time, \( t \), of the state-space model, the distribution of the location of these particles is designed so as to form a representation of the underlying distribution of the state \( x_t \) given observed outputs up until time \( t \), \( p(x_t | Y_t, \theta) \). In the same way that each sample from an MCMC algorithm represents the distribution of \( \theta \), at time \( t \) each particle in a particle filter represents the distribution of the present state \( x_t \).

In order to improve the flexibility of the algorithm, the particle filter attaches a weight to each particle after the manner of importance sampling. The weight \( w_t^i \) of each particle are allocated so that the set of particles together with weights \( \{x_t^i, w_t^i\} \), form an approximation to the probability density of \( x_t \). More specifically, an approximation may be constructed as,

\[ \int_A p(x_t) \approx \sum_{i:x_t^i \in A} w_t^i. \]  
\[ (2.88) \]
The steps for each iteration of the particle filter algorithm are illustrated in Figure 2.8. Here the weight of a particle is depicted in the area of the circle representing it. Note that the measurement update and time update each alter the density function for the state, but the resampling step only alters the way it’s represented.

The first step of the algorithm is to draw an initial set of \( M \) particles \( \{ \mathbf{x}_0^i \} \) from the a priori distribution of \( \mathbf{x} \). Following that, for each \( t \) a process of weighting, resampling and propagation is carried out.

### 2.8.1 Weighting

On each iteration, the weight of each particle is updated according to the new measurement \( y_t \). This is done by considering the likelihood of each particle \( \mathbf{x}^i_t \) causing an output equal to \( y_t \), as defined by (2.87). Due to the independence of the noise process \( e_t \), the new weight will simply be the product of the old weight and the likelihood function,

\[
w_{t|t}^i = w_{t|t-1}^i \frac{p_\theta(y_t | \mathbf{x}^i_t)}{K_w},
\]

where \( K_w \) is a constant that is computed via normalising the particle weights.

### 2.8.2 Propagation

For each time step, the state transition kernel (2.86) is used to predict ahead the movement of each particle. One way of achieving this would be for each particle \( i \) to compute

\[
\tilde{x}_t \sim p_\theta(\tilde{x}_t | x_{t-1}).
\]
This could be achieved by evaluating (2.86) with a $v_t$ drawn from the state noise distribution. This is a time update step, and the effect of this is to simulate the expected movement of the state for each of the possible candidate positions as represented by the set of particles.

If using (2.90) to propagate the particles, each particle would retain its existing weight. However there can be advantage in using importance sampling at this step, and instead drawing from a related, but different, proposal distribution,

$$x_i^t \sim \gamma(\cdot | x_{i-1}^t, y_t).$$

(2.91)

In order to compensate for the difference between $\gamma$ and $p_\theta$, a weighting adjustment is applied to each particle,

$$w_{i|t-1} = w_{i|t-1} \frac{p_\theta(x_i^t | x_{i-1}^t)}{\gamma(x_i^t | x_{i-1}^t)}$$

(2.92)

2.8.3 Resampling

With only the propagation and weighting stages, the algorithm implements a Sequential Importance Sampler. While it will calculate correct weights, it has a tendency to retain too many particles with a low weight. Such particles require significant computation, but have little effect on the computed results. The solution to this is via a resampling stage.

Resampling is related to the Bootstrap method, and involves drawing a new set of particles $\{\tilde{x}_i^t, \tilde{w}_{i|t-1}^t\}$ with the same weighted distribution as $\{x_i^t, w_{i|t-1}^t\}$, but where all the new samples $x_i^t$ are of equal weight. This may be achieved simply by drawing with replacement the new values from $\{x_i^t\}$ with a weight of $w_{i|t-1}^t$ applied to each candidate.

The cost of resampling is both computational and also that the resampling process increases the variance of the distribution estimate of the particles. Due to this cost, often the resampling step is not carried out at every iteration of the algorithm, but rather only for a subset of iterations. If resampling is not carried out in a given iteration, the particles and weights are retained with

$$\tilde{x}_i^t = x_i^t$$

$$\tilde{w}_{i|t-1}^t = \frac{w_{i|t-1}^t}{\sum_j w_{j|t-1}^t}.$$  

(2.93)  

(2.94)

2.8.4 Particle Filter Algorithm

Algorithm 2.8.1 summaries the basic process of the particle filter as described in the sections above.

Algorithm 2.8.1 (Particle Filter).

1. For $i = 1, ..., M$ sample $M$ initial particles $x_0^i \sim p_\theta(x_0)$. 
2. Re-weight the particles based on measurement with

\[ w_{i|t}^i = w_{i|t-1}^i \frac{p_\theta(y_t | x_{i|^t}) p_\theta(x_{i|^t} | x_{i|^t-1})}{\gamma(x_{i|^t} | x_{i|^t-1})} \]  \hspace{1cm} (2.95)

3. (if resampling)

For each \( i \in \{1, \ldots, M\} \) resample a new particle \( \tilde{x}_{i|^t} \) with \( P(\tilde{x}_{i|^t} = x_{i|^t}) = w_{i|^t} \), for \( j \in \{1, \ldots, M\} \). Assign weight \( \tilde{w}_{i|^t} = \frac{1}{M} \).

4. Propagate particles according to a proposal distribution \( x_{i|^t} \sim \gamma(\cdot | x_{i|^t-1}, y_t) \)

5. Increment \( t \) and return to step 2.

The particle filter is used within the Metropolis–Hastings algorithm in Section 6.12.

### 2.8.5 Comparison to MCMC

Particle filters and other variants of sequential Monte Carlo have been successfully applied in many areas of signal processing. On occasion they are confused with MCMC methods, but there are a number of significant differences that separate the methods.

Figure 2.9 demonstrates the distinction in the computation timeline of MCMC vs SMC. MCMC performs stochastic sampling over \( M \) iterations where each of those \( M \) iterations involves evaluating across all of the time steps in the model input and output data. In comparison, SMC iterates across each of the outputs \( y_t \), and within each of those iterations evaluates all of the particles.

Beyond the block/sequential differences of the algorithms, there are also quite fundamental differences in how they operate. As presented in their natural usage, the SMC methods deeply relate to the state of the modelled system, whereas the evolution of the
state is largely transparent to the MCMC methods. In MCMC, the state may be estimated, and density functions computed, but it doesn’t affect the progress of the sampling algorithm.

Due to the lack of state dependence in the algorithm, MCMC inherently produces estimates that are fixed-interval smoothed.

2.9 MCMC via SMC

The fundamental use of SMC is in state estimation, while that of MCMC is in parameter estimation.

As a means to better understand the relationship between these two methods, it is useful to construct a Particle Filter sampler that closely follows the operation of an MCMC sampler. To do this, we choose a hypothetical, and somewhat degenerate, auxiliary state space system,

\[
\begin{align*}
  x_{t+1} &= v_t \quad v_t(x) \sim p(Y \mid \theta = x) \\
  y_t &= 0.
\end{align*}
\]

The state \( x \in \mathbb{R}^n \), where \( n \) is the size of the parameter vector \( \theta \), is a representation of an estimate of \( \theta \), and unlike regular particle filtering, does not include these states of the underlying model that produces the \( Y_N \) output. In the model there is no correlation in the state, with \( P(x_{k+1}) = P(x_{k+1} \mid x_k) \). The state is simply random, and drawn from the posterior \( p(Y \mid \theta) \). Being a hypothetical system, there is no measured output available to compare to that of the model. Clearly, the intention of this model is to be able to acquire estimates of the state \( x_t \), and use these as samples from \( p(Y \mid \theta) \). In this sense, the state of this SMC sampler now performs the same role as the state of the MCMC algorithm.

2.9.1 Algorithm Implementation

The SMC algorithm is initialised from the same distribution that the initial state of the MCMC algorithm is, where for some reasonable estimate \( \hat{p} \),

\[
x_0^i \sim \hat{p}(\theta \mid Y).
\]

The propagation step involves a choice of a proposal distribution. In this situation, it is analogous to the proposal of the MCMC algorithm, and so the same distribution \( \gamma \) is used with

\[
\xi_k^i \sim \gamma(\cdot \mid x_{k-1}^i).
\]

For example, this could be a multi-dimensional random walk from \( x_t \), subject to a linear
transformation. While there is no correlation in the underlying model, the choice of proposal does introduce correlation into the sequences of \( \{x^t_i\} \).

The weights for propagation are calculated as per (2.92) using the particularly chosen state distribution \( v_t \),

\[
    w^i_t = w^i_{t-1} \frac{p(Y \mid \xi^i_k)}{\gamma(\xi^i_k \mid x^i_{k-1})}.
\]

In this application of SMC, the weighting step is trivial. This is because the system model has no output, and so no information is gained about the particles from an output \( y_t \). Consequently, the weights remain the same, or are merely normalised.

The resampling step proceeds as per a standard particle filter. A new set of particles \( \{x^t_i\} \) are sampled according to the weighted set of existing particles.

### 2.9.2 Relationship to MCMC

In terms of similarities back to the MCMC algorithm, it is convenient to think of each particle representing the state of one of a set of parallel MCMC chains. They have the same initialisation process, and the proposal distribution for new values in those chains is the same. However, it isn’t immediately obvious whether the acceptance/rejection process of MCMC is retained.

Considering the calculation of the weighting function in (2.100), it may be noted that taking the quotient of the weights of two particles results in a form of expression that is remarkably similar to the acceptance ratio of the Metropolis-Hastings algorithm in (2.74). Assuming equal initial weights for two particles denoted \( i \) and \( j \), then after propagation

\[
    \frac{w^i_t}{w^j_t} = \frac{p(Y \mid \xi^i_k)\gamma(\xi^i_k \mid x^i_{k-1})}{p(Y \mid \xi^j_k)\gamma(\xi^j_k \mid x^j_{k-1})}.
\]

The only difference is that here we compare two different proposed particles, rather than comparing a proposed particle with the present state. The similarities continue when we recognised that the resampling process is essentially a form of acceptance/rejection. In this case, through resampling, the probability of acceptance of any of the proposed particles is proportional to the weight of that particle compared to that of the other particles.

In this section, by tweaking the particle filter to match the behaviour of the Metropolis algorithm, it’s been shown that while it can be made to perform a similar role, the standard application of the Particle filter is very different to that of the MCMC methods. This is perhaps made clearer in Section 6.12, where the particle filter is used in conjunction with the Metropolis–Hastings algorithm in order to perform parameter estimates of non-linear models.
2.10 Conclusion

The task of estimating parameters of stochastic processes from observed measurements is inherently one of integration. Given that the dimension of space over which this integration takes place is related to the number of parameters, for models with larger numbers of parameters this can be a computationally intensive process. One approach of making this task more tractable is to perform the integration via stochastic methods. This may be done by computing the expectation of samples from specially crafted random number generators. In particular, the Markov chain Monte–Carlo methods offer promise as a means of doing this.
Chapter 3

Fundamentals of Markov Chains

3.1 Introduction

The remainder of this thesis is concerned with the implementation of Markov Chain Monte Carlo algorithms. One of the aims is to demonstrate that under suitable implementation choices, the probability density estimates produced from these algorithms always eventually converge to the given target density. This chapter establishes the basic Markov Chain framework that is later used for the implementation of MCMC algorithms. The aim is to make clear the underpinning requirements for the MCMC convergence results.

A Markov Chain is a particular form of random process. The distinguishing feature is that it retains no memory of the past beyond what is known from the current state of the process. Each new state has a transition probability that depends only on the present state. This chapter will consider only discrete-time Markov Chains, where there are a countable number of points in time in which a state transition may occur. MCMC methods may operate over either discrete or continuous state spaces, so both of these cases are considered. This chapter is divided into two sections, relating respectively to the discrete and continuous state cases. As the theory is simpler in the discrete case, this is presented first, with a greater emphasis in finding intuition in the definitions and results.

The results in this chapter are well known, but are here brought together in order to provide the complete basis for the results needed in this thesis. In order to provide a self-contained presentation, proofs are supplied for some results where they provide insight into the way the Markov chain theory works.

3.2 Markov Chains on Countable Spaces

Markov chain theory is somewhat simpler and clearer when the Markov state is constrained to be an element of a countable set. In this case there are a finite, or at least a countable, number of possible states, and the Markov chain may be visualised as a graph
of the type shown in Figure 3.1.

![Figure 3.1: Example Markov Chain](image)

The state transition diagram in Figure 3.1 illustrates a particular Markov Chain with three states. At each point in the discrete time axis, the chain will be in one of the three states. The state for the next point in time has a probability distribution denoted by the labels of the edges in the graph leaving that state. For example, if at some time \( t = t_0 \), the state of the chain is \( s_1 \), then at time \( t = t_0 + 1 \) the probability of transitioning to state \( s_3 \) is \( p_{1,3} \), and in the case shown in Figure 3.1, it is not possible to transition directly to state \( s_2 \) from state \( s_1 \).

For this system, the transition probabilities may be represented in a matrix form,

\[
P = \begin{bmatrix}
p_{1,1} & 0 & p_{1,3} \\
p_{2,1} & 0 & p_{2,3} \\
p_{3,1} & p_{3,2} & 0
\end{bmatrix}.
\] (3.1)

In what follows, the notation,

\[
[P]_{i,j},
\] (3.2)

will be used to denote the element in the \( i \)th row and \( j \)th column of the matrix. For states \( i \) and \( j \), the matrix entry \([P]_{i,j} = p_{i,j}\) represents the probability of transitioning to state \( j \), given a present state \( i \). To satisfy the law of total probability, the sum of probabilities leaving from each state must sum to unity. For example, for the case of state \( s_1 \), this means \( p_{1,1} + p_{1,3} = 1 \). In the matrix form, this corresponds to a requirement that the sum of each row of \( P \) is one.

### 3.2.1 Markov Chain Definition

More formally, let \( I \subset \mathbb{Z} \) be the countable set of possible states of the chain, and define \( i \in I \) as a state. For ease of notation, in this chapter, \( I \) will be constrained to sets of the form \( I = \{1, 2, 3, \ldots\} \). However, arbitrary countable sets may be accommodated simply by mapping them to the countable numbers. The probability of the chain taking a particular state \( i \) may be denoted \( \lambda_i \), where the row vector \( \lambda = [\lambda_1, \lambda_2, \ldots] \) is a distribution on \( I \), in that
3.2. Markov Chains on Countable Spaces

\[ \lambda = \{ \lambda_i, i \in I \}, \quad \sum_{i \in I} \lambda_i = 1. \quad (3.3) \]

Using the probability space \((\Omega, \mathcal{F}, \mathbf{P})\), we define the random variable \(X : \Omega \to I\) taking values in \(I\) with the measure \(\mathbf{P}\) defined by the distribution \(\lambda\) as

\[ \mathbf{P}(X = i) = \lambda_i. \quad (3.4) \]

A matrix \(P\) is termed a stochastic matrix if it obeys the constraints,

\[ [P]_{i,j} \geq 0, \quad \sum_j [P]_{i,j} = 1. \quad (3.5) \]

All of the elements are non-negative, and all the row sums are equal to one, as in the example transition probability matrix in Equation (3.1). A transition matrix \(P\) defines the probability of \(X\) taking state value \(j\) subsequent to it taking state value \(i\) \((i \to j)\) as being \([P]_{i,j}\).

The collection of random variables \(\{X_n\}, n \geq 0\) is a Markov Chain with an initial distribution of \(\lambda\) and transition matrix \(P\) if and only if

1. \[ \mathbf{P}\{X_0 = i_0\} = \lambda_{i_0}; \quad (3.6) \]

2. \[ \mathbf{P}\{X_{n+1} = i_{n+1} \mid X_0 = i_0, \ldots, X_n = i_n\} = \mathbf{P}\{X_{n+1} = i_{n+1} \mid X_n = i_n\}, \quad (3.7) \]

for each state sequence \(i_0, i_1, \ldots, i_{n+1} \in I\).

When the process \(\{X_n\}\) meets these criteria, it is said to be “Markov(\(\lambda, P\))”. The first condition defines the initial probability distribution of the state, while the second represents the Markov property. Under (3.7), the only prior state dependence in a state transition is that of the immediately prior state. Note that the transition probability in (3.7) is equal to the corresponding transition matrix entry,

\[ \mathbf{P}\{X_{n+1} = i_{n+1} \mid X_n = i_n\} = [P]_{i_n, i_{n+1}}. \quad (3.8) \]

3.2.2 Time Evolution of Distribution

In (3.7), the Markov criteria is expressed in terms of the conditional probability for a state transition being independent of past states. This independence implies that the joint probability of a sequence of state transitions can be calculated directly from the individual transition probabilities.

Hence, an alternate definition is that the process \(\{X_n\}\) is Markov (\(\lambda, P\)) if and only
if the joint probability of a realisation \( \{ X_0, ..., X_N \} \) can be decomposed as

\[
P\{ X_0 = i_0, X_1 = i_1, \cdots, X_n = i_n \} = \lambda_{i_0} \prod_{k=0}^{n-1} [P]_{i_k, i_{k+1}}.
\]  

(3.9)

This represents the probability of a particular state trajectory. In order to evaluate the probability of reaching a particular state, regardless of the trajectory, it’s necessary to sum across all of the possible state sequences that could lead to it. While this could appear difficult to express, the matrix form of the state transition probability distribution, as defined in (3.7), provides a means to characterise the probability density of states after an arbitrary \( n \) steps of the Markov chain.

To consider the probability distribution of the states, a vector \( \lambda \) is formed, with one element for each possible state of the Markov chain. For such a vector \( \lambda \), the notation \([\lambda]_j\) is used to denote the \( j \)th element. Where \( \lambda \) is a probability distribution of the states, \([\lambda]_j\) represents the probability of state \( j \). For an initial distribution of \( \lambda \), after one state transition of the chain, the probability of the chain being in state \( j \) is,

\[
P\{ X_1 = j \} = \sum_{i \in I} \lambda_i P_{ij} = [\lambda P]_j.
\]  

(3.11)

Multiplication of the state probability vector by \( P \) provides the state probability distribution after one state transition. By extension, the probability distributions of states in subsequent steps may be found by multiplication with higher powers of the \( P \) matrix. So for all \( n \geq 0 \)

\[
P\{ X_n = j \mid X_0 = i \} = [P^n]_{i,j},
\]  

(3.12)

\[
P\{ X_n = j \} = [\lambda P^n]_j,
\]  

(3.13)

where \( P^0 \) is defined as the identity matrix.

### 3.2.3 Communication between States

According to the structure of the Markov chain transition matrix, it may or may not be possible to reach a certain state \( j \in I \). Where there is a non-zero probability of eventually reaching a state \( j \) from a starting state \( i \), it is said that \( i \) communicates with \( j \), and is denoted by \( i \to j \). More specifically, \( i \) communicates with \( j \) when

\[
P\{ X_n = j \text{ for some } n \geq 0 \mid X_0 = i \} > 0.
\]  

(3.14)

Using the transition matrix notation, this condition may be equivalently expressed
as,

$$[P^n]_{i,j} > 0 \text{ for some } n \geq 0. \quad (3.15)$$

A given state, $i$, will communicate with another, $j$, precisely when there exists a sequence of states between $i$ and $j$ where the transition between each has a non-zero probability. Let a sequence of this form be denoted $i_0, \ldots, i_n$ with $i_0 = i$ and $i_n = j$ and some $n > 0$. A necessary and sufficient condition for $i \to j$ is that such a sequence exists with

$$\prod_{k=0}^{n-1} [P]_{i_k,i_{k+1}} > 0 \quad (3.16)$$

When two states $i$ and $j$ communicate with each other, that is $i \to j$ and $j \to i$, we say that $i$ intercommunicates with $j$. This is written $i \leftrightarrow j$.

### 3.2.4 Classes of the state space $I$

In a given Markov chain, there may be useful characteristics of that chain which do not apply to the whole of the state space. In this situation, it is useful to partition the space of possible states into regions that are amenable to analysis.

From (3.16), it is readily seen that if $i \to j$ and $j \to k$, then $i \to k$. Coupled with the property that $i \to i$, the operator $\leftrightarrow$ forms an equivalence relation on $I$. The intercommunication operator then partitions $I$ into a set of communicating classes. All states in a communicating class communicate with every other state in the class.

In the general case, the state of the Markov chain may move between different communicating classes. A class that can never be left is called a closed communicating class. A communicating class $C$ of $I$ is said to be closed if

$$\forall i \in C, \quad i \to j \implies j \in C. \quad (3.17)$$

When a closed class consists of only one state, that state is called an absorbing state. That is, a state $i$ is absorbing if

$$P\{X_{n+1} = i \mid X_n = i\} = 1. \quad (3.18)$$

In other words, once an absorbing state is reached, the chain never leaves that state.

An irreducible Markov chain is one where the state space $I$ forms a single communicating class. The example in Figure 3.1 is an irreducible chain, because any state can eventually be reached when starting from any other.

### 3.2.5 Hitting Times and Absorption Probabilities

A quantity of interest in a Markov chain is how long it may take until a particular state, or set of states, is reached. The hitting time is an integer valued random variable $H^A_i$.
defined for a subset of states $A \subset I$ and an initial state $i$, according to

$$H^A_i = \min\{k > 0 : X_k \in A, X_0 = i\}, \quad (3.19)$$

with the natural extension of $H^A_i = \infty$ for the case where the Markov chain never reaches the state in $A$. The term first passage time to a state $j$ is used to denote the hitting time of the set of states consisting solely of one state $j$, ie. $H^{(j)}_i$.

When $A$ is a closed class, the probability $h^A_i$ that starting from state $i$, $\{X_n\}$ ever hits $A$ is termed the absorption probability and defined as,

$$h^A_i = P\{H^A_i < \infty\}. \quad (3.20)$$

### 3.2.6 Recurrence and Transience

For a Markov chain run for an infinite time, states in that chain may be characterised by how often they are visited during the running of the chain. A recurrent state is one that is visited infinitely many times with probability 1, while a transient state is one that has zero probability of being visited infinitely many times. It can be shown that a state must either be recurrent or transient[38, Theorem 1.5.3].

Over an infinite period, a state will be visited infinitely many times when there is a finite time between successive visits of that state. This leads to a definition of recurrence based on the hitting time for a state. A state $i$ is said to be recurrent if,

$$P\{H^i_i < \infty\} = 1. \quad (3.21)$$

If instead,

$$P\{H^i_i < \infty\} < 1, \quad (3.22)$$

then the state $i$ is transient.

The infinite number of visits to a recurrent state may also be characterised via the sum of probabilities of return visits to that state. For a recurrent state $i$,

$$\sum_{n=0}^{\infty} [P^n]_{i,i} = \infty, \quad (3.23)$$

while for a transient state $j$,

$$\sum_{n=0}^{\infty} [P^n]_{j,j} < \infty. \quad (3.24)$$

If a state $i$ is recurrent, and $i \to j$, it can be shown that $j$ is also recurrent[38, Theorem 1.5.4]. From this follows the result that if $C$ is a communicating class, then either all of the states in the class $C$ are recurrent or they are all transient. In the former case, $C$ is called a recurrent class, or in the latter, a transient class.

In order to be recurrent, a class must be closed. Otherwise, if a class $A$ is not closed,
there must exist states $i \in A, j \notin A$ such that $i \to j$ but $j \not\to i$. As $[P^n]_{i,j} > 0$ for some $n$, by (3.22) the class must be transient. It may also be shown that all finite closed classes are recurrent [38, Theorem 1.5.6].

When a chain is irreducible, it consists of a single communicating class. If this class is also recurrent, then for all $i, j \in I$,

$$P\{H_i^j < \infty\} = 1.$$  (3.25)

So for an irreducible and recurrent Markov chain, regardless of the initial state, every other state will eventually be reached with probability 1.

### 3.2.7 Invariant Distributions

For each $n$, the random variable $X_n$ will take on a distribution, as defined in (3.7). The way that this distribution evolves over time is an important characteristic of a Markov chain. One special case is that of the distribution remaining constant. This is known as an **invariant** distribution.

A distribution $\lambda$ is termed to be invariant for $P$ if and only if

$$\lambda P = \lambda.$$  (3.26)

If ever an invariant distribution $\lambda$ is reached, then by (3.26) clearly all future $X_n$ will have distribution $\lambda$.

Alternatively, a process $X_n$ may converge in the limit as $n \to \infty$ to a particular distribution $\pi$. In Theorem 3.2.1 it is shown that in this case, such a limiting distribution $\pi$ must satisfy (3.26), and hence is invariant for $P$.

**Theorem 3.2.1.** For a finite state space $I$, suppose that for some $i \in I$ and for all $j \in I$,

$$\lim_{n \to \infty} P(X_n = j \mid X_0 = i) = \lim_{n \to \infty} [P^n]_{i,j} = \pi_j.$$  (3.27)

Then $\pi = [\pi_1, \pi_2, \cdots]$ is an invariant distribution for $P$, i.e. $\pi P = \pi$.

**Proof.**

$$\pi_j = \lim_{n \to \infty} [P^n]_{i,j}$$  (3.28)

$$= \lim_{n \to \infty} \sum_{k \in I} [P^{n-1}]_{i,k} [P]_{k,j}$$  (3.29)

$$= \sum_{k \in I} \lim_{n \to \infty} [P^{n-1}]_{i,k} [P]_{k,j}$$  (3.30)

$$= \sum_{k \in I} \pi_k [P]_{k,j}$$  (3.31)

$$= [\pi P]_j.$$  (3.32)
The interchange of the limit and the sum is possible due to \( I \) being defined finite. The remaining condition, that \( \pi \) is a distribution, is confirmed as,

\[
\sum_{j \in I} \pi_j = \sum_{j \in I} \lim_{n \to \infty} [P^n]_{i,j} = \lim_{n \to \infty} \sum_{j \in I} [P^n]_{i,j} = 1.
\]

This theorem demonstrates a sufficient condition for an invariant distribution to exist, but does not guarantee that a chain will ever reach that distribution. This behaviour may depend on the initial state, or initial distribution of states of the chain, and so further properties are needed in order to demonstrate convergence to the invariant distribution.

### 3.2.8 Detailed Balance

A requirement of Markov Chain Monte Carlo sampling methods is to construct a Markov chain with a specified limiting distribution. This motivates the need for a method to construct a transition matrix with a known invariant distribution. A convenient approach to doing this is to consider the probability of transition between pairs of states.

For a Markov chain \((\lambda, P)\), a distribution \(\lambda\) and transition matrix \(P\) are said to be in detailed balance when for all pairs of states \(i, j \in I\),

\[
\lambda_i [P]_{i,j} = \lambda_j [P]_{j,i}. \tag{3.33}
\]

If \(\lambda\) represents the probability distribution of the states at some time \(t\), the left side of this relationship represents the probability of a transition occurring from state \(i\) to state \(j\) at time \(t+1\). Similarly, the right side represents the probability of a transition from \(j\) to \(i\). So in detailed balance,

\[
P(X_t = i, X_{t+1} = j) = P(X_t = j, X_{t+1} = i). \tag{3.34}
\]

This is illustrated in Figure 3.2, showing the transition probabilities between two connected states.

![State transition flow for detailed balance.](image)

These probabilities may be considered a flow of probability mass, and the equality for both directions represents a form of equilibrium[18].
Theorem 3.2.2. If a Markov chain has a transition matrix $P$ and a distribution $\lambda$ in detailed balance, then $\lambda$ is an invariant distribution for the chain.

Proof. (From [18, Theorem 6.1]) If $P$ and $\lambda$ are in detailed balance, then for all $i \in I$,

$$[\lambda P]_j = \sum_{i \in I} \lambda_i [P]_{i,j} = \sum_{i \in I} \lambda_j [P]_{j,i} = \lambda_j \sum_{i \in I} [P]_{j,i} = \lambda_j. \quad (3.38)$$

Consequently $\lambda = \lambda P$. Therefore, a process that is Markov $(\lambda, P)$ and in detailed balance, has $\lambda$ as an invariant distribution with respect to $P$.

Detailed balance is a sufficient, but not necessary condition. A Markov chain may have an invariant distribution without it necessarily being in detailed balance with the transition matrix. For example, for the chain illustrated in Figure 2.7, the uniform distribution is invariant, but the detailed balance condition isn’t met.

A key feature of Markov chains is that given the present state, the past and future states are independent. Based on the Markov property,

$$p(x_{t+1} \mid x_t) = p(x_{t+1} \mid x_t, x_{t-1}), \quad (3.39)$$

it then follows that,

$$p(x_{t+1} \mid x_t) p(x_{t-1} \mid x_t) = p(x_{t+1} \mid x_t, x_{t-1}) p(x_{t-1} \mid x_t) = p(x_{t+1}, x_{t-1} \mid x_t). \quad (3.41)$$

This demonstrates a time symmetry of the Markov property. It follows that (3.7) may be expressed with reversed time index as,

$$\mathbb{P}\{X_{n-1} = i_{n-1} \mid X_n = i_n, X_{n+1} = i_{n+1}, \cdots\} = \mathbb{P}\{X_{n-1} = i_{n-1} \mid X_n = i_n\}. \quad (3.42)$$

This means that a Markov chain run backwards is also a Markov chain[38, Theorem 1.9.1], although the reversal chain may have a different transition matrix. If a Markov chain and its time-reversal chain have the same transition matrix, it is said to be reversible.

The symmetry of Figure 3.2 suggests that when the transition matrix is in detailed balance with an equilibrium distribution, then there is a time symmetry of a Markov chain. Indeed, when $P$ is an irreducible transition matrix, then a chain that is Markov $(\lambda, P)$ is reversible if and only if $\lambda$ and $P$ are in detailed balance[38, Theorem 1.9.3].
3.2.9 Positive Recurrence

The definition of recurrence is useful in establishing there is a form of predictable long-term behaviour in a Markov chain. However, it is not a strong enough condition to ensure the presence of an invariant distribution. In this section, a stronger variant is introduced that more closely relates to the presence of the invariant distribution.

As defined in Section 3.2.6, a state \( i \) is recurrent if

\[ P\{X_n = i \text{ for infinitely many } n \mid X_0 = i\} = 1, \]

and this can be defined in terms of the hitting time \( H_i^i \) as,

\[ P\{H_i^i(\omega) < \infty\} = 1. \tag{3.43} \]

Another measure of the hitting time is the expected return time, defined as the expected value of \( H_i^i \),

\[ m_i = E\{H_i^i\}. \tag{3.44} \]

Even when the probability \( P\{H_i^i < \infty\} \) is 1, ensuring recurrence, it is possible that \( m_i \) may still be infinite. This leads to the definition of positive recurrence.

**Definition 3.2.1.** If a Markov Chain \( \{X_k\} \) has state \( i \) with an expected return time \( m_i < \infty \), then that state is said to be positive recurrent. Alternatively, a recurrent state that is not positive recurrent is termed null recurrent.

**Theorem 3.2.3.** If an irreducible Markov chain contains one state \( i \in I \) that is positive recurrent, then all states in \( I \) are positive recurrent.

**Proof.** Shown in Norris[38, Theorem 1.7.7]. \( \square \)

Based on this theorem, a Markov chain is said to be positive recurrent if all of its states are positive recurrent. The property of positive recurrence is important because it is closely related to the existence of an invariant distribution of the chain.

**Theorem 3.2.4.** An irreducible Markov chain with transition matrix \( P \) has an invariant distribution \( \pi \) if and only if it is positive recurrent. The invariant distribution is related to the expected hitting times by,

\[ \pi_i = \frac{1}{m_i}. \tag{3.45} \]

**Proof.** Shown in Norris[38, Theorem 1.7.7]. \( \square \)

3.2.10 Convergence to Equilibrium

Intuition suggests that once a recurrent Markov chain has reached steady-state, the Markov state would follow a limiting distribution. However, certain types of periodic
behaviour can prevent this from happening. Consider the case of the 4-state Markov chain in Figure 3.3. This example consists of 4 states configured as a square, where each transition is along one of the two adjacent edges. For an initial state \( X_0 = s_1 \), we have \( X_1 \in \{ s_2, s_4 \} \), then \( X_2 \in \{ s_1, s_3 \} \). Each state will appear only in either odd, or in even time steps. Even though this chain will quickly reach its limiting behaviour, there is no limiting distribution as \( n \to \infty \). The states in this chain are said to be periodic.

In contrast, a state \( i \) is said to be aperiodic if

\[
[P^n]_{i,i} > 0 \quad \forall n > N, \quad (3.46)
\]

for some \( N \in \mathbb{Z} \). This may equivalently be expressed in terms of the greatest common divisor of the set of possible return times, with a state being aperiodic if and only if

\[
gcd \{ n \geq 0 : [P^n]_{i,i} > 0 \} = 1. \quad (3.47)
\]

It is relatively easy to ensure aperiodicity, as shown by Theorem 3.2.5

**Theorem 3.2.5.** If \( P \) is irreducible and has at least one aperiodic state, then all states must be aperiodic.

**Proof.** (From Norris[38, Lemma 1.8.2]) Using (3.46), for the aperiodic state \( i \), there exists an \( N \) where

\[
[P^n]_{i,i} > 0 \quad \forall n > N. \quad (3.48)
\]

As \( P \) is irreducible, for each state \( j \), there exists an \( N_1 \) and \( N_2 \) such that,

\[
[P^{N_1}]_{i,j} > 0, \quad [P^{N_2}]_{j,i} > 0. \quad (3.49)
\]
CHAPTER 3. FUNDAMENTALS OF MARKOV CHAINS

One possible path for \( j \rightarrow j \) is \( j \rightarrow i \rightarrow j \), and so,

\[
[P^n]_{j,j} \geq [P^{N_2}]_{j,i} \times [P^{N}]_{i,i} \times [P^{N_1}]_{i,j}
\]

\[
> 0 \quad \forall n > N_2 + N + N_1.
\]

So any state \( j \) is also aperiodic.

Aperiodicity brings the final element required to ensure that a Markov chain will converge in distribution to an invariant distribution determined by the transition matrix.

**Theorem 3.2.6.** If \( P \) is irreducible, aperiodic, and is known to have an invariant distribution \( \pi \), then it will converge to that invariant distribution. That is:

\[
\lim_{n \to \infty} P(X_n = j \mid X_0 = i) = \lim_{n \to \infty} [P^n]_{i,j} = \pi_j
\]

for all \( i \) and \( j \).

**Proof Outline.** By constructing an auxiliary chain \( \{Y_n\} \) with initial and invariant distribution \( \pi \), it may be shown that the chain \( \{X_n\} \) with transition matrix \( P \) will eventually meet the chain \( \{Y_n\} \) and by a coupling argument, \( \{X_n\} \) have the same limiting distribution[38, Theorem 1.8.3].

Note that while there is not an explicit requirement for the chain to be recurrent, this is implied through the existence of an invariant distribution via Theorem 3.2.4.

### 3.2.11 Ergodic Theorem

While the limit behaviour of \( P^n \) is important in describing the distribution of samples from the chain, it’s not sufficient for the purposes of making inference from a given realisation of a Markov chain. The distribution of \( P^n \) describes the behaviour at time \( n \) across many realisations of the chain. Utilising this result in a Markov chain sampling algorithm requires the evaluation of many Markov chains, with only one sample obtained from each simulation run. This is illustrated in Figure 3.4(a).

In order to reduce computational cost, it is instead preferred to consider many samples from a single Markov chain \( \{X_n\} \), as shown in Figure 3.4(b). The statistics of this type of sampling requires the application of sample path analysis[35].

In order to demonstrate the difference between the behaviours of these two different statistics, it’s useful to consider a pathological example of a Markov Chain.

**Algorithm 3.2.1.**

1. \( x_0 \) is distributed according to some initial distribution \( \lambda \)
2. sample \( x_1 \sim \pi(\cdot) \), the target distribution.
3. sample \( x_k = x_{k-1} \) for \( k > 1 \)


Clearly, for this algorithm, by construction,

$$[P^n]_{i,j} = \pi_j \text{ for } n > 0,$$

(3.53)

and therefore,

$$p(X_n = j \mid X_0 = i) = \pi_j \quad \forall n > 1, \forall i.$$

(3.54)

However, it is quite clearly of no statistical value to sample more than the first two samples from the Markov chain in Algorithm 3.2.1. The samples are too closely correlated in time to be of any benefit. The Markov Chain Monte Carlo methods profiled in this thesis do exhibit time correlation in their samples, so it is important to have a means to demonstrate that their output is useful.

Consequently, stronger requirements are needed for the Markov chain behaviour than provided by Theorem 3.2.6. These are provided by Theorem 3.2.7, which is a strong law of large numbers for samples drawn from a single Markov chain.

**Theorem 3.2.7.** Let $P$ be irreducible and positive recurrent, and $\pi$ be its unique invariant distribution. Then for any bounded function $f : I \to \mathbb{R}$

$$P \left\{ \lim_{n \to \infty} \frac{1}{n} \sum_{k=0}^{n-1} f(X_k) = \overline{f} \right\} = 1,$$

(3.55)

where

$$\overline{f} = \mathbb{E}\{f(X_k)\} = \sum_{i \in I} \pi_i f(i).$$

(3.56)

**Proof.** See Norris[38] Theorem 1.10.2. □

By this theorem, we may use the samples from a single MCMC chain and know that for a sufficiently large number of samples, sample histograms will always converge to the invariant distribution. This is utilised in Markov Chain Monte Carlo methods to calculate functions of the invariant distribution, based on long-run proportion of time spent in each state.
3.3 Markov Processes on Non-Countable Spaces

For the system identification application of this thesis, it’s necessary to consider Markov chains operating on the state space of $\mathbb{R}^n$. As an uncountable space, it isn’t possible to construct a state transition matrix as in the countable case. Attempting to do so will easily result in a matrix of infinite dimension consisting entirely of zero valued elements. Instead of state transition probabilities, we need to look to transition probability densities, and the probability of transitioning to a region of states rather than a specific point in the space.

One could argue that it isn’t necessary to deal with non-countable spaces in stochastic simulation, on account of all digitally sampled data technically being finite in nature. While this is true, it is also very challenging to rigorously deal with the precise nature of the quantisation. The mathematical properties of these systems are much easier to handle in the natural continuous state, and this is the approach taken in this thesis.

3.3.1 Markov Process Definition

As in the case of a Markov chain over a countable space, the chain is again described in terms of a sequence of random variables $\{X_k\}$. Let

$$\{X_k\} = \{X_0, X_1, X_2, \cdots, X_k, \cdots\} \quad (3.57)$$

be a sequence of random variables, with each $X_k \in \mathbb{R}^n$ and where the collection of random variables have a joint probability density function $p(X_0, \cdots, X_N)$ which may be sequentially decomposed as

$$p(X_0, X_1, \cdots, X_N) = p(X_N | X_{N-1}, \cdots, X_0)p(X_{N-1}, \cdots, X_0), \quad (3.58)$$

according to Bayes’ rule. Such a sequence $\{X_k\}$ is termed a Markov process if the conditional probability density $p(X_N | X_{N-1}, \cdots, X_0)$ in (3.58) satisfies

$$p(X_N | X_{N-1}, \cdots, X_0) = p(X_N | X_{N-1}). \quad (3.59)$$

That is, all of the information that can affect future state transition probabilities is contained in the knowledge of the present state. The Markov probability density $p(\cdot, \cdot)$ has a corresponding distribution function $P(\cdot, \cdot)$. $P$ is defined for any Borel measurable set $A \subset \mathbb{R}^n$ and any $x \in \mathbb{R}^n$ as

$$P(x, A) = \int_A p(\xi | x) d\mu(\xi) \quad (3.60)$$

where $\mu$ denotes Lebesgue measure. This represents the probability of transitioning from the state $x$, to any state within the set $A$. This distribution function, termed the transition probability kernel, completely characterises the Markov process[41, p141]. In
3.3. Markov Processes on Non-Countable Spaces

Figure 3.5: State Transition Probability

In order to be a transition kernel, a function $P$ must satisfy two basic requirements:

1. For $\mathcal{B}$ as a Borel sigma algebra on $\mathbb{R}^n$, then $P : \mathbb{R}^n \times \mathcal{B} \rightarrow [0, 1]$ and for any fixed $X \in \mathbb{R}^n$, then $\{\mathbb{R}^n, \mathcal{B}, P(X, \cdot)\}$ is a probability space. Loosely speaking, $\mathcal{B}$ is a set of all possible subsets of $\mathbb{R}^n$, so this requirement ensures that the probability of transitioning from any given point to any given region is in the interval $[0, 1]$.

2. For $P$ to be a transition kernel, it is also necessary that for any fixed $A \in \mathcal{B}$, the function $P(\cdot, A)$ is Lebesgue measurable. This ensures it is possible to integrate over regions of the state space to determine the probability of reaching any set $A$.

The transition kernel performs the same role as the transition probability matrix for a discrete-spaced Markov chain.

3.3.2 Time Evolution of Distribution

The distribution $P(X_{k-1}, A)$ defines the probability of $X_k \in A$ given $X_{k-1}$. However, it’s also of interest to characterise the probability of future states of the Markov chain beyond $k$. For some arbitrary integer $k > 0$, it is of interest to determine the probability of $X_k \in A$ given $X_0$. We denote the distribution that describes this probability as $P_k(X_0, A)$.

In the case of a finite or countable space, this may be conveniently achieved by multiplying the initial state distribution by powers of the state transition matrix as in (3.13). However, without the matrix representation being available for the transition kernel, an alternate approach is necessary.

The equivalent mechanism for an uncountable state space involves a recursive integration across the state space using the transition kernel,

$$P_k(X_0, A) = \int P_{k-1}(\xi, A)P(X_0, d\xi), \quad (3.61)$$

where it is assumed that $P^1(\xi, A) = P(\xi, A)$. Clearly this is less convenient than the matrix representation for countable spaces, but the intuition suggested through the notation $P^k$ may be retained.

3.3.3 Irreducible Processes

In discrete space, an irreducible Markov chain is one where any one state communicates with any other state, as described in Section 3.2.4. From any starting point, the proba-
bility of eventually reaching any state is greater than zero. However, over an uncountable space, this definition isn’t workable. The limitation is that $P(X,A)$ will often be zero for sets $A \in \mathcal{B}$ of measure zero, even in chains which exhibit the practical attributes of irreducibility. For this reason, irreducibility is defined in terms of a measure $\psi$, on $(\mathbb{R}^n, \mathcal{B})$.

**Definition 3.3.1.** The Markov process $P(X,A)$ is termed $\psi$-irreducible [16] if for each $X \in \mathbb{R}^n$ and each $A \in \mathcal{B}$ such that $\psi(A) > 0$, there exists a $k < \infty$ such that

$$P^k(X,A) > 0.$$  \hfill (3.62)

Hence, an irreducible process is one that has a non-zero probability of eventually reaching any set $A$, so long as that set isn’t of measure zero.

The hitting time of a Markov chain is defined similarly to the discrete case, differing only in that the target set $A$ is an element of $\mathcal{B}$, rather than $A \subset I$. The hitting time from a point to a set is defined as

$$H^A \triangleq \inf \{k > 0 : X_k \in A \}.$$  \hfill (3.63)

Again $H^A = \infty$ in the case where $P^k(X,A) = 0$ for all $k > 0$.

Using this definition, irreducibility may alternatively be defined in terms of hitting time [35]. A Markov process is termed $\psi$-irreducible if for any $A \in \mathcal{B}$ such that $\psi(A) > 0$, then

$$P(H^A < \infty | X_0 = X) > 0$$  \hfill (3.64)

for all $X \in \mathbb{R}^n$. However, using the definition of the hitting time (3.63), then (3.64) may be expressed as

$$\inf \{k > 0 : P^k(X,A) > 0 \} < \infty.$$  \hfill (3.65)

for all $\psi(A) > 0$ and all $X \in \mathbb{R}^n$, which is then equivalent to (3.62). Hence these two definitions (3.64),(3.65) of $\psi$-irreducibility are equivalent.

### 3.3.4 Invariant Distributions

The notion of invariance is again important in the analysis of Markov chain behaviour. Essentially we are wanting to construct chains with specific limiting behaviour, and so it is important to find under which conditions that will occur.

Suppose that the $k$th step, $X_k$, of a Markov process $\{X_k\}$ has a distribution $\varphi_k(\cdot)$. After the next state transition, $X_{k+1}$ will have a distribution $\varphi_{k+1}(\cdot)$, which may be determined by integration of the Markov kernel as,

$$\varphi_{k+1}(A) = \int P(\xi,A)\varphi_k(d\xi).$$  \hfill (3.66)

A special case of interest is when the new distribution $\varphi_{k+1}(\cdot)$ is the same as that prior
3.3. Markov Processes on Non-Countable Spaces  

Such a distribution $\varphi(\cdot)$ is termed an \textit{invariant} distribution, and occurs when

$$\varphi(A) = \int \mathbf{P}(\xi, A) \varphi(d\xi). \quad (3.67)$$

Intuitively, due to the Markov property, if ever an invariant distribution is reached, the subsequent states of the chain will continue to be distributed according to that distribution. To demonstrate that an invariant distribution is actually a stationary distribution for the process, we can expand (3.67) by first expanding the measure,

$$\varphi(d\alpha) = \int \mathbf{P}(\xi, d\alpha) \varphi(d\xi) \quad (3.68)$$

and then substituting back into (3.67) with,

$$\varphi(A) = \int \left[ \varphi(d\alpha) \mathbf{P}(\alpha, d\xi) \right] \mathbf{P}(\xi, A) = \int \varphi(d\alpha) \mathbf{P}^2(\alpha, A). \quad (3.69)$$

By continuing to apply the Markov kernel for further state transitions, this may be extended to higher powers as,

$$\varphi(A) = \int \varphi(d\alpha) \mathbf{P}^n(\alpha, A) \quad (3.70)$$

for any $n$. Therefore, the distribution will remain constant for all future state transitions.

3.3.5 Transience and Recurrence

A recurrent state in a Markov chain is one that is visited infinitely often as time extends to infinity. This aspect of recurrence is more involved for a continuous state Markov chain than for one over a countable space. In particular, Harris[22] developed a stronger form of recurrence, which Chan and Geyer in a discussion paper appendicial to [46], describe as overcoming pathological cases from measure theory.

In order to quantify the long-term visit behaviour of a set of states, consider the number of times that the chain returns to a region of the state space. The \textit{occupation time} $\eta_A$ for a set $A \in \mathcal{B}$ is defined as the number of visits the process makes to the set $A$:

$$\eta_A = \sum_{k=1}^{\infty} \chi_A(X_k) \quad (3.71)$$

where $\chi_A(\cdot)$ is the indicator function:

$$\chi_A(x) = \begin{cases} 1 & x \in A \\ 0 & x \notin A. \end{cases} \quad (3.72)$$
Definition 3.3.2. A set $A \in \mathcal{B}$ is termed recurrent if for all $X \in A$,

$$\mathbb{E}_X \{ \eta_A \} = \infty.$$  \hspace{1cm} (3.73)

This expectation may be expressed in terms of the transition kernel as,

$$\mathbb{E}_X \{ \eta_A \} = \sum_{k=1}^{\infty} P^k(X, A).$$  \hspace{1cm} (3.74)

In contrast, a set $A$ is said to be uniformly transient when for all $X \in A$ there exists an $M < \infty$ such that

$$\mathbb{E}_X \{ \eta_A \} \leq M.$$  \hspace{1cm} (3.75)

The main aim of the classification into transience and recurrence is to apply these concepts to the whole process, rather than to individual sets. Suppose now that that $\mathbf{P}(X, A)$ is $\psi$-irreducible. If $A$ is recurrent for all sets $A \in \mathcal{B}$ where $\psi(A) > 0$, then $\mathbf{P}(X, A)$ is termed recurrent. Otherwise it may be shown that $\mathbb{R}^n$ can be covered with countably many sets, all of which are uniformly transient[35, Theorem 8.0.1]. In that case, the process $\mathbf{P}(X, A)$ is termed transient. Due to this dichotomy, as with countable chains, it makes sense to speak of a recurrent process or of a transient process.

An important aspect of recurrence of a chain is that it can demonstrate the presence of a unique invariant distribution.

**Theorem 3.3.1.** If a process defined by a Markov kernel $\mathbf{P}(X, A)$ is recurrent, then it has a unique invariant density $\varphi_k(\cdot)$.

**Proof.** See Theorem 10.4.4 of [35]. \hfill \Box

### 3.3.6 Harris Recurrence

As described above, for a Markov process to be recurrent, it is necessary that for any set $A \in \mathcal{B}$, it satisfy (3.75). Namely, the expected number of return visits back to $A$ must be infinite.

Harris recurrence is a stronger condition where not only must the expected number of return visits be infinite, but also an infinite number of return visits occur with probability one.

More specifically[41, Definition 4.4.8], a set $A \in \mathcal{B}$ is termed Harris recurrent if

$$\mathbf{P}_X(\eta_A = \infty) = 1, \hspace{1cm} \forall X \in A,$$  \hspace{1cm} (3.76)

where $\mathbf{P}_X$ represents probability given an initial state $X \in A$. A process $\mathbf{P}(X, A)$ is then called Harris recurrent if it is $\psi$-irreducible and every set such that $\psi(A) > 0$ is Harris recurrent.

In the countable state space case, in Section 3.2.6, recurrence was defined in terms of the hitting time of states within the chain. The equivalent definition of recurrence in an
3.3. Markov Processes on Non-Countable Spaces

An uncountable space is that if for each $A \in \mathcal{B}$, and each $x \in A$,

$$P\{H^A_x(\omega) < \infty\} = 1.$$  \hspace{1cm} (3.77)

Under this condition, it may be shown that (3.76) holds\[41, Prop. 4.4.9], and then $(X_n)$ also meets the stronger criteria of Harris recurrence.

### 3.3.7 Positive Process

In Section 3.3.4, a process $P$ was defined to be invariant when there existed a distribution $\varphi(\cdot)$ such that,

$$\varphi(A) = \int P(\xi, A) \varphi(d\xi).$$  \hspace{1cm} (3.78)

One may initially be motivated to construct a chain such that it has a particular invariant distribution, simply for the utility of being able to draw samples from that invariant distribution. However, the presence of an invariant distribution also controls the properties of the chain. Intuitively, the existence of an invariant distribution prevents the possibility of the probability mass escaping to infinity$[41]$. This is formalised in the notion of positive recurrence, and its connection to recurrence.

**Definition 3.3.3.** If a process $P(X, A)$ is $\psi$-irreducible for some measure $\psi$, and also admits an invariant probability measure $\varphi$, then $P(X, A)$ is called a positive process$[35]$.

For chains over a countable space, positive recurrence was instead defined in Definition 3.2.1 in terms of the expected return time. However, in that case Theorem 3.2.4 established an equivalence to that described here.

**Theorem 3.3.2** ([35], Prop 10.1.1). If a process defined by a Markov kernel $P(X, A)$ is positive, then it is recurrent.

**Proof.** Suppose the process is instead uniformly transient. In this case there exists a countable cover $\{A_j\}$ of $\mathbb{R}^n$, where each of the $A_j$ are uniformly transient. By (3.74) and (3.75), for some finite set $\{M_j\}$,

$$\sum_{k=1}^{\infty} P^k(X, A_j) \leq M_j.$$  \hspace{1cm} (3.79)

Since $P(X, A)$ is positive, and hence $\varphi$-invariant for some non-trivial $\varphi$, then

$$\varphi(A_j) = \int P(\xi, A_j) \varphi(d\xi).$$  \hspace{1cm} (3.80)

Therefore, using the property of invariance in (3.70),

$$m \cdot \varphi(A_j) = \sum_{k=1}^{m} \int P^k(\xi, A_j) \varphi(d\xi).$$  \hspace{1cm} (3.81)
Then by (3.79),
\[ m \cdot \varphi(A_j) \leq M_j. \] (3.82)
Since this is bounded above by finite \( M_j \) for arbitrarily large \( m \), then \( \varphi(A_j) = 0 \) for all \( A_j \), which implies that \( \varphi \) is trivial, and \( \mathbf{P}(X,A) \) cannot be a positive process. So by contradiction, if \( \mathbf{P}(X,A) \) is positive then it cannot be transient. \( \square \)

### 3.3.8 Harmonic Functions

While a process may be assured of being recurrent via the existence of an invariant measure, another mechanism is needed in order to establish the stronger property of Harris recurrence, that will later be required. One approach to doing this is by using known results regarding a class of functions known as harmonic functions.

**Definition 3.3.4.** A function \( h : \mathbb{R}^n \rightarrow \mathbb{R} \) is termed harmonic with respect to the measure \( \mathbf{P}(X,A) \) if, for all \( X \in \mathbb{R}^n \)
\[ h(X) = \int h(\xi)\mathbf{P}(X,d\xi). \] (3.83)

Intuitively, the integral in (3.83) is the expected value of \( h(X_{k+1}) \), given a known state \( X_k \). A harmonic function is then a function whose expected value does not change across a state transition for any given starting state.

Clearly if a function \( h(X) = \bar{h} \) is a constant everywhere, then,
\[ \int \bar{h}\mathbf{P}(X,d\xi) = \bar{h} \int \mathbf{P}(X,d\xi) = \bar{h}, \] (3.84)
by the law of total probability. So trivially, all constant functions are harmonic functions. Of interest are any possible non-constant functions that are harmonic. For a Harris recurrent process, there is a result that demonstrates that no other harmonic functions exist.

**Theorem 3.3.3** (Theorem 2 of [46]). Suppose that \( \mathbf{P}(X,A) \) is recurrent. Then it is Harris Recurrent if and only if every bounded harmonic function is a constant.

This condition is both necessary and sufficient, and so it may also be used as a means to establish whether or not a process is Harris recurrent.

A related result exists for all recurrent processes, which demonstrates that the difference between recurrence and Harris recurrence can be characterised by the nature of its harmonic functions.

**Theorem 3.3.4** (Proposition 3.13 of [39]). Suppose that \( \mathbf{P}(X,A) \) is recurrent. Then every function \( h \) that is harmonic with respect to \( \mathbf{P}(X,A) \) is constant \( \psi \) almost everywhere.
Later in this thesis, these results relating to harmonic functions are used to show that a broad class of Markov Chain Monte Carlo algorithms are Harris recurrent.

### 3.3.9 Law of Large Numbers

The aim of the Markov chain discussion in this chapter has been to define a set of properties required in order to establish sample path ergodic results. In order to show that a sample realisation \( \{X_k\} \) from a given Markov chain may be used as a source of samples for a Monte Carlo integration, it’s necessary to be able to prove convergence of sums of the form

\[
S_N^f = \frac{1}{N} \sum_{k=1}^{N} f(X_k), \tag{3.85}
\]

for somewhat arbitrary functions \( f \) of the Markov state.

The property of Harris recurrence is important as it may be used to establish this important aspect of ergodic behaviour. Via Theorem 3.3.5, it can be determined that sample sums of functions from a distribution converge to the distribution itself.

**Theorem 3.3.5 (Law of Large Numbers).** Suppose that \( \{\theta_k\} \) is a realisation of a positive Harris recurrent chain with invariant distribution \( \varphi \). Suppose that \( f : \mathcal{X} \to \mathbb{R} \) is such that

\[
\int |f(\xi)| \varphi(d\xi) < \infty. \tag{3.86}
\]

Then

\[
\lim_{N \to \infty} \frac{1}{N} \sum_{k=1}^{N} f(X_k) = \int f(\xi) \varphi(d\xi), \quad \forall X_0 \in \mathcal{X} \tag{3.87}
\]

with probability one.

**Proof.** Follows by Theorem 17.1.7 of [35]. \( \square \)

### 3.3.10 Total Variation Norm

In order to further assess the convergence properties of Markov chains on continuous spaces, it’s necessary to have a means of measuring the similarity between two probability density functions. There are a range of different norms available for this purpose, each with their own properties. In Markov chain theory, the Total Variation Norm can be a useful choice to demonstrate distributional convergence properties.

Supposing that \( \mu \) is a measure on \( \{\mathbb{R}^n, \mathcal{B}\} \), then the total variation norm \( \|\mu\| \) is defined as [35, Page 311]

\[
\|\mu\| = \sup_{f : |f| \leq 1} \left| \int \mu(d\xi) f(\xi) \right| \tag{3.88}
\]

\[
= \sup_{A \in \mathcal{B}} \mu(A) - \inf_{A \in \mathcal{B}} \mu(A). \tag{3.89}
\]
Using this definition, the total variation norm for the difference between two probability measures $P^n(X, \cdot)$ and $\varphi(\cdot)$ is given as

$$
\| P^n(X, \cdot) - \varphi(\cdot) \| = \sup_{A \in B} (P^n(X, A) - \varphi(A)) - \inf_{A \in B} (P^n(X, A) - \varphi(A)) \quad (3.90)
$$

$$
= 2 \sup_{A \in B} |P^n(X, A) - \varphi(A)|. \quad (3.91)
$$

To evaluate this norm, one may find the set $A$ over which the difference between the density functions is maximised. This consists of the event for which the two distributions differ the most. The total variation norm is then the difference in probably of $A$ according to the respective density functions. In the case of marginal probability density functions $p$ and $q$, let $A$ be the event describing the set where $p$ ascribes a greater probability than $q$, so

$$
A = \{ x : p(x) > q(x) \}. \quad (3.92)
$$

Let $A^*$ be the converse,

$$
A^* = \{ x : p(x) < q(x) \}. \quad (3.93)
$$

By the law of total probability

$$
\int p(x) - q(x) \, dx = 0, \quad (3.94)
$$

and so $P(A) = P(A^*)$. This provides a straightforward mechanism to evaluate the total variation norm for a marginal density function, as

$$
\| P(\cdot) - Q(\cdot) \| = 2 \sup_{A \in B} \| P(A) - Q(A) \| \quad (3.95)
$$

$$
= \sup_{A \in B} \| P(A) - Q(A) \| + \sup_{A^* \in B} \| P(A) - Q(A) \| \quad (3.96)
$$

$$
= \int_A p(x) - q(x) \, dx + \int_{A^*} q(x) - p(x) \, dx \quad (3.97)
$$

$$
= \int |p(x) - q(x)| \, dx. \quad (3.98)
$$

### 3.3.11 Aperiodicity

The periodicity of a Markov chain was introduced in Section 3.2.10. The natural extension of this to chains over uncountable spaces would be to consider whether for all $X \in \mathbb{R}^n$

$$
g.c.d. \left\{ k \geq 1 : P^k(X, X) > 0 \right\} = 1. \quad (3.99)
$$

Here g.c.d. is the greatest common divisor. The aim of this condition would be to ensure that there do not exist only certain specific time points at which the process can return
to a state. Rather, for an aperiodic chain it is possible to return to a state at any time.

However, in a continuous state setting, the set including only state $X$ is of measure zero, and so we can expect $P_k(\cdot, \{X\}) = 0$ for all $X$ and $k$. This means that a meaningful definition of periodicity for a Markov chain over a continuous state must involve transition probabilities to sets of states, rather than individual states themselves.

The analogue to discrete-state aperiodicity involves considering the return to particular set of states. For this we introduce the terminology of a small set. A set $C$ is called a $\nu_m$-small set for a kernel $P$ if there is an $m > 0$ and a non-zero measure $\nu_m$ with

$$P^m(x, A) \geq \nu_m(A) \quad \forall x \in C, \forall A \in \mathcal{B}. \quad (3.100)$$

Take now $C$ to be any $\nu_M$-small set, with $\nu_M(C) > 0$. Such a measure $\nu_M$ will exist for any $\psi$-irreducible chain [Prop 5.2.4] [35]. This means that a chain starting in $C$ has a positive probability of returning again to $C$ at time $M$. With this $C$, construct the set,

$$E_C = \{ m \geq 1 : \exists \delta_m > 0 \text{ where } C \text{ is } \nu_m \text{-small for some } \nu_m = \delta_m \nu_M \} \quad (3.101)$$

This represents the set of transitions where there is a positive probability of return to the $\nu_M$-small set $C$. The period of the set $C$ is defined as the g.c.d. of $E_C$. It can be shown [35, Theorem 5.4.4] that the period of this arbitrary set $C$ is the same as that of any small set in the chain, and hence is a property of the chain itself.

A $\psi$-irreducible process is termed to be aperiodic if it has a period of 1.

### 3.3.12 Convergence Of Distribution

The importance of aperiodicity is that when it holds in addition to recurrence, then a chain converges to a stationary distribution which is its invariant measure.

**Theorem 3.3.6.** Suppose that $P(A | \theta)$ is a $\psi$-irreducible and aperiodic recurrent chain with invariant measure $\varphi$. Then for $\varphi$ almost all $\theta$

$$\lim_{n \to \infty} \sup_{A \in \sigma(\mathcal{X})} |P^n(A | \theta) - \varphi(A)| = 0. \quad (3.102)$$

Furthermore, if the chain is Harris recurrent, then (3.102) holds for any initial condition $\theta \in \mathcal{X}$.

**Proof.** The first part not requiring Harris recurrence, is Theorem 1 of [46]. The latter part requiring Harris recurrence is established via [35, Proposition 13.0.1].

Finally, if the distributional convergence in (3.102) is sufficiently rapid, then sample averages of realisations $\{\theta_k\}$ from the Markov chain $P(A | \theta)$ obey a Central Limit Theorem.
**Theorem 3.3.7.** Suppose that $P(A \mid \theta)$ is a $\psi$-irreducible aperiodic Harris recurrent chain with invariant measure $\varphi$ and suppose that for some $M < \infty$, $r \in (0,1)$

$$\sup_{A \in \sigma(X)} |P^n(A \mid \theta) - \varphi(A)| \leq Mr^n. \quad (3.103)$$

Let $f : \mathcal{X} \to \mathbb{R}$ be bounded. Then the limit

$$\sigma_f^2 \equiv \lim_{N \to \infty} \frac{1}{N} \mathbb{E}_\varphi \left\{ \left[ \sum_{k=1}^N f(\theta_k) - \mathbb{E}_\varphi \{f\} \right]^2 \right\} \quad (3.104)$$

exists and is finite, where $\mathbb{E}_\varphi \{\cdot\}$ denotes expectation with respect to the measure $\varphi$. Furthermore, if $\sigma_f^2 > 0$ then

$$\sqrt{N} \left( \frac{1}{N} \sum_{k=1}^N f(\theta_k) - \mathbb{E}_\varphi \{f\} \right) \xrightarrow{D} \mathcal{N}(0, \sigma_f^2) \quad (3.105)$$

as $N \to \infty$.

**Proof.** See Theorem 17.0.1 of [35].

### 3.4 Conclusion

Despite being based on a very simple structure, there are many useful properties of Markov processes. In particular, the law of large number results are valuable in proving the convergence of estimates from MCMC algorithms. The results presented here form the building-blocks for the analysis of MCMC algorithms, and establish the conditions required in order to ensure convergence.
Chapter 4

Soft Detection in Multi-User Communications

4.1 Introduction

This chapter will deal with an application of Markov Chain Monte Carlo methods to physical-layer digital communications. In this case, the probability distribution of transmitted symbols will be estimated, given the measurement of the received signal. This is an example of the use of MCMC methods in the simpler case of systems taking on a discrete state. However, principles learnt from this application may be carried over to the more involved continuous state case, as shown in the subsequent chapter.

4.2 Digital Communications Systems

The physical layer of a communications system involves the transfer of digital data across a noisy analogue communications channel. It consists of that channel, together with a transmitter and a receiver. The components of these are shown in the block diagram in Figure 4.1. The channel itself may consist of a transmission line, optical fibre, or, as in the main application presented here, wireless electromagnetic propagation. The
Source Encoder is employed to efficiently represent the incoming binary information, and is essentially a data compression stage to remove redundancy. The Channel Encoder is included to allow error detection and correction, and to do this it adds additional redundancy into the data in an efficient manner. The Modulator then takes the coded data and generates an analogue representation that may be transmitted across the channel.

The receiver contains corresponding modules to reverse the operations of the transmitter, and produce as its output a copy of the input data. This thesis considers the application of MCMC methods in the demodulation component of the receiver.

### 4.2.1 Suboptimal Demodulation

The demodulation process involves estimating the transmitted data based on the received signal. It is an optimisation process where the objective is to best estimate the transmitted signal prior to modulation. However, in many situations, such as will be described in Section 4.5, the optimal demodulator has a prohibitively high complexity. Consequently, there has been significant effort put into developing suboptimal detectors which trade-off some performance for the sake of feasible computational complexity [48, 24]. Recently, new suboptimal approaches have been developed that utilise Monte–Carlo based averaging of simulated quantities and have been illustrated as capable of impressive performance [15, 8, 51, 52, 55, 11].

This chapter considers the application of Markov-chain Monte–Carlo methods as a means of performing multi-user detection. It will be shown that the process of detection involves operations akin to estimating the parameters of a stochastic model. The “parameters” in this case happen to be the symbols received at the modulator. By selecting the desired symbol posterior as the target density, the MCMC methods will provide the means to deduce the Maximum a Posterior (MAP) estimate from sample averages, together with the probability distribution for each possible symbol.

With such a simulation-based approach, it is clearly important to have a good understanding of not only the accuracy, but also the computation time. This will involve analysis of the bit-error rate, and the number of simulated samples that need to be drawn to achieve it. Such convergence rates have been considered in previous work. For example, in the work [8, 51, 52, 55, 11] there is considerable empirical evidence showing good detection performance with only a modest number of algorithm iterations.

However, the theoretical analysis and understanding of the methods underpinning the MCMC-based approach is much less developed. While the above-mentioned works acknowledge that convergence of the methods they develop can be understood by results from Markov chain theory, this is argued only in fairly general terms. The most complete analysis to date is provided in [7], which gives important insights. The bulk of attention has also been given to Gibb’s sampling approaches, but a contribution of this chapter is that it considers the wider class of Metropolis–Hastings approaches, of which the Gibb’s sampler is a special case.
4.3 Multiple Access Channels

With many communications channels, particularly in wireless applications, it’s necessary that multiple transmitters simultaneously send data in such a way that each original stream may still successfully be decoded. In this situation a means must be devised for a receiver to be able to separate the individual data streams.

There are several different ways of constructing a multiple-access channel. Frequency Division Multiple Access (FDMA) achieves this by using a different frequency band for each of the transmitters. The simple application of a band-pass filter structure at the receiver is sufficient to separate out the desired signal from the interference. A wide variety of applications use this approach, including most analogue communications systems.

Time Division Multiple Access (TDMA) involves dividing the time axis into segments and assigning particular segments for each transmitter. This requires accurate timing to be established between each of the transmitters and receivers. An example of TDMA is the GSM mobile telephony standard.

In Code Division Multiple Access (CDMA), the signals from the different transmitters overlap in both time and frequency. They are differentiated instead by means of modulating against unique signature sequences, denoted here by $s_n(t)$, with the integer $n$ indexing among multiple signatures.

By way of example, a possible pair of signature sequences is shown in Figure 4.2 for a two-user system. In order to send a binary 1, a transmitter $n$ transmits $+s_n(t)$, while to send a 0, it transmits $-s_n(t)$. The channel will then combine the modulated sequences from each of the users.

![Figure 4.2: Example signature sequences.](image)

For an ideal case, where the channel forms a simple addition, and there is perfect synchronisation between the two transmitters, the set of possible outputs is easily determined. The four possible received signals for each combination of inputs are shown in Figure 4.3.

By inspection, each of these cases is clearly distinguishable, and so the transmitted bits may be uniquely determined. Intuitively, the transmission of the first user may be determined by the change in amplitude between the first and second halves of the symbol period, while the transmission of the second user can be determined by the average amplitude. Even though the signals from each of the transmitters overlap in
both the time and frequency domains, it is clearly possible to decode the original pair of
data bits sent if the signature sequences are known at the receiver.

In a more realistic case, the use of these signature sequences still allow successful
detection, but the detection process is significantly complicated by increasing system
dimension and the introduction of noise in the channel. Consequently, it is necessary to
formalise the system model in order to devise an optimisation scheme.

4.3.1 CDMA Signal Model

In this thesis, a baseband CDMA signal model is used to model the CDMA physical
layer. In this model, the n’th each transmitter over a symbol period \( t \in [0, T] \) will emit
an output of,

\[
y_n(t) = x_n s_n(t),
\]

where \( x_n \in \{-1, +1\} \) is the symbol transitted by the n-th user, and \( s_n(t) \) is the signature
waveform assigned to the n-th user. In order that the energy per symbol is maintained
equal for all symbols, the signature waveform is normalised so that,

\[
\int_0^T s_n^2(t) \, dt = 1.
\]

There are many available possibilities for the choice of the signature waveforms \( s_n(t) \).
In this thesis, the emphasis will be on direct-sequence spread spectrum (DS-CDMA),
where signature waveforms fit to a defined structure. This structure consists of a sequence
of \( N_c \) binary-valued chips, each of duration \( T_c \), for a combined sequence length of \( T = N_c T_c \). In the simplest case, a chip consists of a rectangular pulse of duration \( T_c \), but it
may also take on a smoother shape with better spectral efficiency, such as a sinc function.

The direct-sequence signatures in Figure 4.2 consist of two chips of rectangular
pulses. The underlying binary sequences are \( \{1, 0\} \) and \( \{1, 1\} \) respectively. In practice,
the sequences are much longer, and are likely to be characterised by a pseudo-random
sequence[48].
4.3. Multiple Access Channels

4.3.2 Matched Filter Model over Additive White Gaussian Noise Channel

At the receiver, the received signal is assumed to be the sum of the set of antipodally modulated synchronous signature waveforms, but corrupted with additive white Gaussian noise (AWGN). This is represented as the summation,

\[ y(t) = \sum_{n=1}^{K} A_n x_n(t) s_n(t) + \sigma \nu(t), \quad 0 \leq t \leq T, \quad (4.3) \]

where \( K \) is the number of active users and \( A_n \) is the received amplitude of the \( n \)-th user. \( A_n \) is a function of both the transmit power, and the attenuation of the channel. In this model, the additive noise \( \nu(t) \) in (4.3) is white Gaussian, with unit power spectral density. Hence, the single-sided power spectral density of noise at the receiver is given by

\[ N_0 = 2\sigma^2. \quad (4.4) \]

A conventional approach to the detection of CDMA multi-user channels is to employ matched filters at the receiver to estimate the transmitted symbols of each transmitter. The matched filter is implemented as a correlation between the received signal and the signature sequence. This correlation spans each symbol period, and the result \( y_n \) is the matched filter output corresponding to the \( n \)-th user [48],

\[ y_n = \int_0^T y(t)s_n(t) \, dt \]

\[ = A_n x_n + \sum_{j \neq n} A_j x_j \rho_{jn} + \nu_n, \quad (4.6) \]

where

\[ \rho_{ij} = \int_0^T s_i(t)s_j(t) \, dt, \quad (4.7) \]

\[ \nu_n = \sigma \int_0^T \nu(t)s_n(t) \, dt, \quad (4.8) \]

and \( x \) is the vector of transmitted signals. In the case of a system with orthogonal signature sequences, the cross-correlation terms are zero, with

\[ \rho_{ij} = 0 \text{ for all } i \neq j. \quad (4.9) \]

Where this occurs, (4.6) reduces to

\[ y_n = A_n x_n + \nu_n, \quad (4.10) \]
eliminating interference from other transmitters into each matched filter. While complete orthogonality is not achieved in the general case, clearly it is an aim of signature sequence design that the cross-correlation terms $\rho_{ij}$, for $i \neq j$, be minimised, so as to reduce the interference from other users in each matched filter output.

### 4.3.3 Matrix Form

The baseband model (4.5)–(4.8) for $n = 1, \cdots, K$ may be represented in the generic matrix form of,

$$ y = Hx + \nu, $$

(4.11)

where $x$ represents the transmitted symbols, and $y$ the matched filter outputs. The matrix $H$ may be decomposed as

$$ H = \Gamma A, $$

(4.12)

where,

$$ \text{Cov}\{\nu\} \triangleq R = \sigma^2 \Gamma, $$

(4.13)

$$ [\Gamma]_{i,j} = \rho_{ij}, $$

(4.14)

$$ A = \text{diag}\{A_n\}, $$

(4.15)

and the vectors,

$$ y \triangleq [y_1, \ldots, y_K]^T, $$

(4.16)

$$ x \triangleq [x_1, \ldots, x_K]^T, $$

(4.17)

$$ \nu \triangleq [\nu_1, \ldots, \nu_K]^T. $$

(4.18)

The detection task is to estimate the original transmitted symbols $x$, given the measured matched filter outputs $y$ and the estimate of the channel properties $H$. The estimates, $\hat{x}$, of the transmitted symbols may take the form of either hard-decision binary values, or a probability mass function for each of the possible transmitted values. The soft probabilistic values can be of significant value as inputs into a channel decoder. A feature of the class of Bayesian detectors presented in this chapter is that these probabilities are inherently available.

As a part of this process, knowledge of the distribution of the noise $\nu$ is required for accurate soft output estimates. It will be shown in Section 6.6 that this may also be estimated.
4.4 Detection Decision Schemes

Given the baseband model in (4.11), the required detection task involves solving this equation for \( x \), given received information \( y \). However, it is first necessary to develop a criteria to determine which estimate \( \hat{x} \) provides the best solution.

The role of a detector is, given each \( y \), to find an estimate \( \hat{x} \) for the transmitted bits that minimises the probability of errors in detection. This may be achieved by choosing \( \hat{x} \) to maximise the probability of correct decoding,

\[
\hat{x} = \arg \max_w P\{w = x\}. \tag{4.19}
\]

This involves a partition of the observation space that forms the span of \( y \), into a set of decision regions\[48, \text{Prop 3.1}\]. Each decision region \( \Omega_i \), corresponds to a hypothesis that \( i = x \). For each region, there then exists a known probability density function \( p(y \mid i) \)\ (4.20)

to describe the distribution of \( y \) under that hypothesis.

If \( p(i) \) is the a priori probability of \( x = i \), the probability of correct decoding for any partitioning scheme will be,

\[
 p_c = \sum_i p(i) \int_{\Omega_i} p(y \mid i) \, dy \tag{4.21}
\]

If there are a total of \( m \) equiprobable decision regions, then this probability can be evaluated as,

\[
 p_c = \frac{1}{m} \sum_{i=1}^m \int_{\Omega_i} p(y \mid i) \, dy \tag{4.22}
\]

\[
 \leq \int_{\Omega_i} \max_j p(y \mid j) \, dy, \tag{4.23}
\]

with \( i = \arg \max_j p(y \mid j) \). The inequality in (4.23) then represents a bound on the performance of the best possible detection scheme for minimising the probability of bit error.

4.4.1 Maximum Likelihood

If a detection scheme can be found that reaches the bound in (4.23), then for the case of equiprobable decision regions, it will be optimal for the bit error criteria. By construction, this may be achieved if the boundaries of the decision regions are chosen so that,

\[
\Omega_i = \{ y : p(y \mid i) = \max_{j=1,\ldots,m} p(y \mid j) \}. \tag{4.24}
\]
This definition may result in overlapping regions, but where this occurs the regions may be arbitrarily reduced in size to form a partition, where the same bit error performance is retained regardless of this choice. The decision scheme in (4.24) is known as Maximum Likelihood (ML) estimation.

4.4.2 Maximum a Posteriori

The more general detection case is where the decision regions \( \Omega_i \) are not equiprobable. In this situation, there is a known a priori probability mass function \( p(x) \), describing the unconditional probabilities of the transmitted data. Using Bayes’ theorem, the a posterior probabilities may be calculated with

\[
p(x | y) = \frac{p(y | x)p(x)}{p(y)}. \tag{4.25}
\]

The extension of the ML decision method in (4.24) to provide optimal bit-error behaviour when prior information is available, is to instead to define the decision regions to maximise \( p(x | y) \). This is known as maximum a posterior (MAP) estimation.

4.5 CDMA Detection Methods

The simplest approach to achieving multi-user detection (MUD), is to directly use the sign of the matched filter outputs in (4.5). By the decomposition in (4.6), the matched filters provide an estimate of the \( x_n \) sent, with the only estimation error being due to the corruption by the noise and the interference through the \( \rho_{ij} \) cross-correlation terms.

While the cross-correlation of the signature sequences can be ignored, and the matched filter outputs used directly, doing so comes with a penalty in bit error rates, with the bound in (4.23) not being reached. Given that that the maximum a-posteriori (MAP) estimator provides the optimal estimate of the transmitted data, a logical choice is for \( \hat{x} \) to instead be estimated according to,

\[
\hat{x} = \arg\max_{x \in \{-1, +1\}^K} p(x | y). \tag{4.26}
\]

The probability \( p(x | y) \) may be evaluated by applying Bayes’ theorem, and evaluating \( p(y | x) \) using the matched filter model in (4.11),

\[
p(x | y) = \frac{p(x)}{p(y)}p(y | x) \tag{4.27}
\]

\[
= \frac{p(x)}{p(y)} p_\nu(y - Hx), \tag{4.28}
\]

where \( p_\nu(\cdot) \) is the joint probability density function of the elements in the noise vector \( \nu \), and \( p(y) \) is a quantity normalising the total probability to be one.
4.6 MCMC Methods

For the particular case where $p(x)$ assigns equal prior probability to all the transmitted bits, and for an assumption of $p_Y$ being Gaussian, then (4.28) takes on the well known explicit form of

$$p(x \mid y) = \frac{\text{const}}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{R}^{-1/2}(y - \mathbf{H}x)\|^2\right),$$

(4.29)

where $\| \cdot \|$ denotes the Euclidean norm. Note that Gaussian density is given by way of example. The algorithms presented here will be suited to the common Gaussian noise assumption, but may equally be applied to other noise models with a known density function.

In this Gaussian case, the MAP estimate $\hat{x}$ is given by

$$\hat{x} = \arg\min_{x \in \{-1,+1\}^K} \|\mathbf{R}^{-1/2}(y - \mathbf{H}x)\|^2.$$  

(4.30)

The challenge imposed by this estimate is that $x$ takes on discrete values. The minimisation of (4.30) is therefore a combinatorial optimisation problem requiring, in general, an exhaustive search over the $2^K$ possible combinations of transmitted bits. This is a general issue with the MAP optimisation problem as presented in (4.26) for binary valued vectors. Consequently, the optimal estimate exhibits computational complexities which increase exponentially with the number of users in a CDMA system [47]. Over recent years, this has motivated the search for computationally cheaper solutions, and here we look to MCMC methods as a means of efficiently reaching a good estimate of the MAP detector.

4.6 MCMC Methods

This thesis considers an MCMC approach to the MAP estimator that has been proposed by several authors [15, 8, 51, 52, 55]. An MCMC estimator involves generating a realisation $\{x_1, \ldots, x_k, \ldots, x_L\}$ of a Markov Chain with limiting distribution equal to the desired posterior in (4.28). For any given initial $x_0$, the required limiting probability distribution of $x_k$ is

$$\lim_{k \to \infty} p(x_k = x \mid x_0) = p(x \mid y).$$  

(4.31)

This simulated realisation $\{x_k\}$ is then used as if it were a random sample from the density $p(x \mid y)$. As $x_k$ is contained in a countable space, Theorem 3.2.6 may be used to establish this convergence limit for an appropriately constructed Markov chain. Then the strong law of large numbers property in Theorem 3.2.7 may be used to show that the sequence leads to consistent estimates of various quantities.

For example, for an arbitrary measurable function $f$, MCMC algorithms allow the numerical computation and consistent estimation of the conditional expectation $E\{f(x) \mid y\}$
as
\[ E\{f(x) \mid y\} = \sum_x f(x)p(x \mid y) \approx \frac{1}{L} \sum_{k=\ell+1}^{\ell+L} f(x_k). \tag{4.32} \]

Here \( \ell \) is the length of an initial “burn-in” period of the algorithm, where samples are discarded to allow a convergence away from the initial state of the chain. In particular, this approach allows the computation of rather arbitrary posterior densities according to
\[ p(f(x) \in F \mid y) \approx \frac{1}{L} \sum_{k=\ell+1}^{\ell+L} \chi_{f^{-1}(F)}(x_k), \tag{4.33} \]

where \( \chi \) is the indicator function and \( F \) is a \( f \)-measurable set. The notation \( f^{-1}(\cdot) \) represents the pre-image of a set under \( f \), defined as,
\[ f^{-1}(F) = \{x : f(x) \in F\}. \tag{4.34} \]

Note that \( f \) need not be invertible, and when using the MCMC algorithm, there is no need to calculate an inverse of \( f \). The evaluation of each summation term in (4.33) involves only a computation of \( f(x_k) \) in order to evaluate the set membership. Essentially, the right hand side of (4.33) is simply the computation of a sample histogram of \( f(x) \).

While the proceeding chapter has demonstrated how it’s possible to prove that a Markov chain may have the required properties for a sampling estimator, it isn’t necessarily clear how to successfully construct such a chain for an arbitrary multivariable posterior density \( p(x \mid y) \). The Markov chain Monte Carlo methods, as introduced in Section 2.6, provide a means to achieve this.

### 4.6.1 Metropolis–Hastings

The Metropolis–Hastings algorithm is one of the most prominent MCMC methods. Algorithm 4.6.1 describes the Metropolis–Hastings algorithm as it is applied to Multi-User Detection. Each iteration produces a sample \( x_k \), which may be used as if a sample from \( p(x \mid y) \).

While this is the definition of the general algorithm to be used here, the specifics of the implementation are also important to the MUD application. The choice of the proposal density \( \gamma(\cdot) \) greatly impacts the performance of the algorithm, and a proposal density choice is developed in this chapter that offers a dramatic reduction in per-iteration computation of the Metropolis algorithm.

A particular special case of the Metropolis–Hastings algorithm will be used in this thesis, which happens to also be the original Metropolis algorithm, as described in Section 2.6.2. For this case, the proposal density is symmetric with \( \gamma(\xi \mid x) = \gamma(x \mid \xi) \), resulting in the relationship
\[ \frac{\gamma(x_{k-1} \mid \xi_k)}{\gamma(\xi_k \mid x_{k-1})} = 1. \tag{4.37} \]
4.6. MCMC Methods

Algorithm 4.6.1 (Metropolis–Hastings).

1. Initialise $x_0$ at some value such that $p(x_0 | y) > 0$.

2. At iteration $k$, consider a candidate value $\xi_k$ which is drawn from a proposal density $\gamma(\cdot | x_{k-1})$. That is, find a realisation $\xi_k$ of $x_k$ as

$$
\xi_k \sim \gamma(\cdot | x_{k-1}).
$$

3. Compute the acceptance probability

$$
\alpha(\xi_k | x_{k-1}) = \min \left\{ 1, \frac{p(\xi_k | y) \cdot \gamma(x_{k-1} | \xi_k)}{p(x_{k-1} | y) \cdot \gamma(\xi_k | x_{k-1})} \right\}.
$$

4. Accept the proposed candidate $x_k = \xi_k$ with probability $\alpha(\xi_k | x_{k-1})$, otherwise leave $x_k$ unchanged and discard the candidate.

5. Increment $k$ and if $k < \ell + L$ return to step 2 otherwise terminate.

Consequently, the acceptance probability (4.36) may be simplified to

$$
\alpha(\xi_k | x_{k-1}) = \min \left\{ 1, \frac{p(\xi_k | y)}{p(x_{k-1} | y)} \right\}.
$$

4.6.2 Gibbs Sampling

Another common MCMC method is Gibbs Sampling. The use of Gibbs Sampling involves the partitioning of the Markov state vector into a number of sub-blocks, and drawing from tractable marginal probability density functions.

For the purposes of generating a proposal density in either Gibbs Sampling, or in the Metropolis algorithms, the vector $x$ may be divided into a set of sub-blocks, as shown in Figure 4.4. The $i$-th sub-block of $x$ is denoted as $x^i$, and the notation $x^{-i}$ is used to denote the complement of this, namely everything except the $i$-th sub-block.

![Figure 4.4: Subblock division of vectors.](image)

The Gibbs Sampling specialisation of Algorithm 4.6.1 occurs in the situation where it is possible to divide $x$ into a partition of sub-blocks and to draw samples from the proposal density

$$
\gamma(\xi_k^i | x_{k-1}) = p(\xi_k^i | x_{k-1}^{-i}, y).
$$
When a block $i$ is proposed in this way, the remainder of $\xi$ is chosen by retaining

$$\xi^{-i}_k = x^{-i}_{k-1}. \quad (4.40)$$

Note that the non-trivial term in the acceptance probability (4.36) affected by this choice becomes

$$\alpha_s = \frac{p(\xi_k \mid y)}{p(x_{k-1} \mid y)} \cdot \frac{\gamma(x_{k-1} \mid \xi_k)}{\gamma(\xi_k \mid x_{k-1})} \quad (4.41)$$

$$= \frac{p(\xi_k \mid y)}{p(x_{k-1} \mid y)} \cdot \frac{p(x_{k-1} \mid \xi^{-i}_k, y)}{p(\xi_k \mid x^{-i}_{k-1}, y)} \quad (4.42)$$

$$= \frac{p(\xi^{-i}_k \mid \xi_k \mid y)p(\xi^{-i}_k \mid y)}{p(x_{k-1} \mid x^{-i}_{k-1}, y)p(x_{k-1}^{-i} \mid y)} \cdot \frac{p(x_{k-1}^{-i} \mid x_{k-1} \mid y)}{p(\xi^{-i}_k \mid \xi_k \mid y)} \quad (4.43)$$

$$= \frac{p(\xi^{-i}_k \mid y)}{p(x_{k-1}^{-i} \mid y)} \quad (4.44)$$

$$= 1. \quad (4.45)$$

In (4.43), the definition of conditional probability $p(A, B) = p(A \mid B)p(B)$ is used to split $p(\xi_k \mid y)$ and $p(x_{k-1}^{-i})$. Also, by design of the algorithm $\xi^{-i}_k = x^{-i}_{k-1}$, which is used in (4.43) and (4.44). The trivial result in (4.45) means that with this choice of proposal, the acceptance probability $\alpha(x_k \mid x_{k-1})$ in (4.38) may be taken as one, and hence the proposals drawn from the density $p(\xi_k \mid x^{-i}_{k-1}, y)$ are always retained. In this special case, Algorithm 4.6.1 becomes an instance of the *Gibbs sampling* algorithm, and is presented again in the Gibbs form in Algorithm 4.6.2. An application of Gibbs Sampling to multi-user detection is presented in [7].

**Algorithm 4.6.2 (Gibbs Sampling).**

1. Initialise $x_0$ at some value such that $p(x_0 \mid y) > 0$.
2. At iteration $k > 0$, randomly choose $i$ from the set of sub-blocks.
3. Draw a new $x_k$ from the density with

$$x^i_k \sim p(x^i_k \mid x^{-i}_{k-1}, y) \quad (4.46)$$

4. retain the remainder of $x$ with,

$$x^{-i}_k = x^{-i}_{k-1}. \quad (4.47)$$

5. Increment $k$ and if $k < \ell + L$ return to step 2 otherwise terminate.
4.7 MCMC Implementation Details

With the essentials of the MCMC algorithms established, attention now turns to application of these techniques to the specific problem of Multi-User Detection. This involves the specification of particular proposal distributions and a means for evaluating the required state transition probabilities.

4.7.1 MU Detection via the Gibbs Sampler

In order to utilise Gibbs sampling, a suitable partition of $x$ into sub-blocks is necessary. The choice of size of the sub-blocks involves a trade-off between the number of iterations required and the computation required per iteration. At one extreme, where each sub-block is one binary variable, it will take at most two evaluations of $p(y \mid \cdot)$ to find the necessary sampling probabilities. However, there will be considerable correlation between successive samples, as most of the sub-blocks will remain the same from one iteration to the next. At the other extreme, where there is only one sub-block, evaluation of an iteration of the algorithm requires sampling from $p(x \mid y)$, which is precisely the original problem we were trying to solve in the first place.

In this thesis, the principal investigation is of the case where the sub-blocks are a single element of $x_k$. In this situation, the Gibbs proposal (4.46) may be computed via the following Lemma, which may also be used to recognise the connection between the Gibbs sampler and other Metropolis-Hastings approaches.

**Lemma 4.7.1.** Assuming that the elements $\{x^i\}$ are statistically independent of one another, and are assigned prior probabilities $p(x^i = +1) = \rho_i$, then

$$p(\xi^i_k = +1 \mid x^{-i}_{k-1}, y) = \frac{1}{1 + \beta_i^{-1}},$$

(4.48)

where

$$\beta_i \triangleq \frac{p(y \mid x^{-i}_{k-1}, \xi^i_k = +1)}{p(y \mid x^{-i}_{k-1}, \xi^i_k = -1)} \cdot \frac{\rho_i}{1 - \rho_i}$$

(4.49)

with

$$p(y \mid x^{-i}_{k-1}, \xi^i_k) = p(v) (y - Hz), \quad z \triangleq \xi^i_k \cup x^{-i}_{k-1}.$$

(4.50)

**Proof.** By application of Bayes’ rule and the definition of conditional probability

$$p(\xi^i_k \mid x^{-i}_{k}, y) = \frac{p(y \mid \xi^i_k, x^{-i}_{k}) p(\xi^i_k \mid x^{-i}_{k})}{p(y)}.$$

(4.51)

Therefore, since the symbols sent by different users are assumed independent of one another,

$$\begin{align*}
p(\xi^i_k = +1 \mid x^{-i}_{k}, y) &= \frac{p(y \mid x^{-i}_{k}, \xi^i_k = +1) p(\xi^i_k = +1)}{p(y \mid x^{-i}_{k}, \xi^i_k = -1) p(\xi^i_k = -1)}.
\end{align*}$$

(4.52)
Consequently, with the definitions (4.49) and

\[ \alpha_i \triangleq p(\xi_k^i = +1 | x_k^{-i}, y), \]  

(4.53)
equation (4.52) indicates that

\[ \frac{\alpha_i}{1 - \alpha_i} = \beta_i \Rightarrow \alpha_i = \frac{1}{1 + \beta_i^{-1}} \]  

(4.54)
which completes the proof. \(\square\)

Combining the general Algorithm 4.6.2 with the proposal density (4.46) computation method described in (4.49) delivers the Gibbs sampling approach to multi-user detection [7, 15]. This is expressed in Algorithm 4.7.1, where the formulation has been chosen to highlight the similarities to the Metropolis algorithm.

**Algorithm 4.7.1 (Gibbs Sampler for MUD).**

1. Initialise \(x_0\) at some value such that \(p(x_0 | y) > 0\).
2. At iteration \(k\), for a randomly chosen index \(i\), compute the proposal density value

\[ \gamma(\xi_k | x_{k-1}) = p(\xi_k^i = +1 | x_k^{-i}, y) \]  

(4.55)
according to (4.48)-(4.50).
3. Draw a realisation \(\zeta \sim U[0, 1]\), where \(U[0, 1]\) represents a uniform distribution on the interval \([0, 1]\).
4. Set \(x_k^i = +1, x_k^{-i} = x_{k-1}^{-i}\) if

\[ \zeta < \gamma(\xi_k | x_{k-1}) \]  

(4.56)
otherwise set \(x_k^i = -1, x_k^{-i} = x_{k-1}^{-i}\).
5. Increment \(k\) and if \(k < \ell + L\) return to step 2 otherwise terminate.

### 4.7.2 Multi–User Detection via the Metropolis Algorithm

The Metropolis algorithm is another specialisation of Algorithm 4.6.1. However, in this case the proposal density \(\gamma\) may be chosen much more generally than is dictated in the Gibbs sampling case. This offers several potential benefits relative to the Gibbs sampler, including enhanced convergence rate, and the ability to handle more complex estimation/detection problems.

For each application of the Metropolis algorithm, a key decision is the nature of the proposal distribution \(\gamma(\cdot | x)\) to be used in (4.35). For the multi-user detection application, an effective proposal choice is to simply randomly invert one of bits in the
binary vector \( x_{k-1} \) to form the proposal \( \xi_k \). In order to formulate this proposal, it’s convenient to express it in terms of what is known as Hamming distance.

**Definition 4.7.1.** The Hamming distance \( H(x, y) \) between \( x \) and \( y \) is defined as the number of bits in \( x \) that differ from those in \( y \).

This allows a proposal probability mass function that simply flips one randomly assigned bit to be specified as,

\[
\gamma(\xi_k | x_{k-1}) = \begin{cases} 
\frac{1}{K}; & \text{if } H(\xi_k, x_{k-1}) = 1 \\
0; & \text{otherwise}
\end{cases}
\] (4.57)

where \( K \) is the number of binary elements in \( x_k \) and \( H(., .) \) represents the Hamming distance. Note that due to the symmetry of the Hamming distance operator, it follows that,

\[
\gamma(\xi_k | x_{k-1}) = \gamma(x_{k-1} | \xi_k).
\] (4.58)

This meets the condition of (4.37), and so reduces the Metropolis–Hastings algorithm to the simpler Metropolis case. The acceptance probability \( \alpha(\xi_k | x_{k-1}) \) of this new proposed state may then be calculated according to (4.36). For the case of proposing a move from \(-1\) to \(+1\) at the \( i \)-th bit,

\[
\alpha(\xi_k | x_{k-1}) = \min \left\{ 1, \frac{p(\xi_k | y) \cdot \gamma(x_{k-1} | \xi_k)}{p(x_{k-1} | y) \cdot \gamma(\xi_k | x_{k-1})} \right\}
\] (4.59)

\[
= \min \left\{ 1, \frac{p(y | x_{k-1}, \xi_k^i = +1) \cdot \gamma(x_{k-1} | \xi_k)}{p(y | x_{k-1}, \xi_k^i = -1) \cdot \gamma(\xi_k | x_{k-1})} \right\}
\] (4.60)

\[
= \min \{1, \beta_i\}.
\] (4.61)

Similarly, for the reverse case of proposing a move from \(+1\) to \(-1\) at the \( i \)-th bit,

\[
\alpha(\xi_k | x_{k-1}) = \min \left\{ 1, \frac{p(y | x_{k-1}, \xi_k^i = -1) \cdot \gamma(x_{k-1} | \xi_k)}{p(y | x_{k-1}, \xi_k^i = +1) \cdot \gamma(\xi_k | x_{k-1})} \right\}
\] (4.62)

\[
= \min \{1, \frac{1}{\beta_i}\}.
\] (4.63)

The resulting Metropolis algorithm for MUD is shown in Algorithm 4.7.2.
Algorithm 4.7.2 (Metropolis for MUD/MIMO).

1. Initialise $x_0$ at some value such that $p(x_0 \mid y) > 0$.

2. At iteration $k$, for a randomly chosen index $i$, set $\xi_k = -x_{k-1}^i \cup x_{k-1}^i$.

3. Compute the acceptance probability

$$\alpha(\xi_k \mid x_{k-1}) = \begin{cases} \min \{1, \beta_i\} & \text{for } \xi_k^i = +1 \\ \min \{1, \beta_i^{-1}\} & \text{for } \xi_k^i = -1 \end{cases}$$

where $\beta_i$ is given by (4.49)–(4.50).

4. Accept the proposed $x_k = \xi_k$ with probability $\alpha(\xi_k \mid x_{k-1})$.

5. Increment $k$ and return to step 2.

4.7.3 Metropolis and Gibbs Sampling Compared

Comparison between the Metropolis algorithm and the Gibbs Sampler implementation reveals that the procedure and calculations are very similar. In each case, one iteration of the algorithm will either leave the state the same, or invert one bit. The probability of a bit being inverted depends on the ratio $\beta$, of likelihoods between the alternate and present states. For the Gibbs Sampling method, the dependence is given by (4.48), while for the Metropolis algorithm (4.64) is used.

The difference between the two approaches is illustrated in Figure 4.5. Among these two algorithms, the Metropolis version will transition state more often than the Gibbs Sampling algorithm. In particular, note that when the present state, and proposed state are equally likely ($\beta = 1$), the Metropolis algorithm will always move to the proposed state, while the Gibbs Sampler will only do so with 50% probability. This higher probability of state transition results in less correlation in samples from the chain, and explains the slightly superior performance of the Metropolis algorithm as shown in Figure 4.6.

4.8 Convergence Analysis

The aim of the algorithms presented in this chapter is that the states $\{x_k\}$ be used as samples from $p(x \mid y)$. By computing the sample averages, estimates for the individual bit probabilities $p(x^i \mid y)$ may be obtained. Consequently, it is necessary to demonstrate that such estimates will indeed approach the correct probabilities.

The first objective of the analysis of these algorithms is to confirm the strong convergence of sample estimators such as (4.32), (4.33) for both the Gibbs and Metropolis sampling cases. This will be done by applying the Markov Chain results from Chapter 3 to the algorithms that have been presented.
4.8. Convergence Analysis

As explained in Section 4.5, Algorithms 4.7.1 and 4.7.2 are particular cases of Algorithm 4.6.1 that depend on the choice of the proposal density $\gamma(\cdot | \cdot)$. Consequently, much of the analysis is common for both of the algorithms.

4.8.1 Strong Law of Large Numbers

By construction, Algorithms 4.7.1 and 4.7.2 implement time homogeneous Markov chains, where the state transition kernel is a function of the proposal distribution $\gamma(\cdot | \cdot)$, and the acceptance probability $\alpha(\cdot | \cdot)$.

In calculating the state transition probabilities of the algorithms, there are two distinct cases to consider, according to whether or not the state vector actually changes value from one time-step to the next. For the case where $x_k \neq x_{k-1}$, the probability of observing $x_k$ given $x_{k-1}$ is simply the probability $\gamma(x_k | x_{k-1})$ of proposing $x_k$ times the probability $\alpha(x_k | x_{k-1})$ of accepting this proposal

$$\kappa(x_k | x_{k-1}) = \alpha(x_k | x_{k-1}) \cdot \gamma(x_k | x_{k-1}). \quad (4.65)$$

For the alternate case where $x_k = x_{k-1}$, the probability of observing $x_k$ is a sum accounting for the fact that this may occur either via rejection of a proposal, or via the proposal actually being $x_{k-1}$ and being accepted

$$\kappa(x_k = x_{k-1} | x_{k-1}) = \alpha(x_{k-1} | x_{k-1}) \cdot \gamma(x_{k-1} | x_{k-1})$$

$$+ \sum_z \gamma(z | x_{k-1}) [1 - \alpha(z | x_{k-1})]. \quad (4.66)$$

Equations (4.65) and (4.66) together define the Markov chain transition kernel. It is now shown that this transition kernel supports $p(x | y)$ as an invariant distribution, as defined in Section 3.2.7, as a first step to showing that the chain will indeed approach the

Figure 4.5: State transition probabilities for Gibbs and Metropolis algorithms.
desired distribution. Recall from (3.26), that a distribution \( \lambda \) is invariant for \( P \) when,

\[
\lambda P = \lambda. \tag{4.67}
\]

**Lemma 4.8.1.** Algorithms 4.7.1 and 4.7.2 implement a Markov chain with an invariant distribution equal to \( p(x \mid y) \).

**Proof.** Consider first the case where the proposed \( x_k \) is unequal to \( x_{k-1} \). Let \( p_v(x_{k-1}, x_k) \) be the probability of being in a state \( x_{k-1} \), and then transitioning to another state \( x_k \). Assuming the states do indeed have a distribution of \( p(x \mid y) \), then

\[
p_v(x_{k-1}, x_k) = p(x_{k-1} \mid y)\kappa(x_k \mid x_{k-1}) \tag{4.68}
\]

\[
= p(x_{k-1} \mid y)\gamma(x_k \mid x_{k-1})\alpha(x_k \mid x_{k-1}) \tag{4.69}
\]

\[
= p(x_{k-1} \mid y)\gamma(x_k \mid x_{k-1}) \min \left\{ 1, \frac{p(x_k \mid y)}{p(x_{k-1} \mid y)} \cdot \frac{\gamma(x_k \mid x_{k-1})}{\gamma(x_{k-1} \mid x_k)} \right\} \tag{4.70}
\]

\[
= \min \{ \gamma(x_k \mid x_{k-1})p(x_{k-1} \mid y), \gamma(x_{k-1} \mid x_k)p(x_k \mid y) \}. \tag{4.71}
\]

On the other hand, by the same reasoning, the reverse process of being in the state \( x_k \), and then transitioning to \( x_{k-1} \) is given by,

\[
p_v(x_k, x_{k-1}) = p(x_k \mid y)\kappa(x_{k-1} \mid x_k) \tag{4.72}
\]

\[
= p(x_k \mid y)\gamma(x_{k-1} \mid x_k)\alpha(x_{k-1} \mid x_k) \tag{4.73}
\]

\[
= \min \{ \gamma(x_{k-1} \mid x_k)p(x_k \mid y), \gamma(x_k \mid x_{k-1})p(x_{k-1} \mid y) \}. \tag{4.74}
\]

That is

\[
p_v(x_{k-1}, x_k) = p_v(x_k, x_{k-1}). \tag{4.75}
\]

For the alternate case, where \( x_k = x_{k-1} \), then quite trivially (4.75) again holds via the trivial equivalent identity of

\[
p_v(x_k, x_k) = p_v(x_k, x_k). \tag{4.76}
\]

Hence, by (3.33), \( p(x \mid y) \) is in detailed balance with the Markov transition kernel. Consequently, by Theorem 3.2.2, \( p(x \mid y) \) is an invariant distribution of the Markov chain. \( \square \)

This lemma leads to the principal result needed to show the convergence of estimates from the Metropolis-Hastings and Gibbs Sampling algorithms as presented in this chapter. Theorem 4.8.1 shows that under some mild conditions on the additive white Gaussian noise and prior probabilities, the MCMC algorithms 4.7.1 and 4.7.2 meet the conditions needed for the strong law of large numbers result in Theorem 3.2.7.
4.8. Convergence Analysis

Theorem 4.8.1. Suppose that the noise corruption $\nu$ in (4.11) has an underlying density which satisfies $0 < p_\nu(\cdot) < \infty$. Suppose further that the a-priori symbol probability satisfies $0 < p_i(x^i = +1) < 1$. Then Algorithms 4.7.1 and 4.7.2 both generate a realisation sequence $\{x_k\}$ for which

$$
\lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} f(x_k) = \sum_{x \in \{-1, +1\}^K} f(x) p(x | y) = \mathbb{E}\{f(x) | y\}\tag{4.77}
$$

with probability one where $f : \mathbb{R}^K \to \mathbb{R}$ is an arbitrary bounded function.

Proof. A first step in demonstrating this result is to show that the Markov chains formed by these algorithms are irreducible. Recall from Section 3.2.4 that a Markov chain is irreducible if there is a non-zero probability of eventually reaching any given state, when starting from any other state.

In order to show this condition is met, consider first the Metropolis Algorithm 4.7.2 and the case of any two states $i$ and $j$ which differ in exactly one bit position, and hence for which

$$
\gamma(x_{k+1} = j | x_k = i) = \frac{1}{K}, \tag{4.78}
$$

where $K$ is the dimension of $x$. Then according to (4.65)

$$
\kappa(x_{k+1} = j | x_k = i) = \gamma(j | i) \alpha(j | i) \tag{4.79}
$$

$$
= \frac{1}{K} \cdot \min\left\{1, \frac{p(j | y)}{p(i | y)}\right\}, \tag{4.80}
$$

where by (4.28)

$$
p(j | y) = \frac{p(x = j) p_\nu(y - H_j)}{p(x = i) p_\nu(y - H_i)}. \tag{4.81}
$$

Furthermore, by the assumptions of the Theorem all the terms in (4.81) are non-zero and bounded, so by (4.80)

$$
\kappa(x_{k+1} = j | x_k = i) > 0. \tag{4.82}
$$

Turning to the Gibbs sampling Algorithm 4.7.1, in this case when $i$ and $j$ again differ in only one bit position, the Markov transition kernel is

$$
\kappa(x_{k+1} | x_k) = \gamma(x_{k+1} | x_k), \tag{4.83}
$$

where the latter is given by (4.39), (4.48) and (4.49). The probability densities in (4.49) may be expressed in terms of the noise model as

$$
\beta = \frac{p_\nu(y - H s(z))}{p_\nu(y - H s(w))} \cdot \frac{\rho}{1 - \rho}, \tag{4.84}
$$
where
\[ z \triangleq x_{k+1}^i = +1 \cup x_{k-i}^+, \quad (4.85) \]
\[ w \triangleq x_{k+1}^i = -1 \cup x_{k-i}^- . \quad (4.86) \]

Therefore, again by the assumptions of the theorem, all terms are non-zero and bounded, and so (4.82) also holds in the Gibbs Sampling case.

Consequently, for both algorithms, and states \( i \) and \( j \) differing in an arbitrary number \( d \) of bit positions, there is a \( d \)-step path by which \( j \) can be reached from \( i \) with probability
\[
\kappa(x_{k+d} = j \mid x_k = i) \geq \kappa(x_{k+d} = j \mid x_{k+d-1} = \xi_{d-1}) \times \prod_{\ell=2}^{d-1} \kappa(x_{k+\ell} = \xi_\ell \mid x_{k+\ell-1} = \xi_{\ell-1}) \kappa(x_{k+1} = \xi_{1} \mid x_k = i) \quad (4.87)
\]
\[
> 0. \quad (4.88)
\]

Therefore, the chain implied by \( \kappa(x_{k+1} \mid x_k) \) is irreducible. By Lemma 4.8.1, this chain also has an invariant density \( p(x \mid y) \). With these properties, by Theorem 3.2.4 it is positive recurrent, and hence (4.77) follows by Theorem 3.2.7.

The utility of this Theorem is that it shows that estimates of the posterior distribution \( p(x^i \mid y) \), of the transmitted bits, may be achieved through sample histograms of the state trajectory of the MCMC algorithms. In particular, by choosing \( f \) as the Kronecker delta function \( f(x) = \delta(x - z) \) for any \( z \in \{-1, +1\}^K \)
\[
\lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} \delta(x_k - z) = \sum_{x \in \{-1, +1\}^K} \delta(x - z) p(x \mid y) = p(z \mid y) \quad (4.89)
\]
with probability one. That is, the sample histogram formed from realisations of either Algorithm 4.7.1 or 4.7.2 converges (as \( M \) increases) to the true underlying posterior density \( p(\cdot \mid y) \) and therefore provides an estimator for it.

### 4.8.2 Limiting Distribution

In related work[7], the Gibbs Sampler is used, and an alternative approach is taken for assuring convergence of the algorithm. The convergence in (4.31) is addressed by noting that
\[
\lim_{k \to \infty} \max_{x \in \{-1, +1\}^K} |p(x_k = x \mid x_0) - p(x \mid y)| = 0. \quad (4.90)
\]
Although it is not formally established in [7] that (4.90) holds, it is noted that it depends on aperiodicity and irreducibility of the underlying Markov chain.

The strong law of large numbers result in Theorem 4.8.1 does not require aperiodicity of the Markov chain, but the additional conditions needed in order to assure the aperi-
4.8. Convergence Analysis

Theodicy of the MCMC algorithms presented here are quite mild, and are made explicit in Theorem 4.8.2.

It isn’t intuitively clear that an additional condition ought to be required to establish the distributional convergence in (4.94), over that needed for the sample histogram convergence in (4.89) of Theorem 4.8.1. However, the reason for this was shown in Section 3.2.10. The example periodic Markov chain shown there in Figure 3.3 provides the expected uniform distribution from a sample histogram, but does not meet the distributional convergence requirements.

In the same way, distributional convergence is not a sufficient condition for the strong law of large numbers result in Theorem 4.8.1 that assures sample histogram convergence. The counter-example shown in Section 3.2.11 demonstrates this.

Of these two forms of convergence, it is actually strong convergence of the form (4.77) presented in Theorem 4.8.1 that is of most interest from a practical standpoint, despite the fact that in what theoretical analysis has been previously available, only distributional convergence has been considered. To argue this further, note that while distributional convergence such as (4.94) is reassuring, it does not imply the convergence (4.77) of the sample-average type estimators that have been proposed in the MCMC literature.

However, for completeness it will be shown that point-wise distributional convergence does also hold for these MCMC algorithms. The following results establish the conditions under which the Gibbs Sampler in Algorithm 4.7.1 and the Metropolis Algorithm 4.7.2 both imply such an aperiodic and irreducible chain, and hence for which (4.90) holds.

Before presenting the convergence result, it is useful to introduce a technical result that is needed to prove the aperiodicity of the Markov Chain.

**Lemma 4.8.2.** For $x_a$ and $x_b$ vectors in $\{-1, +1\}^K$, and with $\mathcal{H}(\cdot, \cdot)$ denoting Hamming distance, let $f : \mathbb{R}^K \to \mathbb{R}$ be an arbitrary function. If $f(x_a) = f(x_b)$ for all $x_a, x_b$ such that $\mathcal{H}(x_a, x_b) = 1$, then $f(x)$ is a constant for all $x \in \{-1, +1\}^K$.

**Proof.** Take $x_p$ and $x_q$ arbitrary and denote $d = \mathcal{H}(x_p, x_q)$. Then by successively inverting the differing bits it is possible to construct a sequence of $d + 1$ states,

$$\{x_0 = x_p, x_1, x_2 \ldots x_d = x_q\},$$  \hspace{1cm} (4.91)

with the property that

$$\mathcal{H}(x_r, x_{r+1}) = 1 \quad \forall r \text{ s.t } 0 \leq r < d.$$  \hspace{1cm} (4.92)

Consequently, by the assumptions of the lemma,

$$f(x_r) = f(x_{r+1}) \quad \forall 0 \leq r < d.$$  \hspace{1cm} (4.93)

However, $x_p$ and $x_q$ were arbitrary, and hence $f(x_p) = f(x_q)$ for all $x_p$ and $x_q$. \hfill \Box
Theorem 4.8.2. Under the conditions of Theorem 4.8.1, together with the further assumption that there exist at least two symbol vectors $x_1$ and $x_2$ such that $p(x_1 \mid y) \neq p(x_2 \mid y)$, Algorithms 4.7.1 and 4.7.2 both generate a realisation sequence \{x_k\} for which

$$\lim_{k \to \infty} p(x_k = x \mid x_0) = p(x \mid y)$$

(4.94)

for any $x_0 \in \{-1, +1\}^K$.

Proof. The proof uses the result of Theorem 3.2.6, and so depends on establishing that the Markov chain implied by $\kappa(x_{k+1} \mid x_k)$ is aperiodic, as described in Section 3.2.10. To do this, it is necessary to ensure that the algorithms cannot follow deterministic state patterns.

For this purpose, suppose that for all $i, j \in \{-1, +1\}^K$ that differ in only one bit position, that $p(i \mid y) = p(j \mid y)$. Then by Lemma 4.8.2,

$$p(i \mid y) = p(j \mid y) \quad \forall \ i, j \in \{-1, +1\}^K,$$

(4.95)

which contradicts the assumptions of the Theorem, and hence there must exist a pair of states $i$ and $j$ differing in only one bit position for which

$$p(i \mid y) > p(j \mid y).$$

(4.96)

Therefore, in the Metropolis algorithm case, by (4.65)

$$\alpha(x_{k+1} = j \mid x_k = i) = \min \left\{ 1, \frac{p(j \mid y)}{p(i \mid y)} \right\} < 1.$$  

(4.97)

Therefore, again according to (4.65)

$$\kappa(x_{k+1} = i \mid x_k = i) = \sum_{\ell} \gamma(\ell \mid i)[1 - \alpha(\ell \mid i)]$$

$$\geq \gamma(\ell = j \mid i)[1 - \alpha(\ell = j \mid i)]$$

$$> 0.$$

(4.98)

In the Gibbs sampling case, $\gamma(x_{k+1} = i \mid x_k = i) > 0$ from (4.39), and so it follows that

$$\kappa(x_{k+1} = i \mid x_k = i) > 0.$$  

(4.99)

By simple induction, there is a non-zero probability of staying in state $i$ for an arbitrary duration, with

$$\kappa(x_{k+n} = i \mid x_k = i) > 0 \quad \forall \ n \geq 1.$$  

(4.99)

Hence by the definition of an aperiodic state in (3.46), and Theorem 3.2.5, the Markov
4.9 Simulation Study

In order to focus the attention on the practical issues surrounding an MCMC implementation, a concrete simulation example is now presented. As a means of comparison, it is first useful to find a theoretical optimum performance, against which these algorithms may be judged.

4.9.1 Single-User Bound

There is no closed form solution describing the performance of an optimal detector for multi-user CDMA. However it is straightforward to compute the optimal error probability for a single-user CDMA channel. As the existence of additional users cannot decrease the bit error rate for an optimal detector, this acts as lower bound on the error rate for detection in the multi-user case.

The single-user version of the CDMA signal model in (4.3), for one user $n$, may be represented as,

$$y_n(t) = A_n x_n(t) s_n(t) + \sigma \nu(t), \quad 0 \leq t \leq T,$$

(4.100)

where $A_n$ describes the received amplitude, $x_n \in \{-1, +1\}$ is the transmitted bit, and $s_n(t)$ is the signature sequence normalised to unit energy. The noise introduced with $\sigma \nu(t)$ has a single-sided power spectral density $N_0$ of

$$N_0 = 2\sigma^2.$$  

(4.101)

The optimal detector for this signal model follows that of a Binary Phase Shift Keyed (BPSK) signal[48, §3.2], and has a bit error rate of,

$$p_e = Q \left( \frac{A_n}{\sigma} \right),$$  

(4.102)

where the $Q$-function is the complementary cumulative distribution function for the unit
Gaussian random variable,
\[ Q(w) \triangleq \int_{w}^{\infty} \frac{1}{\sqrt{2\pi}} e^{-t^2/2} \, dt. \]  
(4.103)

Given that \( s_n(t) \) has unit energy of across a symbol, the bit energy is
\[ E_b = A_n^2. \]  
(4.104)

This allows the computation of the optimal bit error rate as a function of the \( E_b/N_0 \) ratio. \( E_b/N_0 \) represents the ratio of energy per bit to the noise power spectral density.
\[ p_e = Q\left( \sqrt{\frac{2E_b}{N_0}} \right) \]  
(4.105)

This function is included as the single user bound in the bit-error ratio plots.

### 4.9.2 Algorithm Performance

Considered here is a non-trivial scenario of multi-user detection with \( K = 96 \) users with spreading factor \( N = 128 \). An additive white Gaussian noise channel is used and the choice \( A_k = 1, k = 1, \ldots, K \) taken for the transmit powers. The signature waveforms are of direct sequence form with \( s_n(t) \in \{\pm 1\} \), and chosen randomly and independently.

The noise is introduced to the channel to a specified energy per bit to noise power spectral density ratio \( (E_b/N_0) \), with \( E_b \) and \( N_0 \) here defined respectively in (4.104) and (4.101). At each value of \( E_b/N_0 \), Algorithms 4.7.1 and 4.7.2 were both run for \( M = 5000 \) iterations, with detection achieved by thresholding of the resulting posterior probability estimates. The bit error rate (BER) was then evaluated by repeating the simulation with different noise and spreading sequence realisations until a total of 5000 error events were observed.

The results are presented in Figure 4.6. Also shown there is the performance, under the same conditions, of the linear minimum mean-squared error (MMSE) detector \[48\], an iterative parallel decision-feedback detector (P-DFD) and a 2-stage iterative successive DFD (IS-DFD) \[54\]. These detectors are examples of other sub-optimal algorithms that have been proposed for systems such as this where the true MAP detector is infeasible to calculate. The single-user performance is also shown, which presents a lower bound on the possible performance that may be achieved by a CDMA system.

For lower SNR’s Algorithms 4.7.1 and 4.7.2 outperform the MMSE, P-DFD and IS-DFD receivers by approximately 1dB. Also of note is that the Metropolis based Algorithm 4.7.2 provides a modest performance advantage relative to the Gibbs sampling based Algorithm 4.7.1.

However, it is also clear that the efficacy of both Algorithms 4.7.1 and 4.7.2 degrade as the SNR increases. This ‘hold-up’ phenomenon has been previously observed in \[7\]
and a solution suitable for the Gibb’s sampling case of Algorithm 4.7.1 has been recently provided in [15].

In order to better understand the practical convergence properties of the algorithms, it’s necessary to see how they perform as a function of the number of iterations. Figure 4.7 shows the bit error rate resulting from the Metropolis algorithm for a varying number of iterations of the Markov chain. For the case of the SNR=4dB, the algorithm converges quickly to approach the minimum value within $10^4$ iterations. However, in the case of SNR=8dB, even at $10^5$ iterations, convergence has not been reached.

Despite the long convergence times on the 8dB plot, Comparison with Figure 4.6 shows that it does nonetheless eventually reach an error rate very close to the single user bound, and consequently, optimality. Attention will now turn to the question of why the higher SNR leads to such long convergence times for the MCMC algorithms.

4.10 Hold-Up Effects

In order to understand the problems experienced by MCMC algorithms at higher SNR, it is necessary to consider how SNR affects the Markov state trajectory within the algorithm. The behaviour of the state trajectory is largely determined by the proposal and acceptance characteristics, and so analysis can be broken down into how each of these components change over the range of SNR.

Considering the case of the Metropolis algorithm, the proposal stays the same for all SNR, and so initial attention is on how the acceptance ratio behaviour changes.
From (4.38), the acceptance probability in the Metropolis case is simply a likelihood ratio,

$$\alpha(\xi_k | x_{k-1}) = \min \left\{ 1, \frac{p(\xi_k | y)}{p(x_{k-1} | y)} \right\}.$$  \hspace{1cm} (4.106)

For the Gaussian noise case under consideration, the likelihood is expressed in (4.29) as,

$$p(x | y) = \frac{\text{const}}{\sqrt{2\pi}\sigma^2} \exp \left( -\frac{1}{2\sigma^2} \| \Gamma^{-1/2}(y - Hx) \|^2 \right).$$  \hspace{1cm} (4.107)

The probability ratio component in (4.106) then becomes,

$$\frac{p(\xi_k | y)}{p(x_{k-1} | y)} = \exp \left[ -\frac{1}{2\sigma^2} \left( \| \Gamma^{-1/2}(y - H\xi_k) \|^2 - \| \Gamma^{-1/2}(y - Hx_{k-1}) \|^2 \right) \right].$$  \hspace{1cm} (4.108)

While the change in SNR will clearly affect the received signal $y$, the dominant effect on the probability ratio is via the leading $\frac{1}{2\sigma^2}$ term. As the SNR increases, the value of $\sigma$ decreases. For the case of interest, where the ratio is less than one, a decreasing $\sigma$ will reduce the probability ratio closer to zero.

The consequence is that with a high SNR, the distribution of the acceptance ratio will bifurcate to regions close to one, or close to zero. In terms of algorithm performance, it means that the Metropolis algorithm will quickly find a high likelihood state, but then stay stationary a long time before moving away from any local maxima. When samples $x_k$ are at or near a local maxima, the acceptance probability of moving away from them will be very small, and the hence they will ‘lock’ there.

This raises the question of whether this behaviour is a result of a flaw in the MCMC algorithms that can be corrected. However, by considering what the algorithm must
achieve, it is in fact a natural consequence of the true underlying posterior $p(x \mid y)$ becoming more sharply peaked, both at the global and local maxima, and with very low probabilities away from these peaks. For a sampling algorithm to accurately depict the nature of those peaks, it is inherently necessary to draw a very large number of samples around those peaks.

### 4.10.1 High SNR Solution

The remedy proposed in this thesis is instead of altering the algorithm, to alter the shape of the posterior $p(x \mid y)$ to be sampled from. The aim will be to create a shape more amenable to the use of the Metropolis algorithm, and then develop a mechanism to reverse the bias that this process inherently introduces into the results. This will involve a form of importance sampling, as described in Section 2.3.4.

The method proposed involves replacing $p(x \mid y)$ in (4.36) with a modified function $p_\eta(x \mid y)$ which is defined as, for some $\eta \in (0, 1)$,

$$
p_\eta(x \mid y) \equiv \frac{1}{\zeta} [p(x \mid y)]^\eta,
$$

where $\zeta$ is the normalising constant,

$$
\zeta \equiv \sum_{x \in \{-1, 1\}^K} [p(x \mid y)]^\eta. \tag{4.110}
$$

Since $\eta < 1$, at high SNR any sharp peaking in $p(x \mid y)$ will be reduced under the transformation to $p_\eta(x \mid y)$. This will result in higher acceptance probability $\alpha(\xi_k \mid x_{k-1})$ in regions between maxima of $p(x \mid y)$, and hence greater ‘movement’ in the Markov chain.

The effect of this can be seen by considering the new probability ratio in the acceptance probability previously described in (4.108),

$$
\frac{p_\eta(\xi_k \mid y)}{p_\eta(x_{k-1} \mid y)} = \exp \left[ -\frac{\eta}{2\sigma^2} \left( \|R^{-1/2}(y - H\xi_k)\|^2 - \|R^{-1/2}(y - Hx_{k-1})\|^2 \right) \right]. \tag{4.111}
$$

The $\eta$ parameter may now easily be tuned to compensate for the effects of a low value of $\sigma$.

### 4.10.2 Convergence of Modified Algorithm

In order to be confident of the behaviour of the modified algorithm, it is necessary to ensure that the convergence results of Section 4.8 still hold.

The proof of Lemma 4.7.1 establishes that to incorporate this transformation in Algorithms 4.7.1 or 4.7.2 simply requires the transformation $p_\nu(\cdot) \rightarrow [p_\nu(\cdot)]^\eta$ in (4.50). Note that in this latter case a normalising constant akin to $\zeta$ in (4.109) is unnecessary since (4.50) is only used to compute the ratio $\beta_i$ appearing in (4.49).
For the same reason, it is not necessary to compute $\zeta$ when implementing the transformed versions of Algorithms 4.7.1 or 4.7.2, although as explained below it is necessary if soft posterior density outputs are required.

Finally, examination of the Theorems 4.8.1, 4.8.2 establishes that under the transformation (4.109) the convergence properties of Algorithms 4.7.1 and 4.7.2 are unchanged save for the replacement of $p(x \mid y)$ with the transformed $p_\eta(x \mid y)$ in (4.77) and (4.94). The requirement that $\eta > 0$ ensures that the conditions on $p_\nu$ for Theorem 4.8.1 are met. Consequently, the estimated sampling posterior will converge to $p_\eta(x \mid y)$.

4.10.3 Maintaining Soft Outputs

Naturally, it is of little value to simply substitute $p_\eta$ for $p$ for computational efficacy, if it cannot produce an estimate for the desired $p(x^i \mid y)$. Without further modification, the algorithm will instead produce estimates for $p_\eta(x^i \mid y)$.

If the posteriors are only used to inform hard decisions, then we may utilise the property that the transformation (4.109) is monotonic for any $\eta$. Consequently,

$$\arg\max_x p_\eta(x \mid y) = \arg\max_x p(x \mid y),$$

and hence it is equivalent to work with either $p(x \mid y)$ or $p_\eta(x \mid y)$ in this case.

However, if soft outputs are required, then it is necessary to adjust for the scaling of $p_\eta$. Denote by $\hat{p}_\eta(x \mid y)$, the posterior estimated by either of Algorithms 4.7.1 or 4.7.2. Then the required estimate $\hat{p}(x \mid y)$ may be found by inverting the transformation (4.109) according to

$$\hat{p}(x \mid y) = [\zeta \hat{p}_\eta(x \mid y)]^{1/\eta}.$$ (4.113)

In order to achieve soft outputs for the marginal probability mass function $p(x^i \mid y)$, it is necessary to perform importance sampling (Section 2.3.4), on the states of the Markov Chain. Importance Sampling provides a generic approach to performing inference on one distribution using samples from a related distribution.

By applying the importance sampling result in (2.27) to the MCMC result of Theorem 4.8.1,

$$\mathbb{E}\{f(x) \mid y\} = \lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} \frac{p(x_k \mid y)}{p_\eta(x_k \mid y)} f(x_k)$$

$$= \lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} \zeta p(x_k \mid y)^{(1-\eta)} f(x_k),$$ (4.115)

where $x_k$ are sampled from the Markov chain with limiting density of $p_\eta(x_k \mid y)$. When estimating a posterior by building a histogram, the correction is applied as a weight to each sample prior to histogram summation. The additional computation is minimal, as $p(x_k \mid y)$ is already known as part of the Metropolis algorithm acceptance probability
calculation. In the case where a marginal probability distribution is to be determined, the constant $\zeta$ need not be calculated, as the computed distribution may instead be normalised.

4.10.4 Performance of the Modified Algorithm

To illustrate the effect of employing these ideas, Figure 4.8 shows the resultant BER performance of the ensuing modified Algorithms 4.7.1 and 4.7.2 under the same experimental conditions as described in the previous section. Note in particular the removal of the hold-up effect illustrated in Figure 4.6 to such an extent that the performance now approaches the single user bound.

![Figure 4.8: BER vs SNR for Modified Metropolis and Gibbs samplers with $\eta = \sigma^2 - 4dB$.](image)

In this example, a scaling exponent of $\eta$ was chosen such that $\sigma^2/\eta = 4dB$. However an important question remains of what the optimum value for $\eta$ is, and how it can be calculated in general. For this case example, a study of BER performance versus $\eta$ for a range of $E_b/N_0$ values is illustrated in Figure 4.9. In this plot, for 5000 iterations of the algorithm, error performance is shown as a function of the choice of $\eta$.

First, note that the choice of $\eta$ is not critical, since the BER performance is relatively insensitive to it. It may vary over a range of around 3dB while still maintaining close to optimal performance. Second, note that in the Gaussian case profiled in this simulation,

$$[p_{\nu}(x)]^{\eta} \propto \exp \left(-\frac{\eta}{2\sigma^2}\|x\|^2\right).$$

Therefore, in this case the transformation (4.109) simply amounts to a scaling of the
estimated channel noise variance. This explains the choice for the $x$-axis in Figure 4.9. Regardless of the actual $E_b/N_0$ of the received signal, the optimum value of the combined quantity $\sigma^2/\eta$ used by the algorithm remains largely constant. This is important, as it shows that $\sigma^2/\eta$ may be designed once for the system, rather than have to adapt for differing operational conditions.

One way of looking at this result is to consider the case of hard-decision detection in the basic algorithm without the $\eta$ scaling. In this situation, and for this numerical example, the best approach is to always use a $\hat{\sigma}$ estimate in detection that corresponds to an $E_b/N_0$ of 4dB, regardless of what the true noise in the received signal actually is. In the hard decision case, this shows that Algorithms 4.7.1 and 4.7.2 are robust to the quality of a prior estimate of the channel noise variance.

The consistency of the $\sigma^2/\eta$ ratio as a measure of performance shows that the MCMC algorithm itself does not vary greatly as a function of the received signal noise. The optimal $\sigma^2/\eta$ factor may then be determined independently of the operational $\sigma^2$. With a fixed $\sigma^2/\eta$, the only actual need for knowledge of $E_b/N_0$ is so that $\sigma$ is known and hence $\eta$ may be computed to be used in the scaling of the soft-output computation.

Figure 4.10 shows algorithm convergence for SNR of 4dB, 6dB and 8dB, where $\eta$ has been chosen in each case to preserve the $\sigma^2/\eta$ ratio that would normally be achieved at an SNR of 4dB. For each of these cases, the convergence behaviour is similar, which may be contrast to the SNR=8dB, $\eta = 1.00$ plot, which has much slower convergence behaviour.
4.11. Convergence Rate Analysis

Convergence for varying SNR, 96 Users, Gain=128

\[ \text{Bit Error Rate} \]

\[ \text{Algorithm Iterations} \]

\[ \text{SNR=4dB, } \eta=1 \]
\[ \text{SNR=6dB, } \eta=0.63 \]
\[ \text{SNR=8dB, } \eta=0.40 \]
\[ \text{SNR=8dB, } \eta=1.00 \]

Figure 4.10: BER vs Algorithm Iterations

4.10.5 Soft-Output Performance

To this point, the simulation results have considered only the hard decision performance of the MCMC detectors proposed here. However, one of the main attractions of MCMC detectors is the ability to provide soft posterior probability outputs which, for example, may be fed to soft-input channel decoders including turbo and low-density parity-check decoders[25]. To profile the accuracy of the soft outputs, the estimated versus true posterior \( p(\{x_i\} = 1 | y) \) for the simulation setup previously described and a SNR of 8dB is shown in Figure 4.11. The straight line relationship shown there verifies the accuracy of the estimated posteriors.

The x-axis of this histogram represents a partition of the decoded symbols according to the estimated likelihood of \( p(\{x_i\} = 1) \). So the right side of the histogram represent the decoded bits ascribed by the detector as having a high probability of being a one. The y-axis represents the proportion of such bits that were transmitted as a one.

Note that there is a bias on the histogram bins at each end, where they approach zero and one respectively. This is due to the very high proportion of bits in those bins which are correctly assigned probabilities very close to zero or one on account of the high SNR. They dominate the behaviour of the respective bins that contain them, and so slightly skew the values of the end-most histogram bins.

4.11 Convergence Rate Analysis

Having now established the convergence of Algorithms 4.7.1 and 4.7.2, this section turns to the practically relevant question of rate of convergence. While it is known that with sufficient iterations, the estimates will eventually converge to the target posterior probabilities, the theory to this point gives no indication to how quickly this will happen. For a
practical situation, we are interested in how many iterations are necessary for a required level of accuracy. This is a very challenging problem, and for a typical Metropolis–Hastings algorithm only quite general bounds are possible [41]. Consequently, in the interests of most clearly exposing the key factors affecting convergence rate, the approach taken here is to instead study a hypothetical, more tractable Markov chain, which is nevertheless closely related to Algorithms 4.7.1 and 4.7.2. This new Markov chain is still a realisation of the base Metropolis–Hastings Algorithm 4.6.1, but uses a different proposal density $\gamma$, which is more amenable to analysis. The key feature is that the new proposal density $\gamma$ is constrained to be independent of previous realisations from the chain.

This independence of the proposal density will permit explicit analysis of convergence rate. As it is being considered as an analysis tool, there is no constraint required to actually make it feasible to implement this new proposal in a practical algorithm. There may instead be a considerable amount of computation required. However, in order for this rate to be as close as possible to the practical chains implied by Algorithms 4.7.1 and 4.7.2, in the following sections the probability of a given candidate $\xi_k$ in these algorithms in steady state will be matched to that of the hypothetical (analysable) chain via a special choice of the independent proposal.

### 4.11.1 Metropolis–Hastings with Independent Proposal

To study the numerical properties of the independence chain, the state transition probabilities are represented as a $2^K \times 2^K$ transition probability matrix $P$. As described in Section 3.2, this is given as

$$[P]_{\ell,j} = \kappa(x_{k+1} = j \mid x_k = \ell).$$

(4.117)
The associated vector of conditional state probabilities $\pi$ for the limiting distribution is defined as

$$\pi = [p(x = 1 \mid y), p(x = 2 \mid y), \ldots, p(x = 2^K \mid y)].$$

(4.118)

The time evolution of a Markov Chain was considered in Section 3.2.2, and (3.13) shows that for an initial state distribution $\mu$, 

$$\pi = \lim_{k \to \infty} \mu P^k.$$  

(4.119)

The convergence properties of the chain are then determined by the behaviour of $P^k$ as $k$ increases. This motivates considering the eigendecomposition of $P$, where if there is a full set of linearly independent eigenvectors, then $P$ may be factored into,

$$P = QAQ^{-1},$$

(4.120)

where the columns of $Q$ are the eigenvectors of $P$, and $\Lambda$ is a diagonal matrix of the corresponding eigenvalues. Then,

$$P^k = QA^kQ^{-1}.$$  

(4.121)

Clearly then, the speed of convergence is closely linked to the size of the eigenvalues, and in particular that of the eigenvalues closest to unity. Attention will now turn to estimating these eigenvalues.

For the case of a Metropolis–Hastings algorithm with an independent proposal, denote the probability of proposing at any iteration a next state of $i$ to be $\gamma_i$. By total probability, 

$$\sum_{i=1}^{2^K} \gamma_i = 1.$$  

(4.122)

When representing the Markov kernel as a transition matrix, each state is ascribed an index into that matrix and into any corresponding density vector. However, the choice of state index assignment may be made arbitrarily. For the purposes of this analysis, the state indices are allocated in a specific order to ensure a regular structure in the transition matrix. The state assignment is made so that the elements of $\pi$ are sorted according to

$$\frac{\pi_1}{\gamma_1} \geq \frac{\pi_2}{\gamma_2} \geq \ldots \geq \frac{\pi_{2^K}}{\gamma_{2^K}}.$$  

(4.123)

Noting that by Lemma 4.8.1, (4.118) is an invariant distribution of $P$, so by (4.67)

$$\pi P = \pi,$$  

(4.124)
and hence $\pi$ is a left eigenvector of $P$ with eigenvalue $\lambda_0 = 1$. The remaining eigenvalues, denoted $\lambda_1, \ldots, \lambda_{2^K-1}$ are of significant interest, because they are the ones that will determine the rate of convergence to this stationary distribution $\pi$. They may be directly formulated as follows.

Theorem 4.11.1. The transition matrix $P$ implied by Algorithm 4.6.1 with proposal $\gamma$ constrained to be independent of previous realisations has eigenvalues in order of decreasing size given by

$$\lambda_0 = 1, \quad \lambda_j = \sum_{i=j}^{2^K} \left( \frac{\gamma_i - \pi_i \gamma_j}{\pi_j} \right), \quad 1 \leq j < 2^K. \quad (4.125)$$

Proof. We apply a method due to Liu [32]. Due to the independence of the proposal distribution, it is possible to determine the form of the transition matrix $P$, as defined in (4.117). The resulting transition matrix is,

$$P = \begin{bmatrix}
\gamma_1 + \lambda_1 & \gamma_1 \pi_2 / \pi_1 & \ldots & \gamma_1 \pi_{2^K-1} / \pi_1 & \gamma_1 \pi_{2^K} / \pi_1 \\
\gamma_1 & \gamma_2 + \lambda_2 & \ldots & \gamma_2 \pi_{2^K-1} / \pi_2 & \gamma_2 \pi_{2^K} / \pi_2 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
\gamma_1 & \gamma_2 & \ldots & \gamma_{2^K-1} + \lambda_{2^K-1} & \gamma_{2^K-1} \pi_{2^K-1} / \pi_{2^K-1} \\
\gamma_1 & \gamma_2 & \ldots & \gamma_{2^K-1} & \gamma_{2^K}
\end{bmatrix} \quad (4.126)$$

with $\lambda_j$ given by (4.125). To establish this, notice that for the subdiagonal terms $\{[P]_{\ell,j} : \ell > j\}$, according to Algorithm 4.6.1 the acceptance probability is

$$\alpha(j, \ell) = \min \left\{ 1, \frac{\pi_j \cdot \gamma_\ell}{\pi_\ell \cdot \gamma_j} \right\} \quad (4.127)$$

$$= \min \left\{ 1, \frac{\pi_j}{\gamma_j} \cdot \frac{\gamma_\ell}{\pi_\ell} \right\} \quad (4.128)$$

$$= 1 \quad (4.129)$$

where (4.129) follows according to the ordering (4.123). Hence for $\ell > j$ the proposal will always be accepted. These terms below the diagonal correspond to transitions to a more likely state than the present state. According to Algorithm 4.7.2, such a transition will always be accepted if it is proposed. Thus, the sub-diagonal transition probabilities are equal to the associated proposal probability $\gamma_j$.

Similarly, for the superdiagonal terms where $\ell < j$, the ordering (4.123) implies an acceptance probability of less than one. Consequently, the transition probability is the acceptance probability times proposal probability of

$$\alpha(j, \ell) = \left( \frac{\pi_j}{\pi_\ell} \cdot \frac{\gamma_\ell}{\gamma_j} \right) \gamma_j = \frac{\pi_j}{\pi_\ell} \gamma_\ell. \quad (4.130)$$
4.11. Convergence Rate Analysis

The remaining diagonal terms may be determined by known properties of the matrix $\mathbf{P}$. Since $\mathbf{P}$ is a stochastic matrix, its row sums are unity and hence the diagonal terms are given by

$$
[P]_{j,j} = 1 - \sum_{i=1}^{j-1} \gamma_i - \sum_{i=j+1}^{2^K} \frac{\pi_i}{\pi_j} \gamma_j
$$

(4.131)

$$
= \sum_{i=j}^{2^K} \gamma_i - \left( \sum_{i=j}^{2^K} \frac{\pi_i}{\pi_j} \gamma_j - \gamma_j \right)
$$

(4.132)

$$
= \sum_{i=j}^{2^K} \left( \gamma_j - \frac{\pi_i}{\pi_j} \gamma_j + \gamma_j \right)
$$

(4.133)

$$
= \gamma_j + \lambda_j,
$$

(4.134)

where the fact that $\sum_i \gamma_i = 1$ has been exploited for the first term in (4.132).

With this formulation of $\mathbf{P}$ now established, it is claimed that its eigenvalues are given by (4.125) with corresponding right eigenvectors given by

$$
v_k = \begin{cases} 
[1, 1, \cdots, 1]^T; & k = 0 \\
0, 0, 0, \ldots, 0, -\sum_{\ell=k+1}^{2^K-1} \frac{\pi_\ell}{\pi_k}, 1, 1, \ldots, 1 \end{cases}^T; & 0 < k < 2^K,
$$

(4.135)

where there are $k-1$ zeros in (4.135). These eigenvalue/vector pairs are now verified by demonstrating

$$
\mathbf{P}v_k = \lambda_k v_k.
$$

(4.136)

To confirm this relationship, let $\mathbf{V}$ be a matrix whose columns are $v_k$, and then set $\mathbf{U} = \mathbf{PV}$. Then for $j < k$

$$
[U]_{j,k} = \gamma_j \frac{\pi_k}{\pi_j} \left( - \sum_{\ell=k+1}^{2^K} \frac{\pi_\ell}{\pi_k} \right) + \sum_{\ell=k+1}^{2^K} \gamma_j \frac{\pi_\ell}{\pi_j}
$$

(4.137)

$$
= 0.
$$

(4.138)

For $j = k$

$$
[U]_{j,k} = (\gamma_k + \lambda_k) \left( - \sum_{\ell=k+1}^{2^K} \frac{\pi_\ell}{\pi_k} \right) + \sum_{\ell=k+1}^{2^K} \gamma_k \frac{\pi_\ell}{\pi_k}
$$

(4.139)

$$
= -\lambda_k \sum_{\ell=k+1}^{2^K-1} \frac{\pi_\ell}{\pi_k}.
$$

(4.140)
For $j > k$

$$[U]_{j,k} = \gamma_k \left( -\sum_{\ell=k+1}^{2k} \frac{\pi_\ell}{\pi_k} \right) + \left[ \sum_{\ell=k+1}^{j-1} \gamma_\ell \right] + [\gamma_j + \lambda_j] + \left[ \sum_{\ell=j+1}^{2k} \gamma_k \frac{\pi_\ell}{\pi_k} \right]$$

(4.141)

$$= - \left( \sum_{\ell=1}^{k-1} \gamma_\ell \right) + \gamma_k + \lambda_k + \left[ \sum_{\ell=1}^{j-1} \gamma_\ell \right] + [\gamma_j + \lambda_j] + \left[ \sum_{\ell=j+1}^{2k} \gamma_k \frac{\pi_\ell}{\pi_k} \right]$$

(4.142)

$$= -1 + \lambda_k + \left( \sum_{\ell=1}^{j-1} \gamma_\ell \right) + [\gamma_j + \lambda_j] + \left[ \sum_{\ell=j+1}^{2k} \gamma_k \frac{\pi_\ell}{\pi_k} \right]$$

(4.143)

$$= \lambda_k$$

(4.144)

where the simplification results from observing that the first line of (4.142) and last line of (4.144) are the sums of the $k$’th and $j$’th rows of $P$ and hence equal to one. Finally, this fact of all row sums of $P$ being equal to 1 establishes that $v_0$ given by (4.135) is a right eigenvector with eigenvalue $\lambda_0 = 1$.

This exact formulation of the eigenvalues of $P$ in (4.117) makes possible a simple and explicit convergence analysis. The following theorem uses the eigenvalue result to place a rate of convergence bound on reaching the limiting distribution.

**Theorem 4.11.2.** For any initialisation $x_0$, and under the assumptions of Theorem 4.8.2, the distribution $p(x_k|x_0)$ of a realisation $x_k$ from Algorithm 4.7.2 with $\gamma$ modified to be independent of past realisations satisfies

$$|p(x_k = x \mid x_0) - p(x \mid y)| < R\lambda^k$$

(4.145)

where $R < \infty$ is a constant and

$$\lambda = 1 - \min_x \tau(x),$$

(4.146)

$$\tau(x) \triangleq \frac{\gamma(x)}{p(x \mid y)}.$$  

(4.147)

**Proof.** Denote the initial Markov state distribution at $k = 0$ as the vector

$$\mu_0 = \left[ p(x_0 = 1), p(x_0 = 2), \ldots, p(x_0 = 2^K) \right].$$

(4.148)

Note that from (3.12), the distribution $\mu_k$ of the Markov chain defined by $\kappa(\cdot \mid \cdot), \mu_0$ at
4.11. Convergence Rate Analysis

Time \( k > 0 \) is given by successive multiplication against the \( P \) matrix with,

\[
\mu_k = \mu_0 P^k. \tag{4.149}
\]

If \( P \) has distinct eigenvalues \( \{\lambda_\ell\} \), with associated left eigenvectors \( \{w_\ell\} \), it’s possible to define \( \{\alpha_\ell\} \) via the decomposition of the initial state distribution,

\[
\mu_0 = \sum_{\ell=0}^{2^K-1} \alpha_\ell w_\ell. \tag{4.150}
\]

By applying successive multiplications, and using the property that \( w_\ell P = w_\ell \lambda_\ell \), then

\[
\mu_k = \mu_0 P^k = \left( \sum_{\ell=0}^{2^K-1} \alpha_\ell \lambda_\ell \right) P^{k-1} = \left( \sum_{\ell=0}^{2^K-1} \alpha_\ell \lambda_\ell^m \right) P^{k-m} = \sum_{\ell=0}^{2^K-1} \alpha_\ell \lambda_\ell^k w_\ell. \tag{4.153}
\]

Theorem 4.11.1 establishes that \( \lambda_0 = 1 \) is a maximal eigenvalue with all others satisfying \( |\lambda_k| < 1 \) for \( 0 < k < 2^K \). Furthermore, via (4.124) \( w_0 = \pi \) and hence

\[
\mu_k = \pi \alpha_0 + \varepsilon_k, \quad \varepsilon_k \triangleq \sum_{\ell=1}^{2^K-1} \alpha_\ell \lambda_\ell^k w_\ell. \tag{4.154}
\]

Considering each \( i \)-th element \( \varepsilon_k \) and \( w_k \) as \( \varepsilon^i_k \) and \( w^i_k \) respectively, define the constants with respect to \( k \),

\[
R_i \triangleq \sum_{\ell=1}^{2^K-1} |\alpha_\ell||w^i_\ell|. \tag{4.155}
\]

An arbitrary \( i \)'th element of \( \varepsilon_k \) then satisfies

\[
|\varepsilon^i_k| < R_i \cdot \max_{1 \leq \ell < 2^K} |\lambda_\ell|^k. \tag{4.156}
\]

However, by Theorem 4.11.1

\[
\max_{1 \leq \ell < 2^K} |\lambda_\ell| = \lambda_1 = 1 - \frac{\gamma_1}{\pi_1} < 1 \tag{4.157}
\]
so that \( \lim_{k \to \infty} |\varepsilon_k| = 0 \) with
\[
|\varepsilon_k| < R\lambda^k. \tag{4.158}
\]

Hence also by Theorem 4.8.2, \( \alpha_0 = 1 \), and (4.145) follows.

This result is important as it puts a numerical bound on the rate of convergence of a Metropolis Algorithm based on the proposal and target densities. So in the situation where the Metropolis Algorithm presented in Algorithm 4.7.2 is modified to have independent proposal, the convergence rate of the sample density \( p(x_k | x_0) \) to the posterior \( p(x | y) \) is exponential, and governed by the quantity,
\[
\min_x \tau(x) = \min_x \frac{\gamma(x)}{p(x | y)}. \tag{4.159}
\]

This may be interpreted as being determined by the maximum mismatch between the desired posterior \( p(x | y) \) and the proposal \( \gamma \).

In particular, at high SNR when the MAP estimate \( \hat{x} \) implies \( p(\hat{x} | y) \approx 1 \), then for uniform proposal \( \gamma(x) = 2^{-K} \)
\[
\min_x \tau(x) \leq \tau(\hat{x}) \approx \frac{1}{2^K}. \tag{4.160}
\]

For large \( K \) (e.g., number of users in a CDMA system), this will imply \( \lambda \approx 1 \) and hence slow convergence, which is consistent with the ‘hold up’ observed in [7, 15].

### 4.11.2 Independent Proposal Choice

While the convergence rate result from the independence chain provides a good measure of the speed of convergence, it is a significant limitation that it only apply to independent proposals, as practical algorithms often generate proposals based on the current Markov state. In fact, considering (4.146) it is beneficial to have the proposal \( \gamma(x) \) similar in shape to \( p(x | y) \), and an effective way of achieving that is to use the Markov state, which in turn is influenced by \( p(x | y) \).

Using this reasoning, one may even consider using a proposal \( \xi \sim \gamma(x) \) where \( \xi_k = x_{k-1} \), with the rationale that if the algorithm is operating correctly then \( x_{k-1} \sim p(x | y) \). However, clearly this will not work, as not only are the irreducibility conditions not met for the Markov Chain convergence proofs, but the chain will simply never move state. The random-walk and random bit-flipping approaches presented in this thesis may be considered in a similar light. They are an attempt to use the present state \( x \sim p(x | y) \) to generate a proposal that is close to \( p(x | y) \), while still promoting chain movement. It’s natural to then consider just what the sample density \( \gamma(\xi) \) is of the proposed states within these Metropolis algorithms.

Consequently, we now further investigate the choice of an independent proposal den-
4.11. Convergence Rate Analysis

sity $\gamma$ to best match the density of the algorithm already presented. It will be selected so that the probability $\gamma(\xi)$ of a given proposed Markov chain realisation $\xi$ is identical to that arising from the original (non-independent proposal) Metropolis Algorithm 4.7.2 after it has converged. The underlying argument is that via this construction, the convergence properties of the independent chain can be expected to be informative as possible for those of Algorithm 4.7.2.

In this case, the convergence rate can again be judged by the results of Theorem 4.11.2 in combination with the following Lemma quantifying $\tau(x)$.

**Lemma 4.11.1.** Suppose that the independent proposal $\gamma(\xi)$ is chosen to be, as a function of $\xi$, identical to the dependent proposal $\gamma(\xi|x)$ employed in Algorithm 4.7.2 after convergence has occurred. Then the quantity $\tau(x)$ defined in equation (4.147) of Theorem 4.11.2 is given by

\[
\tau(x) = \frac{q_j(x|y)}{p(x|y)},
\]

where

\[
q_j(x|y) \triangleq \sum_{\sigma_j} \frac{1}{K} p(\sigma_j|y), \quad \text{with} \quad \sigma_j \triangleq \{\xi : H(x, \xi) = j\}
\]

and $H(\cdot, \cdot)$ is the Hamming distance (Definition 4.7.1).

**Proof.** Note that according to Theorem 4.8.2, the unmodified Algorithm 4.7.2 converges so that $x_k \sim p(x|y)$, in which case the probability of a candidate realisation $\xi$ being proposed is

\[
\sum_x \gamma(\xi|x)p(x|y).
\]

Furthermore, Algorithm 4.7.2 uses a (non-independent) proposal which involves randomly inverting one bit of $x$. This may be expressed as

\[
\gamma(\xi|x) = \begin{cases} 
K^{-1} & \text{if } H(\xi, x) = 1 \\
0 & \text{otherwise},
\end{cases}
\]

Therefore, by combining (4.163) and (4.164)

\[
\gamma(\xi) = \sum_{x: H(\xi, x) = 1} \frac{1}{K} p(x|y)
\]

is an independent proposal that, after burn in, is identically distributed to the non-independent proposal used in Algorithm 4.7.2. Using this in (4.146) completes the proof.

To understand this result, note that $\{\xi : H(\xi, x) = j\}$ is the set of elements differing
from $x$ in exactly $j$ bit positions. Therefore, if the initial distribution for the transmitted bits is independent and equiprobable with $p(x^i) = 1/K$, then $q_j(x | y)$ is the posterior probability that the transmitted symbol differs from $x$ by $j$ bits.

Therefore, $\tau(x)$ given by (4.161) is the likelihood ratio for testing the hypothesis that an estimate $x$ is in error by one bit against it having zero bits of error. With $q(\cdot)$ as defined in (4.162), then $q_0(x | y) = p(x | y)$, and therefore the ratio in (4.161) may be expressed as,

$$
\tau(x) = \frac{q_1(x | y)}{q_0(x | y)}.
$$

Clearly here, $\tau(x)$ is the probability ratio of $x$ being one bit in error to zero bits in error.

Consequently, when the analysable independent proposal case is as closely matched as possible to the implementable Metropolis Algorithm 4.7.2, then according Theorem 4.8.2, the exponential convergence rate slows down for increasing levels of certainty in the received bit decisions based on the posterior $p(x | y)$.

To be more particular, assume that $\min_x \tau(x)$ occurs at the MAP estimate $\hat{x}$, ie

$$
\hat{x} = \arg \min_x \tau(x).
$$

This is a reasonable assumption for the operation of the algorithm at higher SNR values, where the MAP estimated value is significantly more likely than any one-bit error estimate. Under this assumption, denote by $L_E$ this likelihood ratio for testing the hypothesis that $\hat{x}$ is in error by one bit, versus the alternate hypothesis of zero bits in error,

$$
L_E = \frac{q_1(\hat{x} | y)}{q_0(\hat{x} | y)} = \tau(\hat{x}).
$$

Then according to Theorem 4.8.2, the independent sampler tied to Algorithm 4.7.2 converges as

$$
|p(x_k = x | x_0) - p(x | y)| < R[1 - L_E]^k.
$$

This will present slow convergence with small values of $L_E$ approaching zero, and increasingly fast convergence for larger values. Small values of $L_E$ will occur in the case of a high signal to noise ratio, as in that situation there is much more certainty whether a given estimate $\hat{x}$ is supported by the observation $y$.

Consider then the behaviour of the algorithm utilising the hold up remedy proposed in Section 4.10. This simply involves replacing $p(x | y)$ by $p_\eta(x | y)$ during the computation of acceptance probability $\alpha(\xi | x)$. Perusal of the proofs of Theorems 4.11.1,4.11.2 illustrates that apart from this replacement, they are unaffected, which then implies for this modified algorithm,

$$
|p(x_k = x | x_0) - p(x | y)| < R[1 - L_E^\eta]^k.
$$
As the values of $\eta$ used are in the range $0 < \eta < 1$, and also $0 < L_E < 1$, then,

$$L_E^\eta > L_E.$$ (4.171)

Hence increased convergence rate may be expected in the higher SNR case, for which the assumption in (4.167) holds.

Since $L_E$ becomes smaller as SNR increases, the convergence rate will decrease, which has been empirically observed by other authors [7, 15], but this may be compensated for by choosing $\eta < 1$. This theoretical analysis is consistent with the empirical observations and discussion made in Sections 4.9 and 4.10.

### 4.12 Computationally Efficient Method

A potential criticism of using MCMC methods for multi-user detection is that they are computationally expensive. A useful detector must operate at high data rates, and so the computation time available for each detection calculation is quite small.

The burden of the MCMC algorithms is that they require a large number of iterations of the Markov Chain in order to obtain accurate results. Consequently, the focus of attention for computation is on the calculations required for each iteration of the Markov Chain. The analysis for this section will be based on the Metropolis algorithm, as described in Algorithm 4.7.2.

The first per-iteration step in Algorithm 4.7.2 is to draw a random index $i$ on which to operate. As the number of iterations of the algorithm is only in the range of $10^3$ to $10^4$, there are quite mild requirements on a pseudo-random sequence generator. It can also be shown that algorithm convergence is retained when using a deterministic scan approach instead of a random selection of indices [43]. Consequently, this random number generation may be omitted entirely if it imposes a computational burden. The remainder of the proposal step in Algorithm 4.7.2 merely requires the inversion of one bit in $x_k$.

The majority of the computation in the Metropolis algorithm is to evaluate the acceptance probability for the proposal via the probability ratio in (4.49),

$$\beta_i = \frac{p(y \mid x_{k-1}^{-1}, \xi_k^i = +1)}{p(y \mid x_{k-1}^{-1}, \xi_k^i = -1)} \cdot \frac{\rho_i}{1 - \rho_i},$$ (4.172)

where

$$p(y \mid x) = \frac{\text{const}}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{1}{2\sigma^2} (y - RAx)^T R^{-1} (y - RAx) \right).$$ (4.173)

In this equation, constants such as $R^{-1}$ and $RA$ may be pre-computed. However, for each iteration, it is still necessary to compute three matrix-vector multiplications, each involving $k^2$ scalar multiply-adds.

In an attempt to reduce the multiplies needed, the square factors may be expanded.
to provide the alternate formulation,

\[ p(y \mid x) = K_1 \exp \left( -\frac{1}{2\sigma^2} \left( yR^{-1}y - 2y^T Ax + x^T ARAx \right) \right). \quad (4.174) \]

Noting that \( y \) may be treated as a constant, this reduces computation sightly to two matrix-vector multiplications and one vector inner product.

An important feature of the Metropolis algorithm described in Algorithm 4.7.2, is that the quantity \( p(y \mid x) \) itself is not required, but rather the ratio of likelihoods,

\[ \frac{p(y \mid \xi)}{p(y \mid x)}, \quad (4.175) \]

where \( x \) and \( \xi \) are binary vectors that differ in only one position. This detail may be exploited as a means to significantly reducing the per-iteration computation within the Metropolis algorithm.

In order to do this, first define the notation \( e_i \) as

\[ e_i^T \equiv [0 \ldots 0, 1, 0 \ldots 0], \quad (4.176) \]

with the 1 in the \( i \)-th element. The implied length of the vector is \( k \), the number of users in the system.

Consider the case where \( \xi_k^i = -1 \), then the proposal vector may be expressed as \( \xi_k = (I - 2e_i e_i^T) x_{k-1} \), where \( I \) is the \( k \times k \) identity matrix. Then under a Gaussian noise model, the acceptance probability ratio under consideration is then equal to:

\[
\begin{align*}
\alpha &= \frac{p(y \mid \xi)^n}{p(y \mid x)^n} \\
&= \exp \left[ \frac{-\eta}{2\sigma^2} \left( yR^{-1}y - 2y^T A(I - 2e_i e_i^T)x + x^T (I - 2e_i e_i^T) ARA(I - 2e_i e_i^T)x \right) \\
&\quad + \frac{\eta}{2\sigma^2} \left( yR^{-1}y - 2y^T Ax + x^T ARAx \right) \right] \\
&= \exp \left[ \frac{2\eta}{\sigma^2} \left( y^T A(e_i e_i^T)x + x^T e_i e_i^T ARAx - x^T e_i e_i^T ARA e_i e_i^T x \right) \right] \\
&= \exp \left[ \frac{2\eta}{\sigma^2} (e_i^T Ay - e_i^T ARA(I - e_i e_i^T)x) \right]. \quad (4.179)
\end{align*}
\]

Note that the iteration subscripts have been omitted for clarity. While the result still appears to be computationally expensive, the structure is now easily exploited. The \( e_i^T Ay \) term is simply a scaling of the \( i \)-th element of \( y \) by the \( i \)-th diagonal element of \( A \). To provide the per-iteration performance for the remaining component, a matrix \( M \) is pre-computed so that the \( i \)-th row of \( M \) is defined by,

\[ e_i^T M = e_i^T ARA(I - e_i e_i^T) \quad \forall i. \quad (4.181) \]
4.13. Conclusions

Then a simplified expression for the acceptance probability ratio may be expressed as,

\[ \alpha = \exp \left[ \frac{2\eta}{\sigma^2} (x^i_k - 1) \left( [A]_{i,i} y^i - (e^T_i M) x_{k-1} \right) \right]. \quad (4.182) \]

The \( e^T_i M \) product is simply a row selection, so the only significant calculation is the \( k \)-element inner product, together with a small number of scalar multiplications and additions. In addition, the elements of \( x_k \) are \( \pm 1 \) binary valued, and so a suitable hardware implementation can dramatically further reduce the complexity of the inner product, as the multiplications become only a sign change.

The speedup from these optimisations is considerable. For \( K \) users, it has reduced the operations count from \( 3K^2 \) to \( K \). For the example system profiled in this chapter with \( K = 96 \), that represents a speedup factor of 288. It’s this optimisation for fast per-iteration computation that makes MCMC algorithms feasible for high speed applications such as found in communications systems.

4.13 Conclusions

The implementation of optimal MAP detection in CDMA systems is computationally very demanding, while at the same time it is important to maintain a high throughput. Consequently, it is necessary to utilise less computationally intensive, but sub-optimal methods. In this chapter it has been shown that MCMC methods may be used to achieve this.

By providing a means to sample from a desired posterior probability distribution, the Metropolis-Hastings and Gibbs sampling algorithms described in this chapter closely approximate the true MAP estimate. As an added advantage “soft” outputs are also available, giving bit-error probabilities.

Although the estimates are proven to converge to the true MAP estimate as the number of iterations increases toward infinity, it’s important to consider the performance after a computationally reasonable number of iterations. The work in this chapter characterises this performance, and shows that with appropriate algorithm modifications, good results may be achieved after a computationally reasonable number of iterations.

While appropriate algorithm tuning significantly reduces the number of iterations required, the per-iteration computation structures are also an important element in the viability of MCMC for these type of applications. The computationally efficient likelihood evaluation method presented here is a key element in achieving efficient run-times.
Chapter 5

Application in System Identification

5.1 Introduction

This chapter considers the use of the Metropolis–Hastings method for the modelling of dynamic systems based on measured data. There are many practical situations where it is useful to mathematically model the behaviour of physical systems, for the purposes of analysing or controlling their behaviour. Such a model will typically be constructed so that if given measurements of the input to the physical system, it will be able to simulate an estimate of the real output of the physical process. Knowledge of the mathematical properties of the model may then facilitate the choice of input to the physical system, for the purposes of controlling its output.

5.2 Systems Models

There are many different ways of constructing dynamic system models, and this chapter will only consider a few of the more common approaches. Figure 5.1 shows the general layout of a basic system identification problem. An input $u(t)$ is applied to the system, an output $y(t)$ is observed. This output may be corrupted by the presence of some noise $v(t)$. In this example, the aim of the system identification process is to characterise the system $G$.

Figure 5.1: System to be modelled
One approach to identifying such a system is to derive equations from known physical properties. For example, using Newton’s laws of motion, together with known quantities from the mass or inertia of objects, a physical model may be constructed of a mechanical system without prior knowledge of the input and output data. In this case $G$ would be based on differential equations from the understanding of the physical process. However, in many applications, some quantities may not be known, and these may become the parameters

$$\theta = \{\theta^1, \theta^2, \ldots, \theta^K\}, \quad (5.1)$$

of the model to be estimated.

Alternatively, identification may proceed with an absence of a priori knowledge of the physical structure of the system. In this case a generic black-box model structure is used, with the measured input and output data, $u(t)$ and $y(t)$ respectively used to estimate the parameters in the chosen model structure. The examples in this chapter will assume such a black-box model structure, but the techniques are equally applicable in grey-box models, where the structure is known, but there are a set of unknown physical parameters to be estimated.

The system under consideration may be modelled in either continuous or discrete time. In the examples contained here, discrete time models are used, which arise naturally in the analysis of sampled-data systems. In this case we will consider sets of measured data

$$U \triangleq \{u_1, \cdots, u_N\}, \quad (5.2)$$
$$Y \triangleq \{y_1, \cdots, y_N\}, \quad (5.3)$$

of the uniformly sampled input data samples and measured output data samples respectively. The system identification task will be to produce a model $G$ that may predict the system output $\{y_t\}$, given the observed exogenous inputs $\{u_t\}$.

5.2.1 General Probabilistic Model

In order to be of the most general utility as possible, the MCMC system identification will be first introduced in the context of a very general model. The practical examples will then be implemented as special cases of this model. In the general setting, the system identification task is to estimate the parameters of a stochastic model expressed in the form of an arbitrary probability density function linking the measured output with the model parameters.

Given $\theta \in \mathcal{X} \subseteq \mathbb{R}^n$ as the vector of model parameters, and for a known set of inputs $U$, the model ascribes a probability density for the possible realisation of outputs according to,

$$Y \sim p(Y \mid \theta), \quad (5.4)$$
where $Y$ is the sampled data record of length $N$, as defined in (5.3).

Here, $p(Y \mid \theta)$ is the joint probability density function for the elements of $Y$, and for the purposes of estimation of $\theta$, is considered to be completely described by $\theta$. In practice, this will also depend on the exogenous input sequence $\{u_t\}$, but as this is not part of the estimation objective, it isn’t explicitly notated. The density $p(Y \mid \theta)$ also depends on the choice of model structure but this, along with many other factors, are also implicitly acknowledged rather than notated.

According to this model, the physical process underlying the observation vector $Y$ performs the role of a random sampler with distribution $p(\cdot \mid \theta^*)$, where $\theta^*$ is some “true” value of parameters for that process. The obtained measured data $\{y_t\}$, are then considered just a realisation of a random process. The objective of the identification process is then to characterise the knowledge of the process based on this observation.

### 5.2.2 State Space Models

While simply modelling the dynamic system as a probability density is convenient in analysis, more structure is needed in a practically realisable model. To this end, it’s convenient to describe (5.4) as a Markov process. By introducing a Markov state $x_t$, the model can more specifically represent the dependence in time between the samples $\{y_1, \cdots, y_N\}$, with

$$
	extnormal{At each time-step, a new state is drawn from a random process } p_\theta(x_{t+1} \mid x_t), 	ext{ which is a function of the parameters } \theta \text{ and the present state } x_t. \text{ The output is also a random process, again depending on } \theta \text{ and the present state. Again, the dependence on the input } \{u_t\} \text{ is implicit in this model.}
$$

From this model, by further defining the means by which the uncertainty enters the model, we arrive at the very popular example of the state-space model structure,

$$
x_{t+1} = f_\theta(x_t, u_t) + w_t, \\
y_t = h_\theta(x_t, u_t) + \varepsilon_t.
$$

This state-space model employs a Markov state vector $x_t$ which progresses over time according to an additive random process. The sequences $\{w_t\}, \{\varepsilon_t\}$ are independent random variables, assumed to be zero mean and have covariances of,

$$
\mathbf{E} \{w_t w_t^T\} = Q \\
\mathbf{E} \{\varepsilon_t \varepsilon_t^T\} = R.
$$

An important special case of this general state space model occurs when $f_\theta$ and $h_\theta$ are
linear functions of the state and input. In this situation, the model may be characterised as,

\[
\begin{align*}
    x_{t+1} &= Ax_t + Bu_t + w_t, \\
    y_t &= Cx_t + Du_t + \varepsilon_t,
\end{align*}
\]

for parameter matrices \( A, B, C, D \), and independent zero-mean noise sequences \( \{w_t\}, \{\varepsilon_t\} \). For many systems, these sequences will be assumed to have a Gaussian distribution, in which case

\[
\begin{bmatrix} w_t \\ \varepsilon_t \end{bmatrix} \sim \mathcal{N} \left( 0, \begin{bmatrix} Q & S \\ S^T & R \end{bmatrix} \right).
\]

The parameter vector \( \theta \) consists of a concatenation of the elements of each of the matrices \( A, B, ..., R \).

### 5.2.3 Transfer Function Models

While the linear state-space model structure is sufficiently concrete to be of practical utility, there is still a role for further specialisations. This may be motivated by the large number of individual parameters in the parameter vector \( \theta \) as the size of the state vector \( x_t \) grows. There exist a number of canonical forms of the linear state-space model which are achieved by constraining many elements of the matrices \( \{A, B, C, D\} \) to specific values.

A set of canonical forms are derived from considering difference equations between the input \( \{u_t\} \) and output \( \{y_t\} \). One of the simpler input-output relationships is based on the output error model structure illustrated in Figure 5.2.

\[
\begin{align*}
    w_{t+1} &= a_1 w_{t-1} + \cdots + a_{n_a} w_{t-n_a} + b_1 u_{t-1} + \cdots + b_{n_b} u_{t-n_b} \\
    y_t &= w_t + \varepsilon_t,
\end{align*}
\]

where \( \varepsilon_t \) is a zero mean i.i.d. random process. Using \( q \) as the forward shift operator, we
may define
\[
A(q) = 1 + a_1 q^{-1} + a_2 q^{-2} + \ldots + a_n q^{-n} \tag{5.12}
\]
\[
B(q) = 1 + b_1 q^{-1} + b_2 q^{-2} + \ldots + b_n q^{-n}. \tag{5.13}
\]
Then (5.11) may be expressed in a transfer function format
\[
y_t = G(q) u_t + \varepsilon_t, \tag{5.14}
\]
where
\[
G(q) = \frac{B(q)}{A(q)} \tag{5.15}
\]
The parameter vector \( \theta \) then consists of the difference equation coefficients,
\[
\theta = [b_0, \ldots, b_n, a_1, \ldots, a_n]. \tag{5.16}
\]
This model may be extended to include dynamics in the disturbance term, resulting in
\[
y_t = G(q, \theta) u_t + H(q, \theta) \varepsilon_t, \tag{5.17}
\]
The functions \( G(q, \theta) \) and \( H(q, \theta) \) are both transfer functions which are rational in the forward shift operator \( q \), and parametrised by the elements of \( \theta \). This structure is considered a natural finite-dimensional parametrisation [33], and is known as the Box-Jenkins model structure.

5.3 Parameter Estimation

5.3.1 Maximum Likelihood

Given a model in the form of (5.4), the task of identification becomes to determine an estimate \( \hat{\theta}_N \) of the true underlying parameters \( \theta^* \). Given the nature of this stochastic model, a natural approach to achieve this is via a maximum-likelihood (ML) estimate,
\[
\hat{\theta}_N \triangleq \arg \max_{\theta} p(Y | \theta) \tag{5.18}
\]
This will find the value of \( \theta \) that maximises the likelihood of the event that corresponds to the observed data record. In order to compute the ML estimate, the joint density function may be decomposed into a more useful form. Considering just the first two observations of \( Y, \{y_1, y_2\} \), by Bayes’ rule,
\[
p(y_1, y_2) = p(y_1) p(y_2 | y_1). \tag{5.19}
\]
By successive applications of Bayes’ rule, and conditioning against the parameter vector \( \theta \),

\[
p(Y \mid \theta) = p(y_1 \mid \theta) \prod_{t=2}^{N} p(y_t \mid y_{t-1}, \ldots, y_1, \theta).
\]

(5.20)

Consequently, the log-likelihood may be defined as,

\[
L_\theta(Y) \triangleq \log p(Y \mid \theta) = \log p(y_1 \mid \theta) + \sum_{t=2}^{N} \log p(y_t \mid y_{t-1}, \ldots, y_1, \theta).
\]

(5.22)

Given that the logarithm function is monotonic, (5.18) may equivalently be expressed in the form,

\[
\hat{\theta}_N = \arg \min_{\theta} -L_\theta(Y).
\]

(5.23)

### 5.3.2 Prediction Error Decomposition

An alternate approach to the estimation of \( \hat{\theta} \) involves considering the magnitude of the error that results from the predictor parameterised on \( \theta \). In many applications of interest, the general model (5.4) may be decomposed across each of the time samples according to

\[
y_t = \hat{y}_{t|t-1}(\theta) + \varepsilon_t(\theta),
\]

(5.24)

where \( \hat{y}_{t|t-1}(\theta) \), is a one step ahead predictor of \( y_t \), based on observations strictly prior to time \( t \), and \( \varepsilon_t \) is a zero mean i.i.d. innovations process that is independent of \( \hat{y}_{t|t-1}(\theta) \). If \( E_\theta \{ \cdot \} \) represents expectation with respect to a probability density dependent on \( \theta \), then \( \hat{y}_{t|t-1}(\theta) \) is simply,

\[
\hat{y}_{t|t-1}(\theta) = E_\theta \{ y_t \mid y_{t-1}, \ldots, y_1 \}.
\]

(5.25)

Clearly, when the innovations \( \varepsilon_t(\theta) \) are small, then the model parameterised by \( \theta \) is a good match for the observed data \( Y \). This motivates the cost function,

\[
V_N(\theta) = \frac{1}{N} \sum_{t=1}^{N} \ell(\varepsilon_t(\theta)),
\]

(5.26)

where the function \( \ell(\cdot) \) is some positive mapping, with the common choice being \( \ell(x) = x^T x \). With this cost function, the prediction error (PE) estimation method may be
defined as,

$$\hat{\theta} = \arg \min_{\theta} V_N(\theta).$$  \hspace{1cm} (5.27)

There are clearly similarities between the forms of the ML estimate (5.23) and PE estimate (5.27). These may be made more explicit by considering the scalar case and expressing the likelihood function in terms of the probability density function \( p_\varepsilon(\cdot) \) of the innovations \( \varepsilon_t(\theta) \) by noting,

$$p(y_t \mid y_{t-1}, \ldots, y_0, \theta) = p_\varepsilon(y_t - \hat{y}_{t|t-1}(\theta)).$$ \hspace{1cm} (5.28)

In the Gaussian case where \( \varepsilon_t \sim \mathcal{N}(0, \sigma^2) \),

$$p_\varepsilon(x) = \frac{1}{\sqrt{2\pi}\sigma^2} \exp \left\{ -\frac{x^2}{2\sigma^2} \right\}. \hspace{1cm} (5.29)$$

Then (5.22) may be expressed as,

$$L_\theta(Y) = \log p(y_1 \mid \theta) + \sum_{t=2}^{N} \log p_\varepsilon(\varepsilon(\theta))$$

$$= \log p(y_1 \mid \theta) - \sum_{t=2}^{N} \frac{1}{2} \log 2\pi + \log \sigma + \frac{1}{2\sigma^2} \varepsilon_t^2(\theta). \hspace{1cm} (5.31)$$

In this Gaussian case, for the choice of \( \ell(x) = x^2 \) in (5.26), the PE and ML estimates are equivalent asymptotically as \( N \) increases.

For this choice of \( \ell(x) = x^T x \), the solution to (5.27) is a nonlinear least-squares problem, and as such has been well studied in the area of numerical analysis. While there is no closed form solution to this problem in the general case, there are many iterative search methods that are effective at quickly locating the minimum provided the predictor \( \hat{y}_{t|t-1}(\theta) \) may be evaluated.

However, in the general nonlinear case (5.6), there is no computationally simple method to compute the required marginal probability densities \( p(y_t \mid y_{t-1}, \ldots, y_1) \). Instead, computation-intensive numerical methods are required. An example is included in this thesis showing the application of a particle filter to this task.

For the linear and Gaussian form of the state space representation (5.9), the Kalman filter may be used to compute the one step ahead predictor (5.24), and \( p_\varepsilon(\cdot) \) with relatively light computational burden. For the transfer function model structure in (5.17) there is an even simpler solution through the use of the steady-state Kalman filter. If \( H(q, \theta) \) is constrained to be monic, the predictor \( \hat{y}_{t|t-1}(\theta) \) is then,

$$\hat{y}_{t|t-1}(\theta) = H^{-1}(q, \theta)G(q, \theta)u_t + [1 - H^{-1}(q, \theta)] y_t \hspace{1cm} (5.32)$$
under the assumption that both $H^{-1}(q, \theta)G(q, \theta)$ and $H^{-1}(q, \theta)$ are stable.

5.3.3 Asymptotic Convergence

Due to the presence of noise in the measurements $Y$, we cannot expect the estimate $\hat{\theta}_N$ to equal the parameters of the underlying "true" system. However, with the collection of sufficient measurement data, we may hope that the estimates would eventually converge to the true parameter values.

In cases where the decomposition (5.24) holds, for a very broad range of model structures, the Maximum–Likelihood approach (5.18) is known to obey well defined asymptotic distributional convergence properties in the number of samples $N$. Specifically, as the number of observations $N$ increases, the probability distribution of an estimate $\hat{\theta}_N$ converges to a limit $\theta^*$ at a rate described by

$$\sqrt{N}(\hat{\theta}_N - \theta^*) \xrightarrow{D} \mathcal{N}(0, P) \quad \text{as } N \to \infty$$

(5.33)

where $\mathcal{N}(0, P)$ denotes a zero mean multivariable Gaussian density with covariance matrix $P$. The covariance of the distribution is governed by the shape of $p(Y | \theta)$ in the region of $\theta^*$. In the case where the model structure is linear and the innovations process, $p_e$ is Gaussian, then $P$ may be readily determined. Using $\mathbb{E}\{\cdot\}$ to denote expectation, the covariance matrix $P$ is determined by,

$$P^{-1} \triangleq \lim_{N \to \infty} \frac{-1}{N} \mathbb{E}\left\{\log p(Y | \theta)\right\}_{\theta = \theta^*}$$

(5.34)

Even if the Gaussian assumption is incorrect, the distributional convergence (5.33) still holds, except that the asymptotic covariance matrix (5.34) needs to be adjusted via a certain scalar multiplication [33].

5.3.4 Quantifying Estimation Error

Clearly (5.33) and (5.34) appear useful in the aim of quantifying the magnitude of estimation error $\hat{\theta}_N - \theta^*$ that results from having only a finite set of measurement data available. However, the error distribution in (5.33) is only defined in terms of a limit as $N \to \infty$. In contrast, practical applications may not have the luxury of having sufficient data samples for this limiting behaviour to be useful.

Nevertheless, in the absence of an alternative, estimation of the error $\hat{\theta}_N - \theta^*$ is typically achieved by assuming the convergence (5.33) has effectively occurred for the finite data length $N$ available. By assuming convergence of (5.33), the error between the estimated parameters from finite data and that from an infinite dataset follows the known distribution,

$$\sqrt{N}(\hat{\theta}_N - \theta^*) \sim \mathcal{N}(0, P).$$

(5.35)
From this, with $\chi^2_n$ denoting an $n$-degree of freedom chi-squared distribution, the approximation may be represented as

$$N(\hat{\theta}_N - \theta_*)^T P^{-1}(\hat{\theta}_N - \theta_*)^T \sim \chi^2_n.$$  \hspace{1cm} (5.36)

Consequently, likelihood bounds on the estimation error may be characterised by ellipsoid surfaces in the space of $\theta$. This is commonly used as a means of generating confidence regions for the estimation error in $\hat{\theta}_N$.

Alternatively, there may be a need to quantify the estimation error in an individual $i$'th element, $\hat{\theta}_N^i$, of the parameter vector. This involves determining the properties of the scalar term $\hat{\theta}_N^i - \theta_*^i$. Considering the $i$'th diagonal entry of (5.35), we have for $N \to \infty$ that

$$\hat{\theta}_N^i - \theta_*^i \sim N\left(0, \frac{1}{N} [P]_{i,i}\right).$$  \hspace{1cm} (5.37)

using the notation that $[P]_{i,j}$ is the $i, j$'th element of $P$. From this, a Gaussian approximation to the marginal probability density function of $\theta^i$ may be derived.

This approach is commonly used to provide error bounds for estimates. Hence, for example, a so-called 'two sigma' bound is computed as

$$2\sigma = \frac{2}{N} [P]_{i,i}$$  \hspace{1cm} (5.38)

Based on Gaussian distribution characteristics, this represents an effective 95% confidence interval bound for the error $\hat{\theta}_N^i - \theta_*^i$ to be contained with the magnitude of $2\sigma$.

### 5.3.5 Error in Functions of Parameters

While knowledge of the error in individual model parameters may be important to have, it is often not the specific value of the parameter vector that is important to analysis, but rather some measure of the behaviour of the model itself.

**Example 5.3.1.** Consider the contour plot of a joint probability density $p(\theta)$ of a parameter pair $[\theta^0, \theta^1]$ shown in Figure 5.3.

By way of example, the parameters $\theta^0$ and $\theta^1$ could have a physical significance that represents the location of elements within a structure. However, the performance of the system may be more dependent on the distance between the two elements. In this case, it may also happen that the probability density of a function of the parameters,

$$f(\theta) = |\theta^1 - \theta^0|.$$  \hspace{1cm} (5.39)

Given the density of Figure 5.3, the error in this distance estimate will be much less than implied by the variability of the values of the parameters $\theta^0$ and $\theta^1$.  

While this example is trivial, the same principle applies in more general circumstances. Consider the frequency response of a model parametrised by $\hat{\theta}_N$. Knowledge of the estimation error in the frequency response of a model may be more valuable in determining the error in the overall behaviour of the system than that of the individual parameters themselves. One way to use this may be to determine the gain margin or phase margin of an identified system coupled to a proposed controller. In contrast to Example 5.3.1, it wouldn’t generally be convenient to re-parameterise the system in terms of frequency response parameters in order to achieve this error estimation.

To quantify this estimation error in functions of parameters, a further approximating step is usually introduced. In this, the functional relationship is reduced to a first order expansion, and Gauss’ approximation formula [31] is then used to estimate that ($^*$ denotes conjugate transpose)

$$ f(\hat{\theta}_N) - f(\theta^*) \sim \mathcal{N} \left( 0, \frac{1}{N} \frac{df(\hat{\theta}_N)^*}{d\theta} P^{-1} \frac{df(\hat{\theta}_N)}{d\theta} \right). $$

(5.40)

Despite the several approximations in the error quantifications (5.36)-(5.40), these approximations have proved effective and sufficiently accurate in many applications. In these applications, $N$ is generally large and hence $\|\hat{\theta}_N - \theta^*\|$ is small so that first order approximations to functions of $\hat{\theta}_N$ are informative.

Equally, there are also many applications where the available estimation data is limited. Having less data will imply that the errors are greater, and so it is in these applications that it generally becomes more important to accurately quantify estimation error and hence the model quality. However this same reduction in data record length also makes the error quantification more difficult to do, as the assumptions underlying (5.36)-(5.40) become problematic [16]. Consequently there is motivation for improving on these methods for the case of short data records.
5.4 A Bayesian Approach to Estimation and Error Quantification

Given the limitations of conventional methods in the quantification of estimation errors, the remainder of this chapter will examine an alternate Bayesian approach. This involves using MCMC methods to calculate the posterior density \( p(\theta \mid Y) \). Such a posterior density represents all the information that can be extracted from the combination of observed data and prior knowledge about the parameters, and from it may be computed representations of error such as confidence regions on functions of parameters.

For given values of \( \theta \) and \( Y \), the posterior \( p(\theta \mid Y) \) may be computed from the likelihood function in (5.20) by combining in the prior information about \( \theta \). This is done via Bayes’ rule, with

\[
p(\theta \mid Y) = \frac{p(Y \mid \theta)p(\theta)}{p(Y)},
\]

where \( p(\theta) \) is the a-priori distribution of \( \theta \) and \( p(Y) \) may be considered a normalising factor given as

\[
p(Y) = \int_{\mathcal{X}} p(Y \mid \theta)p(\theta) \, d\theta,
\]

to ensure \( p(\theta \mid Y) \) has unit volume. As \( p(Y) \) is a constant with respect to \( \theta \), it is often unnecessary to explicitly compute its value. The proportional constant may be left unknown for applications where only comparisons or ratios between different values of \( \theta \) are required.

While this ability to easily evaluate \( p(\theta \mid Y) \) is useful, it isn’t often useful in itself. Typically it will be necessary to integrate \( p(\theta \mid Y) \) over a multi-dimensional region that corresponds to the event whose probability is required. This underlies the need of a suitable optimisation or integration method to use the observable properties of \( p(\theta \mid Y) \) in order to make use of the ability to perform this evaluation.

In Section 4.5, the maximum a-posterior (MAP) estimate was introduced as a means of optimally estimating transmitted data in a communications system. In this context, the MAP estimate \( \hat{\beta}_N \) may be similarly taken as

\[
\hat{\beta}_N \triangleq \arg \max_{\theta} \log p(\theta \mid Y)
\]

\[
= \arg \max_{\theta} \left[ \log p(Y \mid \theta) + \log p(\theta) \right].
\]

When there is little prior information on \( \theta \), then \( p(\theta) \) approximates a constant. For such a diffuse \( p(\theta) \), the point estimates \( \hat{\beta}_N \) in (5.44) and \( \hat{\theta}_N \) in (5.18) co-incide. In other cases \( p(\theta) \) is sufficiently smooth, or otherwise regular that existing algorithms, such as a gradient based search used to compute the ML estimate (5.18), can equally well be applied to compute the MAP estimate (5.43).
However, even in cases where there would be benefit in doing so, the further steps of using the posterior $p(\theta \mid Y)$ as a means of computing estimate accuracy, or using the alternative Bayesian point estimate

$$\hat{\beta}_N = \mathbf{E}\{\theta \mid Y\}$$

(5.45)

are not routinely employed, due to the associated high computational burdens. The computational burden comes about as a result of the integration intrinsic to the computation of the expectation.

For example, if the marginal density of only a particular $i$’th parameter element $\theta^i$ is required, then this requires numerical computation of the multidimensional integral

$$p(\theta^i \mid Y) = \int p(\theta \mid Y) d\theta^i \cdots d\theta^{i-1} d\theta^{i+1} \cdots d\theta^n.$$  

(5.46)

This is problematic, since if $k$ points on the density curve $p(\theta^i \mid Y)$ are required to represent it, then (5.46) implies the numerical evaluation of $k$ multidimensional integrals, and this latter dimension could be relatively large: a modest fifth order transfer function model would imply a nine dimensional integral in order to obtain the marginal density on just one parameter. As demonstrated in Section 2.3.5, integrals of such high dimension are very problematic.

As an additional complication, via (5.20) the posterior (5.41) involves a product over $N$ terms. While this is not an onerous computational burden in itself, it does introduce issues with numerical representation and precision. Even for modest $N$, the dynamic range of the posterior can be very large, resulting in numerical difficulties. One strategy to circumvent this is to instead work with $\log p(\theta \mid Y)$, but then the marginal density (5.46) cannot be computed in this form, since the logarithm and integral operator do not commute.

These computational difficulties suggest that despite the Bayesian approach’s attractive properties, it is not viable for the analysis of systems with a parameter vector of dimension beyond a trivial size. This motivates the development of methods to approximate the explicit Bayesian solution while still remaining computationally feasible.

### 5.5 A Markov Chain Monte–Carlo Solution

There are a number of similarities between this Bayesian estimation problem and that of the multi-user detection example in the previous chapter. In both cases, a model exists such that it is possible to evaluate the probability density $p(Y \mid \theta)$ of the measurements given the parameters, but is desired instead to evaluate the density of functions of the parameters $p(f(\theta) \mid Y)$. This lends the problem again to statistical estimation via Markov Chain Monte–Carlo methods.

The principal difference in this new application is that the parameter vector $\theta$ is
5.5. A Markov Chain Monte–Carlo Solution

now in an uncountable space, rather than the discrete space of the transmitted symbols in the CDMA example. This will require the algorithms to be extended, and also the theoretical treatment in order to accommodate the different properties of the continuous parameter space.

To be more concrete, the elements of the parameters are here taken to be real numbers, with \( \theta \in \mathcal{X} \subseteq \mathbb{R}^n \). Much of the theory presented in this chapter also applies when the parameter space \( \mathcal{X} \) is extended to other uncountable spaces, but this restriction allows the algorithms to be presented in a simpler form, and sufficient for the application at hand.

The MCMC approach again involves numerically computing the required densities by a strategy of first generating a random sequence of realisations \( \{ \theta_k \} \) with each individual element \( \theta_k \in \mathcal{X} \subseteq \mathbb{R}^n \) and with the limiting distribution of \( \{ \theta_k \} \) equal to the desired posterior density; viz.

\[
\lim_{k \to \infty} p(\theta_k = \theta | \theta_0) = p(\theta | Y) \quad \forall \theta_0 \in \mathcal{X} \subseteq \mathbb{R}^n.
\]  

As in the case of the discrete space, the simulated realisation \( \{ \theta_k \} \) is then used as if it were a random sample from \( p(\theta | Y) \). Provided the conditions are met such that distributional convergence holds, then via a law of large numbers based argument, this method will lead to consistent estimates of various quantities related to the distribution of the parameters.

As a simple example, this sampling approach allows the numerical computation and consistent estimation of the conditional expectation \( \mathbb{E} \{ f(\theta) | Y \} \) as

\[
\mathbb{E} \{ f(\theta) | Y \} = \int_{\mathcal{X}} f(\theta)p(\theta | Y)d\theta 
\]

\[
\approx \frac{1}{M} \sum_{k=1}^{M} f(\theta_k) \triangleq \hat{f}_M,
\]

for \( f \) an arbitrary Borel measurable function. The integer \( M \) corresponds to the number of iterations of the sampling algorithm. Clearly the accuracy of the estimate will improve with increasing \( M \), but in practical application this is limited by the time and computational resources available for the estimate.

The motivating example of this chapter is the computation of posterior density functions. With the availability of samples from \( p(\theta | Y) \), these may also be evaluated via sample histograms:

\[
p(f(\theta) \in A | Y) \approx \frac{1}{M} \sum_{k=1}^{M} I_{f^{-1}(A)}(\theta_k).
\]
Here, $I_X(\xi)$ is the indicator function for $\xi \in X$ defined as

$$I_X(\xi) = \begin{cases} 1 & : \xi \in X \\ 0 & : \text{Otherwise} \end{cases}$$

and $A$ is any $f$-measurable set.

While this is an effective means of computing the posterior density, it relies on ability to construct a random number generator with the given joint density $p(\theta \mid Y)$. As demonstrated in the previous chapter, the MCMC methods provide a means to do this when the related task of evaluating the function $p(\theta \mid Y)$ for a given $\theta$ is straightforward.

For example, in the common situation where the general model structure (5.17) is employed, then via (5.41) and (5.20), the evaluation of $p(\theta \mid Y)$ for a given $\theta$ reduces to first computing the prediction error

$$\varepsilon_t(\theta) = y_t - \hat{y}_{t_{t-1}}(\theta),$$

using the standard formula (5.32), and then evaluating the known density $p_\varepsilon(\cdot)$ for the innovations $\varepsilon_t$ in (5.17) at this point.

Given that this evaluation may be conveniently achieved for the common system identification model structures, MCMC methods become a viable option for characterising posterior density functions. The MCMC sampler is now presented for this context.

### 5.5.1 MCMC Sampler over Uncountable Spaces

Algorithm 5.5.1 describes the Metropolis–Hastings algorithm as applied to a parameter vector on the space $X$.

**Algorithm 5.5.1. (Markov Chain Monte Carlo Sampler)**

1. Initialise $\theta_0$ at some value such that $p(\theta_0 \mid Y) > 0$ and set $k = 1$;

2. At iteration $k$, consider a candidate value $\xi_k$ for $\theta_k$ which is drawn from a proposal density $\gamma(\xi_k \mid \theta_{k-1})$. That is, find a possible realisation for $\theta_k$ as

$$\xi_k \sim \gamma(\cdot \mid \theta_{k-1});$$

3. Compute the acceptance probability

$$\alpha(\xi_k \mid \theta_{k-1}) = \min \left\{ 1, \frac{p(\xi_k \mid Y)}{p(\theta_{k-1} \mid Y)} \cdot \frac{\gamma(\theta_{k-1} \mid \xi_k)}{\gamma(\xi_k \mid \theta_{k-1})} \right\};$$

4. Accept the proposed $\xi_k$ and set $\theta_k = \xi_k$ with probability $\alpha(\xi_k \mid \theta_{k-1})$, otherwise leave $\theta_k$ unchanged by setting $\theta_k = \theta_{k-1}$;

5. Increment $k$ and return to step 2.
Note that step 4, while appearing complex, may be simply implemented by drawing a random variable

\[ z \sim U_{[0,1]}(\cdot) \]  

(5.55)

from a uniform distribution on \([0, 1]\) and setting

\[ \theta_k = \begin{cases} 
  \xi_k; & \text{if } z < \alpha(\xi_k \mid \theta_{k-1}), \\
  \theta_{k-1}; & \text{otherwise}. 
\end{cases} \]  

(5.56)

The choice of the proposal density \(\gamma(\xi_k \mid \theta_{k-1})\), in step 2 is a crucial design step. It’s here that the power of the Metropolis algorithms comes about in their flexibility in freely allowing the choice of this proposal density. If it were possible to easily draw samples from the true posterior density, then one would do so, and not require the Metropolis algorithm. Where this is not possible, the Metropolis algorithm instead allows the use of a different proposal sampling distribution, and then it modifies the set of samples in order to construct the correct distribution. The advantage of this is that a distribution may be chosen on the basis of being computationally cheap to sample from. However, it must still be recognised that the choice of distribution will affect the speed of convergence of the algorithm.

For this application of MCMC methods, the main proposal distributions to be profiled in this thesis are based on the random walk. This is the density implied by

\[ \xi_k = \theta_{k-1} + \nu_k, \]  

(5.57)

with \(\nu_k\) being a random perturbation. Note that there is still considerable freedom of design within the random walk framework, as the distribution of \(\nu_k\) may be freely chosen for optimal overall algorithm performance.

As successive proposal steps are not required to be independent of the present state of the Markov Chain, it is convenient to use this present state \(\theta_{k-1}\) as a means of placing proposal samples in regions of \(\mathcal{X}\) where there is a sufficiently high probability of them being accepted.

A special, but common, case occurs where the probability density \(p_{\nu}(\cdot)\) governing \(\nu_k\) is symmetric, ie

\[ p_{\nu}(x) = p_{\nu}(-x). \]  

(5.58)

An obvious example of this is when \(\{\nu_k\}\) is generated by a zero-mean Gaussian process. When (5.58) holds, then the probability of proposing state \(\xi\) from a state \(\theta\) is

\[ \gamma(\xi \mid \theta) = p_{\nu}(\xi - \theta) = p_{\nu}(\theta - \xi) = \gamma(\theta \mid \xi), \]  

(5.59)
which is equal to probability of proposing in the reverse direction. This means that
Algorithm 5.5.1 reduces to being the more basic Metropolis algorithm (§2.6.2), and (5.54)
is simplified to
\[
\alpha(\xi_k \mid \theta_{k-1}) = \min \left\{ 1, \frac{p(\xi_k \mid Y)}{p(\theta_{k-1} \mid Y)} \right\}.
\] (5.60)

The Metropolis specialisation in (5.60) more clearly exposes an intuitive explanation
of Algorithm 5.5.1. Namely, realisations \(\theta_k\) which converge in a distributional sense to a
target distribution \(p(\theta \mid Y)\) are obtained by first drawing a random proposal \(\xi_k\) from an
alternate distribution. If the particular realisation \(\xi_k\) is more likely to be a realisation
drawn from the density \(p(\cdot \mid Y)\) than state of the previous iteration \(\theta_{k-1}\), then we have
that
\[
p(\xi_k \mid Y) > p(\theta_{k-1} \mid Y).
\] (5.61)

In the Metropolis case (5.60), under this circumstance \(\xi_k\) is definitely used as a new
realisation \(\theta_k = \xi_k\). On the other hand, if the randomly drawn proposal \(\xi_k\) is less
likely than the previous state \(\theta_{k-1}\), then it may be discarded, and the previous state
kept. Whether or not it is actually discarded is randomly decided. More specifically, a
proposed \(\xi_k\) less likely than \(\theta_{k-1}\) is retained (in that \(\theta_k = \xi_k\)) with probability
\[
\alpha(\xi_k \mid \theta_{k-1}) = \frac{p(\xi_k \mid Y)}{p(\theta_{k-1} \mid Y)}.
\] (5.62)

The higher the likelihood of the proposed state, the better the probability for acceptance,
but there is still the possibility to transitioning to less probable states in order to allow
the correct target distribution to be met.

This leads to a process which first simulates an arbitrary Markov chain with conven-
ient transition density \(\gamma(\theta_k \mid \theta_{k-1})\), and then by appropriately modifying the resulting
samples via function evaluations of \(p(\theta_k \mid Y)\) it is able to yield vector realisations dis-
buted, according to \(p(\theta \mid Y)\).

The intent is similar to that of Importance Sampling, as described in Section 2.3.4.
Samples are drawn from a computationally feasible density, and then algorithmically
modified to instead follow the desired distribution. However, MCMC methods differ
from Importance Sampling in that the Markov state may be used as part of that proposal
density, potentially making it possible to use a sampler that is much more efficient to use
in the algorithm.

5.6 Markov Model

As the previous chapter has shown, an attractive feature of the MCMC algorithms is that
in implementing a Markov Chain, extensive theory is available to analyse the numerical
behaviour [35, 39]. This facilitates a rigorous convergence analysis to establish the va-
lidity of the approach. In Section 3.3, some useful aspects of Markov chain theory were presented for Markov chains over non-countable spaces. In the following, these results are used as the basis of a convergence analysis of Algorithm 5.5.1 applied to the posterior \( p(\theta | Y) \).

As an initial step in this analysis, it is necessary to develop a Markov model that describes the behaviour of Algorithm 5.5.1. At each step of the algorithm, a new sample \( \theta_k \) is produced, the probability density of which depends only on the previous state \( \theta_{k-1} \). This density is determined by a combination of the proposal and acceptance probabilities at steps 2 and 4 respectively.

Consequently, the mechanism for generating a new sample \( \theta_k \) is a time-homogeneous Markov chain with a transition density \( K(\theta_k \mid \theta_{k-1}) \). For the case where \( \theta_k \neq \theta_{k-1} \), the only way to reach a new \( \theta_k \) is if that the new state drawn from the proposal density. So in this case \( K(\theta_k \mid \theta_{k-1}) \) is simply given as the product of the probability \( \gamma(\xi \mid \theta) \) of proposing a move \( \xi \), times the probability \( \alpha(\xi \mid \theta) \) of accepting it. This results in a Markov Kernel

\[
K(\theta_k = \xi_k \mid \theta_{k-1}) = \alpha(\xi_k \mid \theta_{k-1}) \gamma(\xi_k \mid \theta_{k-1}) \quad \text{for} \quad \theta_k \neq \theta_{k-1}. \tag{5.63}
\]

However, in the special case of \( \theta_k = \theta_{k+1} \), it’s necessary to also consider the effect of rejected proposals, as these will always leave the state unchanged. The most convenient way to formulate this is to use the law of total probability and integrate all the cases where there is a state transition, resulting in

\[
K(\theta_k = \theta_{k-1} \mid \theta_{k-1}) = r(\theta_{k-1}), \tag{5.64}
\]

where

\[
r(\theta_{k-1}) = 1 - \int_{X_{\theta_{k-1}}} \alpha(\xi \mid \theta_{k-1}) \gamma(\xi \mid \theta_{k-1}) \, d\xi
\]

is the probability of no change in the value of \( \theta_k \) from one iteration to another. The integration is over the region

\[
X_{\theta_{k-1}} = \{ \xi \in X : \xi \neq \theta_{k-1} \}, \tag{5.65}
\]

which consists of all of the state space \( X \), except the state \( \theta_{k-1} \). This exception is to allow for the possibility of proposing again \( \theta_{k-1} \).

In combining these two cases, the Markov transition kernel may be expressed as,

\[
K(\theta_k = \xi_k \mid \theta_{k-1}) = \alpha(\xi_k \mid \theta_{k-1}) \gamma(\xi_k \mid \theta_{k-1}) I_{X_{\theta_{k-1}}} (\xi_k) + \delta(\xi_k - \theta_{k-1}) r(\theta_{k-1}), \tag{5.66}
\]

where the delta function is a Dirac delta.

If rather than dividing the computation based on whether \( \theta_k = \theta_{k-1} \), the division
is instead based on whether or not the proposal is accepted, then a slightly different formulation may be achieved. In this case

\[ K(\theta_k = \xi_k \mid \theta_{k-1}) = \alpha(\xi_k \mid \theta_{k-1})\gamma(\xi_k \mid \theta_{k-1}) + \delta(\xi_k - \theta_{k-1}) \left( 1 - \int_{\mathcal{X}} \alpha(\xi \mid \theta_{k-1})\gamma(\xi \mid \theta_{k-1}) d\xi \right) . \]  

(5.67)

By inspection, these two formulations are the same, as it simply involves moving a term equivalent to \( \alpha(\theta_{k-1} \mid \theta_{k-1})\gamma(\theta_{k-1} \mid \theta_{k-1}) \) from one side of the sum to the other.

As well as the transition probability density from one point in the state space to another, it is also useful to consider the transition probability \( P(A \mid \theta) \) from one point in the space to a region \( A \) of states. This may be expressed as

\[ P(A \mid \theta) = \int_A K(\xi \mid \theta) \mu(d\xi), \]  

(5.68)

with \( \mu \) being Lebesgue measure, and \( A \) being any \( \mu \)-measurable set. Furthermore, in what follows, for brevity \( \mu(d\xi) \) will be shortened to simply \( d\xi \) (or equivalent).

### 5.7 MCMC Convergence

A first step in evaluating the ergodic behaviour of samples from the MCMC sampler will be to characterise its invariant density, as described in Section 3.3.4. This is a state probability density, that if reached at some iteration of the chain, will remain the density of the states for all following time steps. Clearly it is desired that the MCMC sampler converge to \( p(\theta \mid Y) \) as an invariant density. After establishing that \( p(\theta \mid Y) \) is indeed an invariant density, this section then demonstrates the conditions under which the Markov chain may be assured to actually reach that invariant distribution.

#### 5.7.1 Invariant Density

In the case where \( \theta_{k-1} \) is drawn randomly according to a probability density function \( \pi_{k-1}(\theta) \). Then the density function \( \pi_k(\theta) \) for the subsequent sample \( \theta_k \) from this Markov chain, is given by the law of total probability as

\[ \pi_k(\theta_k) = \int_{\mathcal{X}} K(\theta_k \mid \theta_{k-1})\pi_{k-1}(\theta_{k-1}) d\theta_{k-1}. \]  

(5.69)

For the realisations \( \{\theta_k\} \) generated by Algorithm 5.5.1 to converge in a distributional sense to realisations having a given constant density \( \pi(\theta) \), then \( \pi(\theta) \) must be an invariant distribution with respect to the transition kernel \( K(\theta_k \mid \theta_{k-1}) \). This is achieved when

\[ \pi_{k-1}(\theta_{k-1}) = \pi_k(\theta_k) = \pi(\theta). \]  

(5.70)
Consequently, as a test for the required stationary distribution, it is required that,

$$\pi(\theta) = \int_X K(\theta \mid \xi) \pi(\xi) \, d\xi.$$  \hfill (5.71)

With this aim made clear, we now establish that Algorithm 5.5.1 is targeted at the posterior $p(\theta \mid Y)$ of interest.

**Lemma 5.7.1.** The posterior density $p(\theta \mid Y)$ defined by (5.41) is an invariant density of the Markov chain realised by Algorithm 5.5.1.

*Proof.* The first stage of this proof will be to establish that Algorithm 5.5.1 implements a Markov chain whose kernel is in detailed balance with the distribution $p(\theta \mid Y)$. As described in Section 3.2.8, this occurs when.

$$p(\theta \mid Y) K(\xi \mid \theta) = p(\xi \mid Y) K(\theta \mid \xi).$$  \hfill (5.72)

To verify this condition, assume first the case that $\xi \neq \theta$, where the proposed new state $\xi$ is different to the existing state. Then according to the formulation (5.63) for the transition density of the Markov chain realised by Algorithm 5.5.1, and using the acceptance probability expression (5.54),

$$p(\theta \mid Y) K(\xi \mid \theta) = p(\theta \mid Y) \gamma(\xi \mid \theta) \times \min \left\{ 1, \frac{p(\xi \mid Y)}{p(\theta \mid Y)}, \frac{\gamma(\theta \mid \xi)}{\gamma(\xi \mid \theta)} \right\} = \min \{ p(\theta \mid Y) \gamma(\xi \mid \theta), p(\xi \mid Y) \gamma(\theta \mid \xi) \}. \hfill (5.73)$$

This represents the probability of, at any given time step, reaching a state sequence of $\theta$ followed by $\xi$. Applying then the same analysis to the reverse sequence of $\xi$ followed by $\theta$ results in the expression,

$$p(\xi \mid Y) K(\theta \mid \xi) = p(\xi \mid Y) \gamma(\theta \mid \xi) \times \min \left\{ 1, \frac{p(\theta \mid Y)}{p(\xi \mid Y)}, \frac{\gamma(\theta \mid \xi)}{\gamma(\xi \mid \theta)} \right\} = \min \{ p(\xi \mid Y) \gamma(\theta \mid \xi), p(\theta \mid Y) \gamma(\xi \mid \theta) \}. \hfill (5.74)$$

Therefore, comparing (5.73) and (5.74) and noting that the $\min \{ \cdot, \cdot \}$ operation is symmetric, implies that (5.72) holds when $\xi \neq \theta$.

Similarly, considering now the case of $\xi = \theta$, then (5.72) trivially holds simply by substitution of $\xi = \theta$ into the left hand side of (5.72).

Therefore, (5.72) holds for all possible transitions, and Algorithm 5.5.1 yields a Markov chain with a kernel that is in detailed balance with the desired posterior density $p(\theta \mid Y)$.

It now remains to be demonstrated that $p(\theta \mid Y)$ meets the test in (5.71) for a stationary distribution $\pi(\theta)$. Substituting then $p(\theta \mid Y)$ for $\pi(\cdot)$ into the right hand side
of (5.71) and using (5.72) then implies

$$\int_{\mathcal{X}} K(\theta \mid \xi)p(\xi \mid Y) \, d\xi = \int_{\mathcal{X}} K(\xi \mid \theta)p(\theta \mid Y) \, d\xi = p(\theta \mid Y) \int_{\mathcal{X}} K(\xi \mid \theta) \, d\xi = p(\theta \mid Y)$$

(5.75)

where the transition to the last line follows since \( K(\xi \mid \theta) \) is a probability density function and hence integrates to one. As this equates to the left hand side of (5.71) for the substituted distribution, invariance is established.

Clearly the acceptance probability of the Metropolis-Hastings algorithm is designed so that the detailed balance criteria in (5.72) is met. This opens the possibility to alter this algorithm via changing the acceptance probability formulation. The main requirement of any alternate acceptance probability is that detailed balance be maintained. In the discrete case illustrated in Chapter 4, the Gibbs Sampler was shown in Section 4.7.3 to be a form of the Metropolis–Hastings algorithm with an alternate acceptance probability.

### 5.7.2 Distributional Convergence

As the desired posterior density \( p(\theta \mid Y) \) is an invariant density, it is a candidate for being the density that realisations of Algorithm 5.5.1 might converge to. Of course, the desire is to ensure that it is the only candidate, and only some further mild requirements are needed in order to ensure that this is indeed the case. To establish this and further results related to convergence, it is useful to define the associated invariant distribution

$$\varphi(A) = \int_A p(\xi \mid Y) \, d\xi$$

(5.76)

and also define the supports \( \Theta, \Gamma \) of \( p(\theta \mid Y), \gamma(\xi \mid \theta) \) as

$$\Theta \triangleq \{ \theta \in \mathcal{X} : p(\theta \mid Y) > 0 \}, \quad (5.77)$$

$$\Gamma \triangleq \{ \xi \in \mathcal{X} : \gamma(\xi \mid \theta) > 0, \forall \theta \in \Theta \}. \quad (5.78)$$

This allows the definition of certain assumptions that will be required to extend the results. Two assumptions are required in order to assure the uniqueness of the invariant density.

**Assumption 5.7.1.** The support \( \Gamma \) of the proposal \( \gamma(\xi \mid \theta) \) contains the support \( \Theta \) of the posterior \( p(\theta \mid Y) \):

$$\Theta \subseteq \Gamma. \quad (5.79)$$

**Assumption 5.7.2.** The target posterior is bounded and its support is connected:

$$p(\theta \mid Y) < \kappa < \infty, \quad \Theta \text{ is a connected set.} \quad (5.80)$$
5.7. MCMC Convergence

The first of these assumptions sets a requirement on the design of the proposal density \( \gamma(\xi \mid \theta) \). Essentially it means that the proposal must be designed so that it can generate proposals \( \{\xi_k\} \) throughout the entire support \( \Theta \). Clearly this is necessary for any sensible implementation of Algorithm 5.5.1, as there must be a means to reach all of the region of \( \Theta \) from which samples are to be drawn.

When a random walk proposal is used, some care needs to be taken to ensure that (5.79) is met. If the system noise model is such that \( \Theta = \mathcal{X} \), the random walk will generate proposals through the entire support, as the walk origin may be spread right through the parameter space \( \mathcal{X} \). When there are regions of \( \mathcal{X} \) that are excluded from \( \Theta \), some care may be required to ensure that the proposal can indeed reach the whole of \( \Theta \), but this generally remains quite a mild requirement, as the proposal design may be crafted to suit this requirement.

The second assumption places restrictions on the types of posteriors \( p(\theta \mid Y) \) that may be implemented by the algorithm. The requirement for boundedness is not restrictive, particularly as if it were indeed unbounded, the numerical evaluation of \( p(Y \mid \theta) \) within the algorithm would not be possible anyhow. The connectedness of \( \Theta \) relates to the first assumption. For a connected \( \Theta \), it is quite easy to ensure that a random walk proposal started from within \( \Theta \), will indeed reach the whole of \( \Theta \).

Using these two assumptions, the uniqueness of \( p(\theta \mid Y) \) as a candidate for what Algorithm 5.5.1 might converge to is established as follows.

**Lemma 5.7.2.** Suppose that Assumptions 5.7.1–5.7.2 hold, Then the invariant density \( p(\theta \mid Y) \) of the Markov chain realised by Algorithm 5.5.1 is unique. That is, it is the only density satisfying (5.71).

**Proof.** According to (5.63), for \( \xi \neq \theta \) the Markov kernel associated with Algorithm 5.5.1 is given by

\[
K(\xi \mid \theta) = \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) = \min \left\{ \gamma(\xi \mid \theta), \frac{p(\xi \mid Y)}{p(\theta \mid Y)} \gamma(\theta \mid \xi) \right\}.
\]

(5.81)

(5.82)

For \( \xi = \theta \), this will also be a lower bound, as this case is distinct only in the fact that the probability is increased due to proposal rejections. Therefore under Assumption 5.7.1, \( K(\xi \mid \theta) > 0 \) for any \( \xi, \theta \in \Theta \) and hence via (5.68), (5.76) and (5.77), for any \( \theta \in \Theta \)

\[
\varphi(A) > 0 \Rightarrow P(A \mid \theta) > 0.
\]

(5.83)

Therefore, using Definition 3.3.1 the Markov chain realised by Algorithm 5.5.1 is \( \varphi \)-irreducible. Then, by Theorem 3.3.2 the chain is also recurrent, and hence by Theorem 3.3.1, the invariant measure \( \varphi \) is unique. Given also the boundedness from Assumption 5.7.2, the density \( p(\theta \mid Y) \) is also unique. \( \square \)
This result rules out the possibility of there being another stationary density that the chain could possibly converge to. Instead, if the realisations from the Markov chain implemented by Algorithm 5.5.1 ever converge to having a stationary density, the only possibility for that density is the desired posterior $p(\theta \mid Y)$.

To address the issue of actual convergence, it’s necessary to draw on Markov chain ergodic results, as presented in Section 3.3. To do this, recall first the definition (5.68) of the transition distribution $P(A \mid \theta)$ of the associated Markov chain, where it may be expressed as the integral over the set $A$.

$$P(A \mid \theta) = \int_A K(\xi \mid \theta) \mu(d\xi), \quad (5.84)$$

Further to this, define by $P^n(A \mid \theta_0)$ the distribution of the $n$’th iteration of the chain when started at $\theta_0$. This may be computed recursively according to [35, Theorem 3.4.2]

$$P^n(A \mid \theta_0) = \int P^{n-1}(A \mid \xi) P(d\xi \mid \theta_0). \quad (5.85)$$

With this formulation of $P^n$ in place, it may be used to measure distributional convergence of the Markov chain. Convergence to the invariant distribution $\varphi$ defined by (5.76) may be established as follows.

**Theorem 5.7.1.** **Under the same assumptions as Lemma 5.7.2**

$$\lim_{n \to \infty} \sup_{A \in \sigma(\mathcal{X})} |P^n(A \mid \theta_0) - \varphi(A)| = 0 \quad (5.86)$$

for $\varphi$ almost all $\theta_0 \in \Theta$, where $\varphi$ is defined by (5.76) and $\sigma(\mathcal{X})$ is the Borel sigma algebra of sets on $\mathcal{X}$.

**Proof.** In the proof of Lemma 5.7.2 above, the Markov chain $P(A \mid \theta)$ has been established as positive recurrent with invariant measure $\varphi$ given by (5.76). Furthermore, via (5.83), (5.85), $P^n(A \mid \theta) > 0$ for any set $A$ satisfying $\varphi(A) > 0$ and hence the chain $P(A \mid \theta)$ is aperiodic for any starting $\theta_0 \in \Theta$. Application of Theorem 3.3.6 then completes the proof. 

**5.7.3 Sample Averages**

While this distributional convergence is sufficient to satisfy that the MCMC algorithm produces a sensible approximation to the desired posterior $p(\theta \mid Y)$, it still falls a step short of demonstrating the actual utility of the algorithm. It is being proposed to use an estimate $\hat{f}_M$ of the form (5.49), (5.50), so while convergence in distribution is of interest, it is equally important to consider convergence of sample averages.

In order to obtain a result for the sample averages from the Markov chain, it will be necessary to place another requirement on the proposal $\gamma(\cdot)$ used in the algorithm.
Assumption 5.7.3. The proposal density is bounded.

\[ \gamma(\xi \mid \theta) < \kappa < \infty, \quad \forall \theta \in \Theta; \quad (5.87) \]

It isn’t difficult to ensure that this assumption is met, as it only requires that the proposal be designed to be distributed across \( \mathcal{X} \), rather than containing point mass. With this assumption, it’s possible to show that sample averages do indeed converge to the required estimates, as shown in Theorem 5.7.2.

Theorem 5.7.2. Under the assumptions of Lemma 5.7.2 and the further conditions that (5.87) holds and that \( f : \mathcal{X} \rightarrow \mathbb{R} \) satisfies

\[ \int_{\mathcal{X}} |f(\theta)| p(\theta \mid Y) d\theta < \infty, \]

then for the sequence \( \{\theta_k\} \) generated by Algorithm 5.5.1

\[ \lim_{M \rightarrow \infty} \frac{1}{M} \sum_{k=1}^{M} f(\theta_k) = \int_{\mathcal{X}} f(\theta) p(\theta \mid Y) d\theta \]

with probability one.

Proof. Firstly, as established in Lemma 5.7.1,

\[ \varphi(A) = \int_A p(\theta \mid Y) d\theta \]

is an invariant distribution of the Markov chain \( \mathbf{P}(A \mid \theta) \) realised by Algorithm 5.5.1, and hence \( \mathbf{P}(A \mid \theta) \) is positive recurrent and \( \varphi \) irreducible. Therefore, if there exists a bounded function \( h : \mathcal{X} \rightarrow \mathbb{R} \) that is harmonic with respect to \( \mathbf{P}(A \mid \theta) \) in that

\[ h(\theta) = \int_{\mathcal{X}} h(\xi) \mathbf{P}(d\xi \mid \theta) \]

then it must hold that \( h(\theta) = \overline{h} \) a constant for \( \varphi \) almost all \( \theta \). Now, define

\[ A^+ \triangleq \{ \theta : p(\theta \mid Y) > 0 \}, \quad F \triangleq \{ \theta : h(\theta) \neq \overline{h} \} \]

so that \( \varphi(F) = 0 \) and \( h(\theta) = \overline{h} \forall \theta \in F^c \). Furthermore, by (5.72)

\[ \alpha(\xi \mid \theta) \gamma(\xi \mid \theta)p(\theta \mid Y) = \alpha(\theta \mid \xi)\gamma(\theta \mid \xi)p(\xi \mid Y). \]

Therefore, by the boundedness of \( \gamma \) and since by definition \( \alpha(\cdot \mid \cdot) \leq 1 \), it holds that for
\[ \theta \in A^+ \]

\[
\int_F \frac{1}{p(\theta \mid Y)} \alpha(\theta \mid \xi) \gamma(\theta \mid \xi) \, d\mu(\xi) \\
\leq \frac{\kappa}{p(\theta \mid Y)} \int_F p(\xi \mid Y) \, d\mu(\xi) = \kappa \frac{\varphi(F)}{p(\theta \mid Y)} = 0 \quad (5.93)
\]

where the restriction of \( \theta \in A^+ \) is necessary due to the division by \( p(\theta \mid Y) \) above.

Therefore, since \( h \) is bounded and since \( \alpha(\xi \mid \theta), \gamma(\xi \mid \theta) \geq 0 \), then \( \forall \theta \in A^+ \)

\[
\int_F \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) h(\xi) \, d\mu(\xi) = 0. \quad (5.94)
\]

Furthermore, via (5.66)

\[
K(\xi \mid \theta) = \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) I_{\mathcal{X}_\theta}(\xi) + r(\theta) \delta(\xi - \theta) \quad (5.95)
\]

so that by the defining property (3.83) of a harmonic function, for \( \theta \in A^+ \),

\[
h(\theta) = \int_{\mathcal{X}} K(\xi \mid \theta) h(\xi) \, d\mu(\xi) \\
= \int_F \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) I_{\mathcal{X}_\theta}(\xi) h(\xi) \, d\mu(\xi) + \\
\int_{F^c} \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) I_{\mathcal{X}_\theta}(\xi) h(\xi) \, d\mu(\xi) + r(\theta) h(\theta) \\
= 0 + \overline{h} \int_{\mathcal{X}_\theta} \alpha(\xi \mid \theta) \gamma(\xi \mid \theta) \, d\mu(\xi) + r(\theta) h(\theta) \\
= \overline{h} [1 - r(\theta)] + r(\theta) h(\theta).
\]

Therefore

\[
[1 - r(\theta)] [\overline{h} - h(\theta)] = 0, \quad \forall \theta \in A^+ \quad (5.96)
\]

and hence since \( P(A \mid \theta) \) is \( \varphi \)-irreducible, then \( r(\theta) < 1 \forall \theta \in \mathcal{X} \), so that

\[
h(\theta) = \overline{h} \forall \theta \in A^+. \quad (5.97)
\]

Finally, by the design of the algorithm, it is initialised with \( \theta \in A^+ \) and all accepted proposals lie in \( A^+ \). Therefore, the transition distribution must satisfy

\[
1 = \int_{A^+} P(d\xi \mid \theta), \quad 0 = \int_{A^+} P(d\xi \mid \theta) \quad (5.98)
\]
5.8. Convergence Rate

for any \( \theta \). Therefore, for \( \theta \not\in A^+ \) and again by the boundedness of \( h \)

\[
h(\theta) = \int_{A^+} h(\xi) P(d\xi | \theta) + \int_{A^+} h(\xi) P(d\xi | \theta) = \overline{h} + 0. \tag{5.99}
\]

Therefore, if \( h(\xi) \) is harmonic to \( P(d\xi | \theta) \), then it is a constant \( h(\xi) = \overline{h} \) for all \( \xi \in X \). Hence as discussed in Section 3.3, by [46, Theorem 2], the Markov chain \( P(\xi | \theta) \) is Harris recurrent. Application of Theorems 3.3.5 and 3.3.6 then completes the proof. \( \square \)

Therefore, with the choice \( f(\theta) = I_{f^{-1}(A)}(\theta) \) being the indicator function for \( f(\theta) \in A \) with \( A \) being an arbitrary (measurable) subset of the range of \( f \), the above theorem establishes that the sample histogram is a strongly consistent estimator of the underlying true posterior density. That is

\[
\lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} I_{f^{-1}(A)}(\theta_k) = p(f(\theta) \in A | Y) \tag{5.100}
\]

with probability one. In particular, if \( f(\theta) = I_{\theta_i \in A}(\theta) \) so that \( f \) determines the presence of only the \( i \)th element of \( \theta \) in a set \( A \), then (5.100) can be used to compute the posterior marginal density \( p(\theta_i | Y) \) which, as explained in Section 5.4, is generally intractable by any other means.

Furthermore if a point estimate of \( \theta \) is required, then since the posterior expectation \( E\{\theta | Y\} \) is the minimum variance estimate [5], it is a reasonable choice. Again, via Theorem 5.7.2, this can be estimated in a strongly consistent fashion with the choice \( f(\theta) = \theta \) to provide

\[
\lim_{M \to \infty} \frac{1}{M} \sum_{k=1}^{M} \theta_k = E\{\theta | Y\} \tag{5.101}
\]

with probability one.

5.8 Convergence Rate

While the proof of convergence of the MCMC chain to the required density is an important step, the rate at which that convergence occurs is also relevant. If convergence does not occur within a practical number of iterations, then the eventual convergence is of little value.

For this purpose, it is necessary to strengthen the assumption of \( \Theta \subseteq \Gamma \), to also include a bound on the proposal \( \gamma(\cdot) \), as specified in Assumption 5.8.1.

**Assumption 5.8.1.** There exists some \( \epsilon > 0 \) such that

\[
\gamma(\xi | \theta) > \epsilon p(\xi | Y) \quad \forall \theta \in \Theta. \tag{5.102}
\]

This is not a strong requirement. For example, in the simple scalar case where \( p(\theta |
Y) = \mathcal{N}(0, \sigma^2_\theta) \text{ and } \gamma(\xi \mid \theta) = \mathcal{N}(0, \theta^2_\xi), \text{ then a straightforward calculation establishes }
\text{that when } \sigma^2_\xi > \sigma^2_\theta, \text{ the choice of } \epsilon < \frac{\sigma^2_\theta}{\sigma^2_\xi} \text{ leads to (5.102) holding. Similarly in practice, this condition is readily met if a Gaussian random walk is employed.}

When the design of the algorithm proposal density is such that strengthened requirement (5.102) does hold, then an exponential bound may be established on the distributional convergence first addressed in Theorem 5.7.1.

**Theorem 5.8.1.** Under the Assumptions 5.7.1–5.7.3, and 5.8.1,
\[
\sup_{A \in \sigma(X)} |\mathbf{P}^n(A \mid \theta_0) - \varphi(A)| \leq (1 - \epsilon)^n
\]
for any \(\theta_0 \in \Theta\).

**Proof.** This proof follows the arguments of [28, 41], as applied to Algorithm 5.5.1. Define initially the sets
\[
R_\theta \triangleq \left\{ \xi \in \mathcal{X} : \frac{p(\xi \mid Y)}{p(\theta \mid Y)} \cdot \frac{\gamma(\theta \mid \xi)}{\gamma(\xi \mid \theta)} < 1 \right\}, \quad T_\theta \triangleq \mathcal{X} \setminus R_\theta
\]
corresponding to the regions to which a proposal from \(\theta\) will be accepted with probability of less than one. Then for any \(\theta \in \Theta\), and with \(I\) denoting the indicator function (5.51)
\[
K(\xi \mid \theta) \geq \alpha(\xi \mid \theta) \gamma(\xi \mid \theta)
= \frac{p(\xi \mid Y)}{p(\theta \mid Y)} \gamma(\theta \mid \xi) I_{R_\theta}(\xi) + \gamma(\xi \mid \theta) I_{T_\theta}(\xi)
\geq \epsilon p(\xi \mid Y) I_{R_\theta}(\xi) + \epsilon p(\xi \mid Y) I_{T_\theta}(\xi) = \epsilon p(\xi \mid Y).
\]
Therefore,
\[
r(\xi \mid \theta) \triangleq \frac{K(\xi \mid \theta) - \epsilon p(\xi \mid Y)}{1 - \epsilon}
\]
is a valid probability density function. Making the transition density \(K(\xi \mid \theta)\) the subject of the above equation allows it to be written in a form split between \(p\) and \(r\), as
\[
K(\xi \mid \theta) = \epsilon p(\xi \mid Y) + (1 - \epsilon)r(\xi \mid \theta).
\]
This provides an alternate means for drawing realisations from the Markov chain with transition probability \(K(\xi \mid \theta)\). Namely
\[
\text{if } \delta_k \sim \text{Ber}(\epsilon) = 0 \text{ then }
\theta_{k+1} \sim r(\cdot \mid \theta_k)
\text{ else }
\theta_{k+1} \sim p(\cdot \mid Y).
\text{ end if}
where \( \text{Ber}(\epsilon) \) denotes a Bernoulli density that delivers a 1 with probability \( \epsilon \), and 0 with probability \( 1 - \epsilon \). Consider now an additional chain \( \{ \beta_k \} \), which is initialised by drawing from the stationary distribution of the chain

\[
\beta_0 \sim p(\cdot \mid Y)
\]  
(5.110)

and is propagated by the above alternate means for realising \( K(\xi \mid \theta) \). The two chains then evolve according to

\[
\begin{align*}
\text{if} \quad & \delta_k \sim \text{Ber}(\epsilon) = 0 \\
\theta_{k+1} & \sim r(\cdot \mid \theta_k), \quad \beta_{k+1} \sim r(\cdot \mid \beta_k) \\
\text{else} \quad & \theta_{k+1} = \beta_{k+1} \sim p(\cdot \mid Y).
\end{align*}
\]

While these algorithms are independently initialised, once the algorithm reaches a \( T = k \) such that \( \theta_{k+1} = \beta_{k+1} \sim p(\cdot \mid Y) \), then all future draws are made so that \( \theta_k = \beta_k \) is preserved.

The essential point is that since the chain \( \{ \beta_k \} \) is initialised from the stationary distribution, then for any set \( A \in \mathcal{X} \) and any \( k \geq 0 \)

\[
p(\beta_k \in A) = \int_A p(\xi \mid Y) \, d\xi = \varphi(A)
\]  
(5.111)

and hence, since the drawings of \( \theta_{k+1} \) and \( \beta_{k+1} \) from \( r(\cdot \mid \cdot) \) above are independent

\[
|P^n(A \mid \theta_0) - \varphi(A)| = |p(\theta_n \in A) - p(\beta_n \in A)| = |p(\theta_n \in A, \theta_n = \beta_n) + p(\theta_n \in A, \theta_n \neq \beta_n) - p(\beta_n \in A, \theta_n = \beta_n)| \\
\leq \max \{ p(\theta_n \in A, \theta_n \neq \beta_n), p(\beta_n \in A, \theta_n \neq \beta_n) \} \\
\leq p(\theta_n \neq \beta_n) \leq p(T > n)
\]

where \( T \) is the ‘coupling time’ which is defined to be the random time at which \( \{ \theta_k \} \) and \( \{ \beta_k \} \) come together. However, since the drawings of \( \delta_k \) are independently \( \text{Ber}(\epsilon) \), then

\[
p(T = n) = \epsilon(1 - \epsilon)^{n-1}
\]  
(5.112)

so that

\[
P(T > n) = (1 - \epsilon)^n.
\]  
(5.113)
Clearly it is desirable that this relation holds for a relatively large value of ϵ. Note, however, that due to the constraint that both γ(ξ | θ) and p(ξ | Y) are unit area, then as ϵ is increased away from zero, the condition (5.102) can only be satisfied if the functional form of γ(ξ | θ) becomes close to that of p(ξ | Y). Therefore, (5.103) indicates that the closer the proposal γ(ξ | θ) is to the desired posterior p(ξ | Y), then the faster the exponential convergence of the distribution of realisations {θk} generated by Algorithm 5.5.1.

5.8.1 Convergence of Sample Averages

With the exponential distributional convergence of Algorithm 5.5.1 established, it remains to show the speed of convergence of the sample averages such as (5.49), (5.50) which are the main focus of the use of this algorithm. In the case of sample averages, an additional complication is that the convergence properties depend on the correlation between samples, which is present in the MCMC algorithms.

To address this, denote a centred (i.e. zero mean) version \( \tilde{f}_k \) of \( f(\theta_k) \) as

\[
\tilde{f}_k \triangleq f(\theta_k) - E(f(\theta_k) | Y)
\]

where \( f : X \to R \) is a function satisfying the conditions of Theorem 5.7.2 together with \( |f| < \infty \). These are constructed so that sample averages of such functions will converge to zero. Associated with function, define then the conditional asymptotic variance \( \sigma_f^2 \) of the estimate \( \hat{f}_M \) (see (5.49)) as

\[
\sigma_f^2 \triangleq \lim_{M \to \infty} \frac{1}{M} E \left\{ \left( \sum_{k=1}^{M} \tilde{f}_k \right)^2 \bigg| Y \right\}
\]

A quantification of the rate of convergence of \( \hat{f}_M \) to \( E\{f(\theta) | Y\} \) is then possible via the following result.

**Theorem 5.8.2.** Under the conditions of Theorem 5.8.1, the limit in (5.115) exists and is finite and for any \( \lambda < 1/2 \)

\[
\lim_{M \to \infty} \frac{1}{M^{1/\lambda}} \sum_{k=1}^{M} [f(\theta_k) - E\{f(\theta) | Y\}] = 0
\]

with probability one. Furthermore, if \( \sigma_f^2 \) defined by (5.115) is non-zero, then

\[
\sqrt{M} \left( \frac{1}{M} \sum_{k=1}^{M} f(\theta_k) - E\{f(\theta) | Y\} \right) \xrightarrow{D} \mathcal{N}(0, \sigma_f^2)
\]

as \( M \to \infty \).

**Proof.** The convergence of (5.115) and the asymptotic distributional result (5.117) follows
5.8. Convergence Rate

by direct application of Theorem 5.8.1 and then Theorem 3.3.7. Furthermore, since (5.115) holds, then for some $C < \infty$

$$\mathbb{E} \left\{ \left( \sum_{k=1}^{M} \tilde{f}_k \right)^2 \middle| Y \right\} < CM. \quad (5.118)$$

The result (5.116) is then obtained by the application of Theorem 2.1 of [37]. □

Considering again the definition in (5.49) for the sample average estimate,

$$\hat{f}_M \triangleq \frac{1}{M} \sum_{k=1}^{M} f(\theta_k). \quad (5.119)$$

For this estimate, the result (5.116) provides with probability one the convergence rate bound

$$\hat{f}_M - \mathbb{E} \{ f(\theta) \mid Y \} = o \left( \frac{1}{M\lambda} \right), \quad \lambda < 1/2. \quad (5.120)$$

For $M$ iterations of the Markov chain, it shows the convergence rate will at least be as good as an order $M^{-1/2}$. Note also, that by the distributional convergence in (5.117), that the asymptotic convergence will neither be faster than $M^{-1/2}$.

5.8.2 Choice of Proposal Distribution

An important observation from the analysis of the convergence rate is that for rapid convergence it’s necessary that functional form of the proposal $\gamma(\xi \mid \theta)$ approximate that of posterior density $p(\xi \mid Y)$. This opens the question to the choice and design of the proposal distribution to be used. Indeed, the main design variable in the implementation of Algorithm 5.5.1 is the choice of the proposal density $\gamma(\xi \mid \theta)$.

The choice of proposal density involves a tradeoff between convergence rate as a function of iterations, and computational complexity per iteration. For example, as the correlation between the realisations $\{\theta_k\}$ decreases, the constant term in (5.120) is reduced, improving the practical speed of convergence of the sample average $\hat{f}_M$ to $\mathbb{E}\{ f(\theta) \mid Y \}$. However, as correlation is minimised, algorithm complexity increases.

At one end of the scale, substituting the choice $\gamma(\xi \mid \theta) = p(\theta \mid Y)$ into (5.54) implies an acceptance probability $\alpha(\xi \mid \theta) = 1$, so that realisations of Algorithm 5.5.1 are independent realisations from $p(\theta \mid Y)$, and hence $\hat{f}_M$ converges maximally fast. This is clearly infeasible, since the entire premise of Markov chain Monte–Carlo method is that they are employed because sampling from $p(\theta \mid Y)$ is computationally impossible. Therefore, it is necessary to consider suboptimal choices for $\gamma(\xi \mid \theta)$ that are reasonable while not overly sacrificing convergence rate.

The following chapter will consider some practical consequences of choices for proposal functions in system identification applications.
5.9 Conclusion

MCMC methods offer value in a wide range of areas, including that of System Identification. While the estimation of model parameters is quite a mature field, with traditional approaches detailed knowledge of system estimates is only available for restricted model structures. The application of MCMC extends those results to systems that are nonlinear, have non-Gaussian noise, or need to be identified using only short data lengths.

While the Metropolis–Hastings algorithm presented is computationally intensive, it is shown to provably produce results that eventually converge to the true probability densities. The convergence analysis shows that the asymptotic rate of convergence is encouraging, however it remains for the following chapter to assess the computation time required in practice for realistic identification tasks.
Chapter 6

System Identification Implementation

While the previous chapter has demonstrated the theoretical utility of using MCMC algorithms for parameter estimation in system identification, much depends on the computational resources required for practical implementation. This chapter addresses this issue with case studies utilising MCMC methods in system identification applications.

The chapter commences by presenting the results of running the MCMC algorithm to identify the parameters of a simple first-order transfer function model. This model is used to demonstrate how accurate density function estimates can be achieved with minimal computational cost, but also how appropriate kernel smoothing of the estimates can dramatically reduce the number of MCMC iterations needed. Measurement of the convergence of the distribution also shows that the asymptotic convergence results anticipated from Chapter 5 are indeed met.

The benefits of the MCMC approach over existing methods are shown through the ability to accurately model the effects of short measurement data length and non-Gaussian noise models, with the results demonstrating that even in this simple model it has a significant impact.

In moving to a higher-order example, it is shown how an inappropriate implementation of MCMC methods can easily lead to excessive computation time. The choice of proposal distribution is key to effective algorithm design, and some intuition is built up via the first-order model, before presenting solutions for the higher-order models.

After considering the linear system models, the methods are extended to non-linear state space model structures. This is achieved through utilising the particle filter, but in doing so greatly increases the computational requirements of the algorithm. To overcome the greater computational cost, a parallel GPU implementation is presented, which can significantly speed up the execution time of the algorithms.
6.1 Simulation Study

6.1.1 System Model

To illustrate the application of these MCMC methods, this chapter initially considers the case of a linear and time invariant system model. In particular, the transfer function model structure, as presented in Section 5.2.3 is used. By choosing \( H(\theta) = 1 \) The model (5.17) is reduced to,

\[
y_t = G(q, \theta)u_t + \varepsilon_t, \quad G(q, \theta) = \frac{B(q, \theta)}{A(q, \theta)},
\]

where

\[
A(q, \theta) = 1 + a_1 q^{-1} + a_2 q^{-2} + \cdots + a_m q^{-m}, \quad (6.2)
\]

\[
B(q, \theta) = b_0 + b_1 q^{-1} + b_2 q^{-2} + \cdots + b_m q^{-m}, \quad (6.3)
\]

\[
\theta^T = [a_1, \cdots, a_m, b_0, \cdots, b_m]. \quad (6.4)
\]

![Figure 6.1: Transfer Function Model.](image)

For the purposes of use within the MCMC algorithms, it is important that there be an efficient method to evaluate the likelihood \( p(Y \mid \theta) \), as in (5.20). In this case, to evaluate (5.20), the required predictor is given by (5.32) as simply

\[
\hat{y}_{t\mid t-1}(\theta) = G(q, \theta)u_t. \quad (6.5)
\]

Computationally, this is easily achieved in the form,

\[
\hat{y}_{t\mid t-1} = b_0 u_t + b_1 u_{t-1} + \cdots + b_m u_{t-m} \\
- (a_1 \hat{y}_{t-1\mid t-2} + a_2 \hat{y}_{t-2\mid t-3} + \cdots + a_m \hat{y}_{t-m\mid t-(m+1)}) \quad (6.6)
\]

so that via (5.41) and (5.20) the posterior \( p(\theta \mid Y) \) can be evaluated for any value of \( \theta \) as

\[
p(\theta \mid Y) = k \cdot p(\theta) \prod_{t=1}^{N} p_\varepsilon(y_t - G(q, \theta)u_t). \quad (6.8)
\]

Here \( k \) is a constant independent of \( \theta \), that will not be required in any subsequent calculations, but is included in (6.8) to ensure it has unit total probability. An advantage
of the Metropolis–Hastings method is that the evaluation of the likelihood function only appears in the form of a ratio between the likelihood of two different parameter values. In these cases, the constant term \( k \) need not be evaluated.

A critical aspect of the output error model in (6.1) is the choice of the noise model \( p_\varepsilon(\cdot) \). In the examples that follow, two noise models are used for the additive noise \( \varepsilon_t \) on the measured samples. In each case, the variance of the noise is defined as

\[
\text{Var}\{\varepsilon_t\} = \sigma^2.
\]  

(6.9)

**Normal Distribution**

In the first case, the measurement noise is modelled as a zero-mean normal distribution, with

\[
\varepsilon_t \sim \mathcal{N}(0, \sigma^2).
\]  

(6.10)

Consequently, the probability density of the output error is given by

\[
p_\varepsilon(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{ -\frac{x^2}{2\sigma^2} \right\}.
\]  

(6.11)

When using this form in the evaluation of (6.8), the leading \( 1/\sqrt{2\pi\sigma} \) term may be absorbed into the constant, and the product of exponentials evaluated via the exponential of the sum,

\[
p(\theta \mid Y) = k^* \sigma^{-N} \cdot p(\theta) \exp \left\{ \sum_{t=1}^{N} -\frac{1}{2\sigma^2} (y_t - G(q, \theta)u_t)^2 \right\}.
\]  

(6.12)

This alternate formulation for the case of normally distributed measurement noise allows for more efficient computation of the likelihood function.

**Uniform Distribution**

Alternately, a uniform distribution is used to model the measurement noise. In this case, the limits are defined so that the variance of the distribution is again \( \sigma^2 \), as specified in (6.9). The resulting zero-mean distribution is then,

\[
\varepsilon_t \sim \mathcal{U}[-\sqrt{3}\sigma, \sqrt{3}\sigma].
\]  

(6.13)

In this case, all values within that range are equally probable, and so \( p_\varepsilon(\cdot) \) will simply be an indicator function,

\[
p_\varepsilon = \frac{1}{2\sqrt{3}\sigma} I_{[-\sqrt{3}\sigma, \sqrt{3}\sigma]}(x).
\]  

(6.14)
When using this uniformly distributed model, each of the terms in the product of (6.8) will be either a constant or zero. As the number of product terms \( N \) in (6.8) is the same for all parameter values, \( p_e \) takes on a value of either zero, or a constant, for all values of \( \theta \). Similarly, in what is to follow, the prior \( p(\theta) \) will be uniform, and hence not affect the computation in (6.8).

### 6.1.2 Metropolis–Hastings Algorithm

As described in Section 5.8.2, an important choice in the implementation of Metropolis–Hastings algorithms is that of the proposal distribution \( \gamma(\xi \mid \theta) \), from which the proposal values \( \xi_k \) are drawn. For the examples in this chapter, proposal distributions based on the random walk (5.57) are used, with

\[
\xi_k = \theta_{k-1} + \nu_k. \quad (6.15)
\]

For the initial examples, \( \nu_k \) will be distributed according to the simple Gaussian form,

\[
\nu_k \sim \mathcal{N}(0, \sigma^2 \nu I). \quad (6.16)
\]

Later sections will show motivation for using more complicated covariance structures in this distribution.

Using the proposal in (6.15), (6.16), at each iteration of the MCMC algorithm, the Markov state at previous iteration \( \theta_{k-1} \) is perturbed by a Gaussian distributed random amount \( \nu_k \). Recall, that as explained via (5.59), in this situation the acceptance probability \( \alpha(\xi \mid \theta) \) simplifies to the Metropolis form (5.60).

As a result, in the example we present here, the MCMC Algorithm 5.5.1 is implemented as follows.

**Algorithm 6.1.1. (MCMC for OE Model)**

1. Initialise \( \theta_0 \) at some value such that according to (6.8) the probability \( p(\theta_0 \mid Y) > 0 \), and set \( k = 1 \);

2. At iteration \( k \), generate a candidate value \( \xi_k \) computed according to the random walk proposal (6.15);

3. Substitute the \( \xi_k \) obtained in step 2 and the \( \theta_{k-1} \) from the previous iteration into (6.8) in order to compute the acceptance probability; viz.

\[
\alpha(\xi_k \mid \theta_{k-1}) = \min \left\{ 1, \frac{p(\xi_k \mid Y)}{p(\theta_{k-1} \mid Y)} \right\}; \quad (6.17)
\]

4. Generate a realisation \( z \sim \mathcal{U}[0, 1] \), where \( \mathcal{U}[0, 1] \) represents a uniform distribution on the interval \([0, 1]\).
5. Set $\theta_k = \xi_k$ if
\[ z < \alpha(\xi_k | \theta_{k-1}). \] (6.18)
otherwise set $\theta_k = \theta_{k-1}$.

6. Increment $k$ and return to step 2.

6.1.3 First-Order System Example

With the algorithm and model setting established, the first case studied is the simplest possible first order one of
\[ m_a = 1, \quad m_b = 0, \quad a_1 = -0.8, \quad b_0 = 0.2 \] (6.19)
with $\{\varepsilon_t\}$ a zero mean normally-distributed i.i.d. process of variance $\sigma^2 = \mathbb{E}\{\varepsilon_t^2\} = 0.01$.

It is then supposed that the available data from this system consists of only $N = 20$ samples of $\{y_t\}$ and $\{u_t\}$ being a sampled step response transiting $1 \rightarrow 0$ at the data record midpoint. The realisation used is illustrated in Figure 6.2, where the solid line is the noise free response, and the samples around this line are the noise corrupted data made available to the estimation algorithm.

![Figure 6.2: Example first order system response data used for identification.](image)

6.1.4 Marginal Density Estimation

Using the model and test scenario described in Section 6.1.3, a Metropolis algorithm has been used to estimate the marginal probability density functions $p(b_0 \mid Y)$ and $p(a_1 \mid Y)$ of the model parameters. The main design choice in using this algorithm is the choice of sampling density for the proposal density $\xi_k$. Here, the simple random walk in (6.15) is used, with a Gaussian perturbation described by (6.16). For this example, the standard
deviation of the walk was equal for both elements of the Markov state, and set to

$$\sigma_\nu = 0.025.$$  \hspace{1cm} (6.20)

This value was chosen after an adaption process, and consequently the first 3,000 samples from the Markov chain were discarded to avoid bias from the chosen starting point and initial choice of $\sigma_\nu$. The result after 10,000 iterations of the Markov chain is shown in Figure 6.3.

\[
\begin{align*}
P_{b_0}, \text{ at } 10^4 \text{ MCMC Iterations} \\
\begin{array}{c}
\text{MC\textsc{mc}} \\
\text{Integrated}
\end{array}
\end{align*}
\]

\[
\begin{array}{c}
\begin{array}{c}
\text{MC\textsc{mc}} \\
\text{Integrated}
\end{array}
\end{array}
\]

Figure 6.3: Marginal density estimates produced by Metropolis algorithm.

In order to determine the quality of this result, the marginal densities were computed via direct integration, as described by (5.46). As this is only a two-dimensional system, this is quite feasible numerically, with each integration evaluation being only over one dimension. Each point on the marginal probability density function is then evaluated by a simple integral.

\[
p(\theta^0 \mid Y) = \int p(\theta^0, \theta^1 \mid Y) d\theta^1 \hspace{1cm} (6.21)
\]
\[
p(\theta^1 \mid Y) = \int p(\theta^0, \theta^1 \mid Y) d\theta^0 \hspace{1cm} (6.22)
\]

This is done using the CUBA integration library[19], and the results from this method are shown as the trace on Figure 6.3. While the MCMC results are close to the true distribution, clearly more iterations of the algorithm are necessary in order to demonstrate a close match.

A result from running the Markov kernel for $10^5$ iterations is illustrated by the histograms in Figure 6.4. With the algorithm implemented in compiled C++, the computation time for this example of $10^5$ iterations is approximately 0.5 seconds using a single
6.2. Density Function Representation

As with all sampling methods, the construction of a probability density function from the MCMC algorithm involves depicting the statistical properties of a set of samples. A simple way of achieving this is to form histograms from the samples, as described in (5.50). This was the approach used for the MCMC result shown in Figure 6.3, where for each plot, (5.50) is evaluated 30 times for non-overlapping regions $A_i \subset \mathbb{R}^2$.

Figure 6.5 shows two further histogram estimates of just the $b_0$ parameter from the same dataset. Here the estimate is instead represented by a segmented line, and there are 100 bins for each histogram. The first plot is for $10^4$ MCMC iterations, corresponding to the result shown in Figure 6.3, while the second is for $10^5$ iterations as presented above in Figure 6.4. Clearly these estimates depart further from the desired posterior estimate.

At this number of iterations, clearly the MCMC method is now providing an estimate very close to the true marginal density.

Note that these marginal densities depend on the specific realisation of the noise in the measurements, and so for each of the tests described here, the same noise realisation is used, to allow meaningful comparisons. Similarly, the peaks in the probability densities do not necessarily coincide with the “true” parameters used to generate the data, in this case 0.2 and -0.8. Instead, alternate methods, such as the direct integration are necessary in order to find the true densities for the given noise realisation. In this case, the alignment with the direct integration result demonstrates that the MCMC algorithm is producing a result centred around the true maximum a-posterior estimation result.
on account of the higher histogram resolution, but the underlying shape of the density may still be inferred visually from the plot.

![Figure 6.5: Histogram estimates (100 bins) of \( b_0 \) for (a) \( 10^4 \) and (b) \( 10^5 \) iterations of MCMC.](image)

An alternative to the histogram is to use a kernel density estimator [50]. This estimate is calculated via the convolution on each of the samples drawn with a chosen kernel function \( K(\cdot) \),

\[
\hat{f}(x) = \int f(z)K(x - z)dz, \quad (6.23)
\]

where \( f(z) \) represents the samples to be smoothed. The function \( K(\cdot) \) is the kernel to be used, and may be an arbitrarily chosen function such that

\[
\int_{-\infty}^{\infty} K(x) \, dx = 1. \quad (6.24)
\]

For the set of \( M \) samples \( \{x_i\} \),

\[
f(z) = \frac{1}{M} \sum_{i=1}^{M} \delta(z - x_i), \quad (6.25)
\]

and so the kernel density may be computed simply as,

\[
\hat{f}(x) = \frac{1}{M} \sum_{i=1}^{M} K(x - x_i). \quad (6.26)
\]
While the function $K$ is arbitrary, the choice of kernel affects the nature of the estimate achieved, and this will dictate the choice. The two functions considered here are of a rectangular function, and the normal density function,

$$K_u(x) = \frac{1}{2\sqrt{3h}} I_{[-\sqrt{3h},\sqrt{3h}]}(x), \tag{6.27}$$

$$K_n(x) = \frac{1}{\sqrt{2\pi h^2}} \exp \left\{ -\frac{x^2}{2h^2} \right\}, \tag{6.28}$$

where $h$ is the kernel bandwidth. The bandwidth is an important tuning parameter to trade off between the variance of the estimator, and the bias introduced into the estimates. In controlling the width of the mass of the kernel, it determines the amount of smoothing to be applied to the density estimate. When the bandwidth is small there’s no smoothing, and hence considerable variance in the density estimate. A larger bandwidth increases the smoothing, but in doing so can introduce a bias into the estimate as the distribution of the samples is smeared.

At first glance, it may appear that a histogram estimate is simply a kernel density estimate with a rectangular kernel. However, there is a difference in the location of the rectangular functions used. For a histogram, all the kernel functions are aligned on a grid, whereas with a kernel estimator, the summed functions are centred on each sample. To achieve equivalence of the methods, and achieve a histogram result from a kernel density estimator, it’s necessary to first quantise the samples to a regular grid before applying the kernel. That grid has a spacing equal to the width of the rectangular kernel.

![Figure 6.6: Kernel smoothed estimate of $b_0$ for $10^4$ and $10^5$ iterations of MCMC.](image)

Again using the results from the MCMC algorithm applied to the model of (6.19), and the input-output data shown in Figure 6.2, A kernel density estimate of the $b_0$
parameter is shown in Figure 6.6. This shows estimates for both $10^4$ iterations, and the $10^5$ iterations used for the histogram estimates shown above in Figure 6.5. Clearly, by $10^5$ iterations, the estimate is very close to the true density.

6.3 Convergence Behaviour

While the estimated density functions have clearly shown convergent behaviour, it would be instructive to see whether the practical rate of convergence matches the expected asymptotic behaviour. This section now looks at this practically observed convergence.

6.3.1 Density norms

In order to measure the practical speed of convergence, it’s necessary to have a norm to measure the difference between two probability density functions. In Section 3.3.10, the Total Variation norm was presented as a means of doing this, and is commonly used in MCMC convergence analysis[35].

Alternatively, a popular choice of norm used to measure distribution error in the context of kernel density estimation is the Integrated Square Error[50]. This is calculated as,

$$E_I = \int \left( \hat{f}(x) - f(x) \right)^2 \, dx, \quad (6.29)$$

where $\hat{f}(\cdot)$ is the estimate, and $f(\cdot)$ the true underlying density. The overall performance of a kernel density estimator is typically measured in terms of this mean integrated square error, which takes the expectation over different sets of realisations of the random variable.

6.3.2 Convergence of First-Order System

The practical convergence rate of the MCMC methods may be investigated by measuring a norm of the error of the estimated distribution as a function of the number of MCMC iterations. Figure 6.7 shows results from doing this for the first-order system described in Section 6.1.3.

These results were obtained by separately running the MCMC algorithm 100 times and computing the integrated square error and total variation norm against the numerically integrated result. The solid trace shows the mean error across all of the 100 chains tested, while the two broken traces indicate the progress of two individual chains. In this case, a histogram estimate is used with a resolution of 100 bins, as illustrated in Figure 6.5.

Comparing the integrated square error on the left, and total variation norm on the right, both of these measures show the same features of the convergence behaviour of the chain. The principal difference is in the expansion of the y-axis due to the integrated
6.3. Convergence Behaviour

MCMC Convergence: Integrated Square Error

MCMC Convergence: Total Variation Norm

\[ \| p(b_0) - \pi(b_0) \|_{ise} \]

\[ \| p(b_0) - \pi(b_0) \|_{tv} \]

\[ 10^{-6} \quad 10^{-5} \quad 10^{-4} \quad 10^{-3} \quad 10^{-2} \quad 10^{-1} \quad 10^{0} \quad 10^{1} \quad 10^{2} \quad 10^{3} \quad 10^{4} \quad 10^{5} \quad 10^{6} \quad 10^{7} \quad 10^{8} \]

\[ \text{Iterations} \]

\[ \text{Mean 100 Chains} \]

\[ \text{Run 1} \quad \text{Run 2} \]

\[ 10^{1} \quad 10^{0} \quad 10^{-1} \quad 10^{-2} \quad 10^{-3} \quad 10^{-4} \quad 10^{-5} \quad 10^{-6} \]

\[ \| p(b_0) - \pi(b_0) \|_{tv} \]

\[ \| p(b_0) - \pi(b_0) \|_{ise} \]

\[ 10^{1} \quad 10^{0} \quad 10^{-1} \quad 10^{-2} \quad 10^{-3} \quad 10^{-4} \quad 10^{-5} \quad 10^{-6} \]

\[ \text{Iterations} \]

\[ \text{Mean 100 Chains} \]

\[ \text{Run 1} \quad \text{Run 2} \]

Figure 6.7: MCMC convergence rate for \( b_0 \) in 1st order system for different norms.

square error being a measure of the square of the error, as opposed to absolute error in the case of the total variation norm.

From \( 10^4 \) iterations onwards, the straight section of these plots demonstrates a convergence with increasing MCMC iterations to the true density, as expected from Theorem 5.7.2. The experimental convergence rate matches the geometric rate of

\[ \| p_M(\cdot) - \pi(\cdot) \|_{ise} < \frac{1}{M} K_{ls} \] (6.30)

for the integrated square error, or

\[ \| p_M(\cdot) - \pi(\cdot) \|_{tv} < \frac{1}{\sqrt{M}} K_{lv} \] (6.31)

as expressed in terms of the total variation norm, for constants \( K_{ls} \) and \( K_{lv} \) respectively. This demonstrates good practical convergence properties, and offers encouragement that this application of the Metropolis algorithm appears well-behaved.

The anomaly in the earlier section of the plots is a consequence of the so-called “burn-in” of the algorithm. A bias is introduced into the result of the initial conditions of the algorithm, and so some of the earlier samples are discarded. In these plots, the first one-third of the samples are discarded, up to a maximum of 3,000 samples. The progressive removal of these biased samples from the sample averages as the total number of iterations increase, allows the error to reduce at a higher rate than simply adding the new samples.
6.3.3 Effect of Smoothing on Convergence

Figure 6.8 demonstrates the effect of applying kernel smoothing to the estimate, for varying kernel widths $h_r$. Here $h_r$ expresses the kernel width relative to $\hat{\sigma}$, the estimate of the standard deviation of the density to be smoothed. The kernel bandwidth $h$, as specified in (6.27)–(6.28), is given by

$$h = \hat{\sigma}h_r$$  \hspace{1cm} (6.32)

This choice to relate the kernel bandwidth to the variance of the estimate itself is based on the assumption that the feature size of the density function is approximately proportional to the standard deviation. This will be true for many unimodal densities, and in particular the ones currently under consideration, but will clearly not be the case for others, and in particular those that are multi-modal.

Figure 6.8(a) depicts the results of using a rectangular smoothing kernel, as specified in (6.27), while Figure 6.8(b) shows the equivalent results when using the kernel in (6.28) which is shaped to the probability density function of a normal distribution. While there is some slight difference between the behaviours of the two kernels, clearly this is not large, and is much less significant than the choice of the width of the kernel.

Convergence of Smoothed Estimates

It may be observed that when a very narrow smoothing window is used, such as in the solid trace in these plots, the density error approaches that of the histogram estimate above in Figure 6.7, for all numbers of iterations.

Conversely, for wide smoothing, there is significant departure from the behaviour previously observed. At lower numbers of iterations, when there is significant error in
the density estimate, the estimate benefits greatly from the presence of smoothing. At $10^4$ iterations, as depicted in Figure 6.6, the integrated square error for a smoothed estimate with $h_r = 0.2$ is similar to that of the unsmoothed estimate at $10^5$ iterations. This suggests that the use of kernel smoothing can potentially reduce the number of MCMC iterations required by an order of magnitude.

The penalty for smoothing the estimates is that the convolution with kernel function biases the estimated result. This is apparent in the $h_r = 0.5$ trace, where from $10^4$ iterations onward there is no further improvement in the estimate from more MCMC iterations. Similarly, for the other kernel widths an equivalent error floor is eventually reached. The optimal choice of smoothing width then must depend on the quality of the estimate that is available.

![Figure 6.9: Smoothed estimate of $b_0$ for $10^4$ iterations of MCMC with rectangular kernel.](image)

While the choice of kernel function has little effect on total variation norm of the estimates, there can still be a cosmetic difference in the appearance of the density functions. Consider the comparison between the density estimates shown for $10^4$ iterations of the MCMC algorithm in Figure 6.6 and Figure 6.9. Both use the same bandwidth of $h_r = 0.2$, but differ through using a Gaussian or rectangular function respectively. While they both have a similar error in the distribution, the Gaussian kernel provides for a smoother appearance.

A potential advantage here of the rectangular function is that it may provide a better indication of the true quality of the estimate. In Figure 6.9, it is quite clear that the irregular nature of the function is due to inadequate sampling, whereas with the Gaussian kernel, as shown in Figure 6.6, it’s more likely to be considered to be representing the true density.
6.4 Comparing the Asymptotic Normal Estimate

A principal motivation for applying MCMC methods to system identification problems is that they provide a measure of error bounds on the estimated parameters. By giving the ability to obtain a full marginal probability density function for each parameter, the uncertainty in the estimate can be well understood.

In Section 5.3.4 an alternate approach was presented to provide error bounds on parameter estimates from prediction error methods. For the case where the measurement noise is modelled as Gaussian, and in the limit where the number of measurements approaches infinity, the marginal probability of each parameter follows a Gaussian density with known mean and variance. The distribution of this error was presented in (5.37) and (5.34), to be expressed as,

$$\hat{\theta}_N^i \sim \mathcal{N}\left(\theta^*_i, \frac{1}{N}[P]_{i,i}\right),$$

for the prediction error estimate of $\theta^*_i$ and with $P$ the inverse of the hessian of the likelihood function, as defined in (5.34).

While this result presents an asymptotic limit in $N$, the number of data points, it may still be applied to shorter data records. Figure 6.10 shows a representation of the asymptotic results for the first-order system described in Section 6.1.3.

![Figure 6.10: MAP marginal density estimate compared to asymptotic normal estimate.](image)

While there is a difference between the asymptotic normal estimate and the true posterior density, as computed via the MCMC algorithm, these are very close for this system example. This serves to validate the MCMC approach, but also shows that it would be unnecessary in this simple model structure, as the asymptotic results are adequate for the purpose.
6.5 Non-Gaussian Noise

A significant benefit of the MCMC methods for system identification is that the model structure may be very general. While the initial example presented in this chapter is a linear system with Gaussian noise assumptions, these conditions are by no means necessary for the operation of Algorithm 6.1.1. To illustrate this, the existing first-order example will be recomputed under different assumptions regarding the nature of the noise.

6.5.1 Uniform Measurement Noise

Considering again the system model as defined in (6.1), we now consider the case where the measurement noise is considered to be uniformly distributed over a region. Essentially this means that we are aware of the maximum error in the measurement, but all errors within that range are equally probable. This is a situation that may occur when there is an error due to quantisation in a system.

The distribution of the noise term is as expressed in (6.13), as

$$\varepsilon_t \sim U[-\sqrt{3}\sigma, \sqrt{3}\sigma].$$

(6.34)

The $\sqrt{3}$ terms appear so that the variance of this distribution is $\sigma^2$, to be comparable to the Gaussian noise case. The change required to the Metropolis algorithm (Algorithm 6.1.1), appears in step 3, in the computation of the acceptance probability. This, in turn, is computed via (6.8). The dependence on the noise distribution is in the probability density $p_{\varepsilon}(\cdot)$, which in the uniform case is given by (6.14), as

$$p_{\varepsilon} = \frac{1}{2\sqrt{3}\sigma}I_{[-\sqrt{3}\sigma, \sqrt{3}\sigma]}(x).$$

(6.35)

The acceptance probability is computed in (6.17) as a ratio of two of these terms. Where the assumed noise variance $\sigma^2$ is kept constant, this acceptance ratio will take on only the values of zero or one. Essentially a new proposal will always be accepted if the calculated noise residuals from the model fall within the specified bounds. Conversely it will always be rejected if it doesn’t. Consequently, it is no longer necessary to draw a number from the uniform distribution in Step 4 of Algorithm 6.1.1.

The solid trace in Figure 6.11 shows the marginal probability density functions for the $b_0$ and $a_1$ parameters. A comparison with the equivalent plot for the Gaussian case in Figure 6.10 shows that the noise model can make a significant difference to the estimated parameters. Note that the same input-output data is used as in the previous case.

As the noise is non-Gaussian, the asymptotic normal estimate described in Section 5.3.4 cannot be computed in this case. However, an approximation could be taken by assuming that the noise were Gaussian, with the same variance. The dashed trace shows this estimate, which clearly deviates significantly from the true density.
Figure 6.11: Marginal density for parameters in the uniform noise model compared to asymptotic normal estimate.

Figure 6.12: Estimates of $b_0$ for $10^4$ and $10^5$ iterations of MCMC for the uniform noise case.

A natural question to ask of these results is whether this change of noise model affects the convergence rate of the algorithm. Figure 6.12 shows estimates available after $10^4$ and $10^5$ iterations, equivalent to those for the Gaussian case in Figure 6.6. The “Gold Standard” trace in the plots is obtained by $10^8$ iterations of MCMC. Given the convergence behaviour of the chain, we can be quite confident that by this number of iterations the samples are a close representation of the true distribution. Subjectively,
the quality of the estimates at $10^4$ and $10^5$ iterations is quite equivalent to that of the Gaussian measurement noise case.

Objective convergence performance is shown in Figure 6.13. Here the total variation norm of the distribution error is shown as a function of iterations. Again, this is quite similar to the performance from Gaussian noise, as shown in Figure 6.8(a).

![Convergence of Smoothed Estimates](image)

Figure 6.13: MCMC convergence rate for $b_0$ in 1st order system with uniform noise model, and rectangular smoothing kernels.

### 6.5.2 Truncated Gaussian Measurement Noise

As another example of a measurement noise model, this section will consider a Gaussian distribution with truncated tails. For this case, the measurement noise in (6.1) is assumed to have a probability density function of,

$$p_\varepsilon = \begin{cases} 
\frac{K_n}{\sqrt{2\pi\sigma^2}} \exp \left\{ -\frac{x^2}{2\sigma^2} \right\}; & \text{if } |x| < 2\sigma \\
0; & \text{otherwise.} 
\end{cases} \quad (6.36)$$

Here $K_n$ is a normalising constant adjusted to ensure that $p_\varepsilon$ integrates to unity. This type of noise model may be appropriate when hard bounds are known on the measurement error, but where it is known that the distribution of $\{\varepsilon_t\}$ is more weighted to the centre than a uniform distribution.

The resulting marginal probability density functions are shown in Figure 6.14. As expected, these show characteristics that are somewhere between the uniform and Gaussian examples. Again, the asymptotic normal result is shown in comparison, where the truncation of the distribution tails is ignored.
6.6 Measurement Noise Estimation

To this point, it has been assumed that the variance of the measurement noise is known at the time of the parameter estimation. However, this may not be known, and instead may itself be a parameter in need of estimation. Here we consider the joint estimation of the model parameters and noise variance.

In order to accommodate the estimation of measurement noise, the model structure presented in Section 6.1.1 needs to be extended. In particular, the parameter vector in (6.4) is augmented to include a term $c_0$ to parameterise the noise.

$$\theta^T = [a_1, \cdots, a_m, b_0, \cdots, b_m, c_0].$$  \hfill (6.37)

The noise estimation process may then be achieved by computing the marginal probability of $p(c_0 \mid Y)$. This is inherently no different to the estimation of any of the other components of $\theta$, provided the likelihood function $p(Y \mid \theta)$ may still be computed. To achieve this, (6.8) is altered to parameterise the probability density according to the parameter vector $\theta$.

$$p(\theta \mid Y) = k \cdot p(\theta) \prod_{t=1}^N p_\epsilon^\theta(y_t - G(q, \theta)u_t).$$  \hfill (6.38)

The probability density function of the residual remains that of a normal distribution
with a variance $\sigma^2(\theta)$,
\[
p_{\epsilon^\theta} = \frac{1}{\sqrt{2\pi \sigma^2(\theta)}} \exp\left\{-\frac{x^2}{2\sigma^2(\theta)}\right\}.
\]  
(6.39)

The standard deviation $\sigma(\theta)$ may simply be assigned to the parameter $c_0$, however in practice it has been beneficial to apply the mapping,
\[
\sigma(\theta) = e^{c_0}.
\]  
(6.40)

This allows the $c_0$ parameter to be proportional to the logarithm of the noise variance. This tends to be more compatible with an additive random walk proposal, as a fixed proposal variance will now move in geometric ratios, and so be able to accommodate the exploration of a greater dynamic range. This is particularly useful in the early adaption and burn-in of the algorithm.

The other effect of adjusting the parameterisation relates to the prior probability $p(\sigma)$. In the absence of providing an adjustment term, a uniform prior is assumed. With the logarithmic parameterisation, this will result in a uniform prior on the logarithm of the variance, rather than a uniform prior on the square root of the variance. While it becomes a philosophical question which of these is the “correct” approach, the logarithmic assumption is good at expressing a lack of knowledge of the order of magnitude of the noise.

Both the prior distribution and the probability density function (6.39) may be altered, rather than choosing the logarithmic parameterisation, but the convenience of the parameterisation approach is that the noise variance parameter behaves more like the other parameters in $\theta$, and so may be treated more uniformly in the algorithm implementation.

### 6.6.1 Gaussian Example

The result of extending the Metropolis algorithm to include the estimation of the measurement noise variance is shown in Figure 6.15. This plot shows the marginal density functions for the $b_0$ and $a_1$ parameters, together with $c_0$ implemented as
\[
c_0 = \log \frac{\sigma}{\sigma_0}.
\]  
(6.41)

Consequently, a value of 0 for $c_0$ represents an estimate equal to the standard deviation of the true system used to generate the measurement data, while $c_0 = 0.4$ results in an estimate of $\sigma$ that is 1.5 times that of true system.

As the assumptions on the noise have changed, we would expect a change in the density functions for the $b_0$ and $a_1$ parameters. Figure 6.15 shows this to be the case, but the difference is quite small. Primarily, the increased possible variability of the noise has allowed a slightly higher probability to the tails of the distributions.

In including the noise estimation into the Metropolis algorithm, the sampling has
moved from a two dimensional space into three dimensions. The expectation would be that due to the increased complexity of the target distribution the convergence time may increase. However, Figure 6.16 shows a comparison of the convergence behaviour of chains that include the noise estimation against those without for the $b_0$ parameter. It actually shows a slight performance improvement for low numbers of iterations, but overall is essentially the same.

While in some cases there may be a penalty for estimating the noise, in this scenario, it essentially comes for free when using the Metropolis algorithm. The parameter estimation performance remains the same.

\begin{figure}[h]
\centering
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{b0_density}
\caption{Marginal probability densities with output noise estimation.}
\end{subfigure}
\hfill
\begin{subfigure}{0.45\textwidth}
\includegraphics[width=\textwidth]{a1_density}
\end{subfigure}
\caption{Marginal probability densities with output noise estimation.}
\end{figure}
6.6.2 Uniform Example

The extension of $\theta$ to include noise estimation was also applied to the uniform noise example detailed in Section 6.5.1. In this case, the same parameterisation is used, with

$$c_0 = \log \frac{\sigma}{\sigma_0}.$$  \hfill (6.42)

The value of $\sigma$ determines the bounds of the uniform distribution, according to (6.14). The results from this estimation are shown in Figure 6.17. For the uniform noise case, there is now a more considerable change in the marginal density functions as a result of removing the known noise assumption.

It may be noted by the $c_0$ parameter that there is a higher certainty on the estimate of the noise, with now the likely margin of error being only about 10%. There are now also significantly greater tails on the $b_0$ and $a_1$ marginal densities, due to including the possibility that the noise is larger than that of the true system.

The convergence properties were again profiled, and are included in Figure 6.18. As with the Gaussian noise case, in this example the inclusion of noise estimation does not increase the number of MCMC iterations required.

6.7 MCMC Error Bounds

A potential criticism of MCMC methods is that while asymptotic limits may be met, and convergence may be good on average, these are not guarantees for each individual run of an MCMC chain. For this, it is encouraging to have results that hold true for every possible run of the algorithm. The performance above has been illustrated with
the marginal density functions of a single run of the MCMC algorithm. In Figure 6.19, bounds are illustrated for over 100 separate runs of the MCMC algorithm, each to $10^4$ iterations.

While it is clear that some of the runs have departed further from the true density than the one previously illustrated, none are considerably worse. Even at this relatively small number of iterations, all of the chains have provided quite usable estimates of the marginal densities of the parameters.

### 6.8 Proposal

A critical part of a Metropolis-Hastings algorithm implementation is the choice of the proposal distribution, defined in (5.53). One of the benefits of the Metropolis-Hastings algorithms is that this proposal distribution is rather arbitrary, so the implementer has
Figure 6.18: Convergence of marginal probability densities with output noise estimation.

Figure 6.19: Experimental error bounds on MCMC estimates.

the choice of using a distribution that is easy to sample from. However, the choice of distribution also has a significant effect on the convergence rate of the algorithm, so while there are only mild constraints on the proposal to ensure theoretical convergence, the requirements are much stronger for timely practical convergence.

In its most general form, the proposed new state vector \( \xi_{k+1} \) may be drawn from an arbitrary density that may depend on present state \( \theta_k \),

\[
\xi_{k+1} \sim \gamma(\cdot \mid \theta_k).
\]  

(6.43)
If \( \gamma \) is not conditioned on \( \theta_k \), it is the special case of the Independence Metropolis-Hastings algorithm. However, in most cases there is an advantage in proposing new \( \xi \) near to the existing \( \theta_k \) in some way. With a smooth likelihood function, it gives a greater chance of selecting a proposed point with higher likelihood, and hence providing control over the acceptance ratio.

6.8.1 Random Walk

The random walk is one of the simplest proposal densities to be used on smooth likelihood functions. It draws new samples from the locality of the existing state of the Markov Chain. It’s defined as,

\[
\xi_{k+1} = \theta_k + \nu_k, \tag{6.44}
\]

where \( \nu_k \) is drawn from an arbitrary distribution, which is typically centered on the origin. A natural choice for this distribution is to use a Gaussian, with

\[
\nu_k \sim \mathcal{N}(0, \sigma^2 I). \tag{6.45}
\]

The random walk is an efficient choice for a proposal density over smooth unimodal target densities, because proposal samples are usually generated in regions of high likelihood. A random walk with a small variance is likely to give proposals with a similar likelihood to that of the present state of the Markov chain. Consequently, the acceptance probability (5.54) will approach

\[
\alpha(\theta_{k-1} | \theta_{k-1}) = \min \left\{ 1, \frac{p(\theta_{k-1} | Y)}{p(\theta_{k-1} | Y)} \right\} = 1. \tag{6.46}
\]

The acceptance ratio of a chain needs to be sufficiently high that samples drawn from the chain are not highly correlated due to there being no acceptance of new values. However, when the walk variance is too small, even though the chain state changes regularly, the samples are again highly correlated because of the spacial proximity of the successive states.

Figure 6.20 compares these different cases in an MCMC sampler of \( U(-1, 1) \). In (a), the walk variance is small, and consequently it is difficult for the chain to reach the range of values in the target distribution. The larger walk variance in (b) overcomes this, while in (c) it is too large, and results in most proposals being rejected.

6.8.2 Random Walk Variance

An advantage of the random walk proposal is that its implementation requires very little knowledge of the nature of the target distribution. Where a Gaussian is used for the random walk density, there is only one parameter to tune, that being the variance of that Gaussian.
We would expect that for any given density function, there would be an optimal walk variance that would minimise the number of MCMC iterations required in order to reach a given expected error in the estimated distribution.

In the case of the first order system, Figure 6.21 shows the effect of varying the walk variance on the convergence rate of the algorithm. In this plot, the number of iterations required to reach an error of total variation norm $10^{-1}$ is shown as a function of proposal walk variance.

There is clearly an optimum walk variance to give the fastest distribution convergence rate. Where the walk variance is very small, the movement through the target density
support is too slow as the movement steps are so small. Where the walk variance is very large, the progress of the algorithm is again slow, but this time because the average acceptance probability is very low, and the Markov state will typically remain stationary for many iterations at a time. Consequently, the samples from that chain are also strongly correlated.

An important MCMC design problem becomes how to select the optimum variance for the random walk proposal. In this example it was necessary to run the chain for many iterations at each proposal walk variance to compare the difference, and clearly it would defeat the purpose of the optimisation in order to have to do this on every example. Consequently, it is desirable to find methods that will estimate a good proposal variance before or during the operation of the MCMC algorithm.

### 6.8.3 Adaption

Proposal adaption methods observe the behaviour of the Markov chain over an iterative process. Measures of the behaviour of the chain are used to predict whether the proposal variance is larger or smaller than the optimal value. The key measure of performance here is normally the average acceptance probability, or acceptance ratio.

Under certain assumptions, an optimal value for the acceptance ratio may be computed. Where each of the components $\theta^i$ of $\theta$, when conditioned on $Y$, are independent, then there is theoretical analysis published [42] that establishes an optimal acceptance ratio, $\alpha^\star$. Under those conditions, a proposal distribution that results in an acceptance ratio of $\alpha^\star = 0.234$ provides the fastest convergence rate, as measured by the total variation norm.

When the acceptance probability is too low on average, then an adaption algorithm will decrease the walk variance, while if it’s too high, it will increase the walk variance, to allow greater movement through the support.

However, methods which use the past performance of the Markov chain to adapt the proposal distribution violate the Markov property of the chain in Algorithm 5.5.1, where the transition kernel is

$$K(\theta_k \mid \theta_{k-1}).$$

With an adaptive proposal, not only the present state of the chain is used in the calculation of the current proposal density, but also information from past states of the chain. It depends greatly on the specific adaption implementation as to whether or not this would greatly affect the accuracy of the results of the MCMC method.

A common example where this bias is introduced is when the initial proposal variance is very small for a chain that uses a random walk proposal. This will result in the early samples being clustered around the starting point. When the proposal variance is later adapted higher, a bias will exist from the early samples from the chain prior to the
adaption. The sample averages collected over the complete run of the chain will be a combination of the average over those initial closely clustered samples, and over the later less correlated samples. This could be considered an element of “burn-in” of the chain, and solved by discarding the early samples drawn. Essentially this involves not using the samples collected while the Markov property is being violated.

A contrived case of this would be to use the data from the sampler in Figure 6.20, and view the transition between (a) and (b) as a proposal adaption. A histogram of the combined data from (a) and (b) appears in Figure 6.22, showing the type of error that can be produced.

![Figure 6.22: Example of density errors from proposal adaption](image)

This should represent a uniform density between -1 and 1, but due to the inclusion of the samples with inappropriate algorithm tuning, there is an additional spike around the starting point of the chain. This could be avoided by discarding the initial samples obtained with the proposal adaption shown in (a).

### 6.8.4 Basic Acceptance Ratio Adaption

A basic proposal adaption scheme is now presented. This scheme involves monitoring the acceptance ratio observed from the algorithm, and increases or decreases the proposal variance to attempt to reach the target acceptance ratio.

Let $\alpha_p^k \in \{0, 1\}$ represent the acceptance of the proposal $\xi_k$ in iteration $k$ of the MCMC algorithm. A value of 0 indicates rejection, and 1 an acceptance of the proposal. For an adaption window of length $N_a$ iterations, the measured acceptance ratio is

$$\alpha_m(N_a, k) = \frac{1}{N_a} \sum_{k-N_a}^{k-1} \alpha_p^k.$$  \hspace{1cm} (6.48)

Partition the iterations of the MCMC algorithm into a set of $M_p$ adaption windows, comprised of $N_a(i)$ iterations for window $i$, as shown in Figure 6.23. The partition lengths
may be defined as,

\[ N_{a}(i) = \begin{cases} 
N_{a}^0 & i < M_c \\
K_pN_{a}(i - 1) & i \geq M_c 
\end{cases} \]  

(6.49)

for algorithm tuning parameters \( N_{a}^0 \), \( K_p \) and \( M_c \). Some typical values used in trial simulations are shown in Table 6.1. The increasing window length over time is used to ensure approximation of the Markov property of the chain as the number of iterations increase. As \( i \) increases, the parameters of the proposal depend progressively less on the recent states of the chain. This is an example of a \textit{vanishing adaption} approach[3].

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( N_{a}^0 )</td>
<td>200</td>
</tr>
<tr>
<td>( M_c )</td>
<td>25</td>
</tr>
<tr>
<td>( K_p )</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 6.1: Example values for adaption partition.

For each adaption window \( i \) in this partition, the acceptance ratio \( \alpha_m(i) \) may be calculated by applying (6.48) across the duration of the window. Based on measured acceptance ratios, each window is allocated a random walk variance \( \sigma_\nu(i)^2 \). The initial value is set to some arbitrary \( \sigma_\nu(0) = \sigma_\nu^0 \), and for \( i > 0 \),

\[ \sigma_\nu(i) = \begin{cases} 
\frac{1}{K_s}\sigma_\nu(i) & \alpha_m(i - 1) < \frac{1}{K_h}\alpha_m^* \\
K_s\sigma_\nu(i) & \alpha_m(i - 1) > K_h\alpha_m^* \\
\sigma_\nu(i) & \text{otherwise}, 
\end{cases} \]  

(6.50)

where \( \alpha_m^* \) is the target value for the proposal acceptance rate. The tuning constants \( K_h > 1 \) and \( K_s > 1 \) control the adaption speed, and typical values used in the simulation study are shown in Table 6.2.

Figures 6.24 and 6.25 demonstrate the operation of this adaptive method for the MCMC example in Section 6.1.3. The initial value of \( \sigma = 1 \) is clearly far too large, and initially the acceptance ratio is very low. As the adaption scheme takes effect, the proposal variance reduces to the point where the acceptance ratio matches the target acceptance. After this occurs, the proposal variance and the resulting acceptance ratio
Table 6.2: Example values for proposal adaption.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K_h$</td>
<td>1.15</td>
</tr>
<tr>
<td>$K_s$</td>
<td>1.20</td>
</tr>
<tr>
<td>$\sigma^0$</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Figure 6.24: Resulting random walk standard deviation from applying the adaptive proposal to the first order model.

Figure 6.25: Resulting acceptance ratio when applying the adaptive proposal to the first order model.

remain largely static.
6.8.5 Multi-Dimensional Case

The discussion to this point has been concerned primarily with univariate proposal random variables. However, the systems of interest are almost invariably multi-dimensional. Considering then, the random walk in (6.44)

\[ \xi_{k+1} = \theta_k + \nu_k. \]  

(6.51)

The perturbation \( \nu_k \) is then a vector, and there is considerable choice available for the distribution of this vector. We will initially restrict this to those drawn from a multivariate normal distribution with covariance \( \Sigma_\nu \Sigma_\nu^T \),

\[ \nu_k \sim \mathcal{N}(0, \Sigma_\nu \Sigma_\nu^T) \]  

(6.52)

\[ \sim \Sigma_\nu \mathcal{N}(0, I). \]  

(6.53)

In the simplest case, the covariance matrix \( \Sigma_\nu \) may be a scaled identity matrix \( \sigma^2 I \).

As in the scalar case, this requires the tuning of only one variance parameter.

For greater flexibility in tuning the proposal distribution, \( \Sigma_\nu \) can instead take on the form

\[
\Sigma_\nu = \begin{bmatrix}
\sigma_0 & 0 & 0 \\
0 & \sigma_1 & \\
0 & \ddots & \\
\end{bmatrix}
\]  

(6.54)

In this case, there is the capability to tune the proposal step size along each of the parameter dimensions. This improves the mixing of the chain in cases where the scaling of each of the parameters differs significantly.

Again using the first-order example system, the convergence rate of an MCMC sampler was evaluated as a function of the ratio \( \frac{\sigma_1}{\sigma_0} \). Subject to that fixed ratio, in each case an acceptance-rate based adaption was used to tune the random walk variance. Figure 6.26 shows that while in this system there is quite a wide tolerance in accommodating a range of values for \( \frac{\sigma_1}{\sigma_0} \), inappropriate scaling dramatically degrades the performance of the sampler.

In the general case of identifying model parameters, the relative scaling between different parameters is not known. As a result, it is necessary to find a means of calculating suitable values for the elements of \([\sigma_0, \sigma_1, \ldots, \sigma_n]\).

The challenge in moving to this richer proposal structure is in determining the parameters needed for the proposal. The advantage of having a single scalar walk variance parameter is that it is relatively easy to implement an algorithm to regulate the acceptance ratio of the chain. Where there is a vector of parameters, it’s more difficult to adaptively choose an appropriate scaling between the different parameters.
6.8. Proposal

6.8.6 Principal Axis Proposals

As expressed in (6.44), the random walk proposal is defined by

$$
\xi_{k+1} = \theta_k + \nu_k,
$$

(6.55)

where $\nu_k$ has been drawn from a suitable proposal distribution. In the scalar case, it was relatively easy to adjust the variance of the process producing $\nu_k$, in response to acceptance ratio. However, with multiple dimensions, it isn’t immediately clear which parameters in the noise covariance should be altered.

However, it may be noted that there is no requirement that $\{\nu_k\}$ be identically distributed, but only that the acceptance ratio be chosen so that detailed balance is preserved for the underlying Markov chain. This allows for the opportunity to alter the walk distribution over time to acquire acceptance probability information suitable for adaption of multi-variable proposals.

In this case, the random walk noise may be drawn with a time varying covariance $\Sigma_k\Sigma_k^T$

$$
\nu_k \sim \mathcal{N}(0, \Sigma_k\Sigma_k^T).
$$

(6.56)

For some $k$ in small subset of the total chain iterations, we may assign a covariance matrix with only a single non-zero element, located on the diagonal. For the first row,
this may be achieved with,

\[
\Sigma_k = \begin{bmatrix}
\sigma_{1,k} & 0 & \ldots & 0 \\
0 & 0 & \ldots & 0 \\
\vdots & \vdots & \ddots & \vdots \\
0 & 0 & \ldots & 0
\end{bmatrix}.
\]  

(6.57)

This will cause the proposal walk to vary only in one of the parameters of the Markov state vector. By independently observing the acceptance ratio of all such proposals, it’s possible to adapt each of the diagonal elements of \( \Sigma_k \) independently according to the proposal adaption scheme in Section 6.8.4. The combined diagonal form of \( \Sigma_k \) may then be used for the majority of the proposals.

As this method only requires the modification of the covariance matrix for the random walk, for sensible implementations the convergence properties of the algorithm are maintained. For the chain to remain irreducible, it is only necessary that all states remain reachable by the proposal. This can be assured by ensuring that for some \( k \), \( \Sigma_k \) is presented with all non-zero terms. Additionally, if the principal direction proposals are applied at random iterations of the algorithm, the Markov chain will necessarily remain aperiodic.

### 6.8.7 Rotating the Proposal Axes

The ability to individually alter the elements of the diagonal covariance matrix is important in designing a proposal to correctly balance between different elements of the proposal vector. However, it is still a significant constraint that the covariance be limited to only the diagonal form.

Considering the two dimensional case, a single scaled identity matrix, such as

\[
\Sigma_\nu = \sigma \begin{bmatrix}
1 & 0 \\
0 & 1
\end{bmatrix}
\]  

(6.58)

would constrain the probability density of \( \{\nu_k\} \) to have circular level curves. As seen above, this is unsuitable when the algorithm has to move much further distances in one direction than another. Then by adding independent scaling for each of the axes, we have

\[
\Sigma_\nu = \begin{bmatrix}
\sigma_1 & 0 \\
0 & \sigma_2
\end{bmatrix}.
\]  

(6.59)

This offers a better ability at matching different target probability density functions, and in Section 6.8.6 a proposal adaption scheme was presented for this form of proposal. This independent control of the diagonal terms expands the level curves of the distribution of \( \{\nu_k\} \) to an elliptical shape, but the principal axes of the ellipsoids must remain in the
direction of the unit vectors.

Consider the scatter plot of samples from an MCMC algorithm in Figure 6.27. The target density is of a heat transfer model with two parameters \( h_1 \) and \( h_2 \). In order to most effectively sample from this target distribution, it would be beneficial to have a proposal walk density that better matched the target density. In this case such a density could be achieved by rotating the ellipse axes.

![MCMC Scatter-Plot for h1, h2 Estimation](image)

Figure 6.27: Samples from example MCMC chain

By adding a rotation to (6.59), we have

\[
\Sigma_\nu = \begin{bmatrix}
\cos \phi & -\sin \phi \\
\sin \phi & \cos \phi
\end{bmatrix}
\begin{bmatrix}
\sigma_1 & 0 \\
0 & \sigma_2
\end{bmatrix}
\]

(6.60)

\[
= \begin{bmatrix}
\sigma_1 \cos \phi & -\sigma_2 \sin \phi \\
\sigma_1 \sin \phi & \sigma_2 \cos \phi
\end{bmatrix}
\]

(6.61)

Returning again to the first-order system example in Section 6.1.3, Figure 6.28 shows the effect of rotating the angle \( \phi \) for proposals derived from (6.61). For each of the angles of rotation \( \phi \), the MCMC algorithm was run with \( \sigma_1 \) and \( \sigma_2 \) adaptively chosen according to the algorithm in Section 6.8.6.

Figure 6.28 shows that for this system, convergence is fastest with a rotation of 0.8 radians on the axis of the proposal perturbation. In this particular example the speedup across the range of angles is not very large but is significant. Clearly, other target densities will exhibit a greater advantage from proposal alignment.

In this example, it would seem plausible to construct a proposal adaption scheme to modulate the \( \phi \) parameter in order to optimise the acceptance ratio. However, it is difficult to extend such schemes to higher dimensions, as in that setting there are many more “rotations” possible. For a system of \( n \) parameters, essentially there is the freedom to choose \( \Sigma_k \) to consist of \( n \) linearly independent basis vectors, each representing
a proposal axis direction. Adaptively tuning these vectors remains an open problem, but some options are considered later in this chapter in the context of identifying higher order systems.

6.9 Fourth Order Linear System

The first-order linear example that has been presented is useful for the illustration of the MCMC method, but practical uses of these methods are likely to involve a greater number of parameters. This section profiles the behaviour of the Metropolis estimation method for a fourth order model which involves the estimation of ten parameters, including the noise variance. With this system size, direct integration of marginal densities is not practical, and so stochastic methods become necessary in order to obtain results.

The discrete-time transfer function model in (6.1) is used, with

$$y_t = G(q, \theta)u_t + \varepsilon_t, \quad G(q, \theta) = \frac{B(q, \theta)}{A(q, \theta)},$$  \hspace{1cm} (6.62)

where

$$A(q, \theta) = 1 + a_1 q^{-1} + a_2 q^{-2} + a_3 q^{-3} + a_4 q^{-4},$$  \hspace{1cm} (6.63)
$$B(q, \theta) = b_0 + b_1 q^{-1} + b_2 q^{-2} + b_3 q^{-3} + b_4 q^{-4},$$  \hspace{1cm} (6.64)
$$\theta^T = [b_0, b_1, b_2, b_3, b_4, a_1, a_2, a_3, a_4, c_0],$$  \hspace{1cm} (6.65)

and $c_0$ a parameter determining the variance of the measurement noise $\varepsilon_t$. The parame-
terisation of $c_0$ is again logarithmic according to (6.42).

![Pole-zero plot for 4th order true system](image)

*Figure 6.29: Pole-zero placement for true system to be identified.*

To obtain the necessary input–output data, a true system was chosen with resonant behaviour. As shown in Figure 6.29, the poles were selected at $0.95 \exp \left( \pm \frac{\pi}{12} \right)$ and $0.75 \exp \left( \pm \frac{\pi}{3} \right)$. The resulting coefficients are

\[
b_0 = 0 \quad b_1 = 0.0029 \quad b_2 = 0.027 \quad b_3 = 0.023 \quad b_4 = 0.0020 \\
a_0 = 1 \quad a_1 = -2.59 \quad a_2 = 2.84 \quad a_3 = -1.71 \quad a_4 = 0.51.
\]  

(6.66) \hspace{1cm} (6.67)

The output of the true system is then corrupted with uniformly distributed noise according to (6.14), with variance $\sigma_v^2 = 0.01$. The resulting input-output data record that is to be used for identification is shown in Figure 6.30.

![Input u, Noise-free y, Noise-corrupted y](image)

*Figure 6.30: Example fourth order system response data used for identification.*
While the noise in the data record was generated with a uniform distribution, the initial identification will be done with a Gaussian noise assumption,
\[ \varepsilon_t \sim \mathcal{N}(0, \sigma^2). \] (6.68)
as introduced in (6.10). By making this assumption, a maximum likelihood estimate will exist, and a point estimate may be made using gradient search techniques. This estimate is used for the starting location of the Markov chain in the Metropolis algorithm.

\[ \begin{align*}
    b_0 &= 0.041 & b_1 &= 0.186 & b_2 &= -0.215 & b_3 &= 0.202 & b_4 &= -0.011 \\
    a_0 &= 1 & a_1 &= -1.24 & a_2 &= 0.013 & a_3 &= 0.137 & a_4 &= 0.207.
\end{align*} \] (6.69)

### 6.9.1 Estimation of Fourth Order System

Using the same implementation of Algorithm 6.1.1 as for the first order system previously, and the per-axis adaptive proposal scheme described in Section 6.8.6, marginal density functions were estimated for the fourth order model. Figure 6.31 shows the results of the estimation after 10\(^8\) iterations of the Markov chain.

The first nine plots show the estimates of the marginal probability density functions for each of the model parameters, which are then followed by \( \varepsilon_0 \) which is the estimate of the measurement noise. The final two plots show estimates of functions of the parameters. In this case the magnitude and phase of \( G \) at a frequency of \( \omega_0 = \frac{\pi}{20} \) radians per sample.

These plots show four traces. The solid trace indicates the results from one sample run of the Markov chain through to 10\(^8\) iterations. On a 2010 era PC, this took approximately 4 minutes of computing time. The upper and lower dashed traces represent the extrema of 100 separate realisations of the Markov chain, to indicate a region of confidence for the results. The final dotted trace represents the true density, as computed by 10\(^10\) iterations of the Metropolis algorithm, with an improved proposal method, as described below.

This plot is equivalent to that shown in Figure 6.19 for the first-order, 2 parameter system, from 10\(^4\) MCMC iterations. Clearly performance has degraded considerably in moving to the larger estimation problem, with the number of iterations required increasing by approximately 4 orders of magnitude.

Note that there is some comfort in the results, in that the final three plots show much better convergence than the parameters themselves. The difference with these three is that they represent physical quantities, those being the measurement noise and the frequency response. So even if the characterisation of the parameters themselves is poor, the characterisation of quantities of physical importance is somewhat better.

### 6.9.2 Hessian-Based Proposal

In Section 6.8 a variety of schemes were presented to adjust the covariance of the random walk proposal. In this section the aim is to modify the proposal in order to improve
Figure 6.31: Marginal parameter estimates for 4th order system at $10^8$ iterations. Solid trace is sample run from one chain, dashed traces the bounds over 100 chains, dotted trace the true result.
the performance of the Metropolis algorithm for the 4th order system. Recall that the
random walk proposal is defined as,
\[ \xi_{k+1} = \theta_k + \nu_k, \] (6.71)
where \( \nu_K \) is a zero-mean symmetric random process, chosen to be a multidimensional
normal distribution,
\[ \nu_k \sim \mathcal{N}(0, \Sigma_v \Sigma_v^T). \] (6.72)
To this point, the covariance of this random process has not included correlation between
the different parameters. However, with the covariance matrix \( \Sigma_v \Sigma_v^T \) now being of di-
mension 10 \( \times \) 10, it isn’t immediately clear how one might go about more appropriately
choosing its elements. With such a large dimension it isn’t possible to simply adapt them
all via some adaptive proposal scheme.

As a consequence of Theorem 5.8.1, it was noted that the fastest convergence to the
correct sampling density occurred when the proposal density followed the shape of the
target density. This provokes the search for an alternate proposal distribution that is
more similar in characteristics to \( p(Y \mid \theta) \). Clearly it isn’t possible in general to sample
directly from \( p(Y \mid \theta) \), but there are approximations that are viable to sample from.

The approach taken here is to retain the random walk for the proposal, and to tailor
the covariance of the walk step to better suit the properties of \( p(Y \mid \theta) \), allowing greater
movement in \( \theta \) without causing a drop in acceptance ratio.

In order to find such a proposal, first define the log likelihood
\[ L(Y \mid \theta) = \log p(Y \mid \theta). \] (6.73)
Consider then a linearisation of \( L(Y \mid \theta) \) around the current value of \( \theta \). So for some \( \xi \)
near \( \theta \),
\[ L(Y \mid \xi) \approx p(Y \mid \theta) + L'(Y \mid \theta)(\xi - \theta) + \frac{1}{2}(\xi - \theta)^T L''(Y \mid \theta)(\xi - \theta). \] (6.74)
In order to simplify this expression, consider the case where \( \theta = \hat{\theta}_N \), the maximum
likelihood estimate. Being at the peak of likelihood function, it may reasonably expected
to be representative of the shape of the region where the Markov chain spends many of
its iterations. For this point, \( L'(Y \mid \theta) = 0 \), and so (6.74) reduces to,
\[ L(Y \mid \xi) \approx L(Y \mid \theta) + \frac{1}{2}(\xi - \theta)^T L''(Y \mid \theta)(\xi - \theta). \] (6.75)
The objective now is to choose a distribution for the walk distance \( \xi - \theta \) that maximises
the movement in \( \xi \) away from \( \theta \) for a given change in the value of \( L(Y \mid \xi) \).
Gaussian distribution, this may be achieved by choosing a covariance,

\[ \xi - \theta \sim \mathcal{N} \left( 0, k_\sigma \left( L''(Y \mid \hat{\theta}_N) \right)^{-1} \right) \]  

(6.76)

It remains to evaluate \( L'' \). This second derivative of \( L \) converges to the information matrix \( \mathcal{I} \), as also does the covariance of a prediction error estimate \( V_N(\theta) = L(Y \mid \theta) \), up to a scaling constant. Consequently, the covariance matrix estimate \( P \), that naturally arises in (5.33)–(5.34) may be used as estimate of the required Hessian \( L''(Y \mid \theta) \). Based on this, the walk perturbation for the random walk in (6.71) is logically chosen as,

\[ \nu_k \sim \mathcal{N} \left( 0, k_\sigma P^{-1} \right) \]  

(6.77)

with \( P^{-1} \) from (5.34), and where \( k_\sigma \) is determined using the univariate proposal adaption strategy in Section 6.8.4 to regulate overall acceptance ratio.

The results from applying the walk covariance in (6.77) are shown in the marginal probability density functions in Figure 6.32. These plots may be compared against the results from the basic proposal shown in Figure 6.31. Critically, the results shown in Figure 6.32 were obtained with only \( 10^6 \) iterations, as opposed to the \( 10^8 \) required previously.

The saving in computation time of this approach may be seen by plotting the total variation norm, as shown in Figure 6.33. For most of the parameters, the number of iterations required is reduced by around three orders of magnitude. Even for those that represent physical parameters, which were previously observed to be less affected by convergence problems, there is still significant benefit in using the modified walk covariance.

### 6.9.3 Sample Covariance Proposal

The improvement resulting from using the modified proposal walk covariance in (6.77) is significant. However, it’s only possible to estimate the required \( P^{-1} \) matrix, (5.34), for a restricted set of underlying model structures. This can be done for many linear system model structures, but for general non-linear systems an alternate approach is necessary.

The required covariance matrix, \( P \) is related by (5.33), to the covariance of the parameter estimate, \( \hat{\theta}_N \). However, this is also something that may be estimated through sample averages from the MCMC algorithm itself. Consequently an adaptive algorithm is proposed, where the sample covariance from a prior run of the chain is used as the basis of the proposal covariance.

As with most adaptive proposal schemes, past statistics of the chain cannot be simply be used in the construction of the proposal, without breaking the Markov property of the chain upon which the convergence proofs are based. For this reason, this method is implemented via adaption at discrete epochs through the overall run of the chain, and decreasing in frequency over time.
Figure 6.32: Marginal parameter estimates using the Hessian based proposal, at only $10^6$ iterations. Solid trace is sample run from one chain, dashed traces the bounds over 100 chains, dotted trace the true result.
6.9. Fourth Order Linear System

MCMC Convergence: Mean Total Variation Norm

Figure 6.33: Convergence comparison between the basic unit axis proposal and the proposal using the Hessian directions.
For the purposes of demonstrating this approach, it was applied to the fit of the fourth-order linear model to data illustrated in Figure 6.30. Adaption epochs were chosen at iterations \( \{i_c\} \), where

\[
i_c = \left\lfloor 10^{(4 + \frac{c}{2})} \right\rfloor, \quad c = 0, 1, 2 \ldots
\]

This means that from \( 10^4 \) iterations onward, a sample covariance is taken twice per decade. This is used as the basis of the proposal walk covariance for subsequent iterations.

The results of this proposal scheme are shown in Figure 6.34. This figure shows the total variation norm of the error in the distribution as a function of the number of iterations, and is compared against using the Hessian estimate, as described in Section 6.9.2.

As would be expected, up until \( 10^4 \) iterations, the covariance based scheme follows that of the basic proposal, as seen in Figure 6.33. To this point, they are using identical proposals. However, at \( 10^4 \) iterations the new proposal based on sample covariance is applied, and the convergence properties of the algorithm immediately improve. The effect of the second adaption at \( 3.2 \times 10^4 \) iterations is also clearly visible. From \( 10^5 \) iterations onwards, the performance closely tracks that of the Hessian-based proposal, and is slightly superior for some parameters.

While it is still advantageous to use the Hessian information where it is available, this performance shows that sample-based methods can be quite effective in cases where the Hessian can’t easily be computed.

### 6.10 Model Order and Convergence

In comparing the results of the first-order example system to the fourth-order system, there is clearly an additional computational cost in moving to a higher order system. Even with the updated Hessian-based proposal, in higher order systems, more iterations are required for a given degree of accuracy in the computed probability density functions.

This may be seen by comparing the convergence seen in the total variation norm in Figure 6.7, to that in Figure 6.33. To better quantify the computational cost of adding additional parameters, the transfer-function model structure presented in (6.1) is here profiled for varying model order.

For a total of \( m \) parameters, the denominator and numerator orders have been chosen respectively as,

\[
m_a = \left\lfloor \frac{m}{2} \right\rfloor, \quad m_b = \left\lfloor \frac{m - 1}{2} \right\rfloor.
\]

An input-output dataset was generated for the case of \( m = 16 \), by extending the fourth-order model described in Section 6.9. The additional poles and zeros of true system are shown in Figure 6.35, and resulting input–output plot in Figure 6.36.

This data record was then used to identify parameters for the model in (6.1), for \( m \) between 1 and 16, with \( m_a \) and \( m_b \) as defined in (6.79). The number of iterations
6.10. Model Order and Convergence

MCMC Convergence: Mean Total Variation Norm

Figure 6.34: Convergence comparison between the adaptive sample covariance based proposal and the proposal using the Hessian directions.
required to reach a given accuracy on the marginal density $p(b_0)$ were then found. As there is no efficient alternate method of determining the true density against which to compare, this was found by running $10^9$ iterations of the MCMC algorithm and ensuring convergence across multiple chains. The solid and dashed traces in Figure 6.37 show these results for total variation norms of $10^{-2}$ and $10^{-1}$ respectively.

For this system, there is a clear trend to an increasing number of iterations as the number of parameters increases. However the cost of additional parameters is much lower than it is for direct integration methods. Here there is still less than an order of magnitude difference in going from one parameter up to 16 parameters. This shows that given an appropriate target density and choice of proposal function, the MCMC methods can scale well to increasing numbers of parameters.
In Section 5.2.2, state-space models were introduced as an important class of model structures. While being able to capture multi-variable dynamics, they also pose additional challenges to the implementation of MCMC methods.

Consider initially the linear Gaussian state-space model, originally presented in (5.9),

\begin{align*}
x_{t+1} &= Ax_t + Bu_t + w_t, \quad \text{(6.80a)} \\
y_t &= Cx_t + Du_t + \varepsilon_t, \quad \text{(6.80b)}
\end{align*}

for parameter matrices $A, B, C, D$. The sequences $\{w_t\}, \{\varepsilon_t\}$ are independent and zero-mean with

\begin{align*}
\mathbb{E} \{w_tw_t^T\} &= Q, \quad \text{(6.81)} \\
\mathbb{E} \{\varepsilon_t\varepsilon_t^T\} &= R. \quad \text{(6.82)}
\end{align*}

The additional complication offered by state-space models is the inclusion of the process noise $\{w_t\}$. This precludes using the simulation approach that was utilised for the transfer function models, as the state is no longer a deterministic function of the input and parameters. However, for the case where $\{w_t\}$ and $\{\varepsilon_t\}$ are Gaussian, the required $p(Y | \theta)$ may be computed via the Kalman filter[1].
6.11.1 The Kalman Filter

The Kalman filter[29] may be used to find an optimal estimate \( \hat{x}_t \), of the state \( x_t \) of the linear state-space system in (6.80). It does this by iteratively computing the complete probability density function \( p_\theta(x_t | Y_t) \) of the state for each time step \( t \). Due to linear and Gaussian nature of the system, this probability density may be expressed by its mean \( \hat{x}_t \), and covariance \( P_t \).

The Kalman filter may be conveniently separated into two stages, a measurement update, and a time update. The measurement update takes an a priori estimate \( \hat{x}_{t|t-1}, P_{t|t-1} \) of the state, together with the measurement \( y_t \), to compute the a posteriori estimate \( \hat{x}_{t|t}, P_{t|t} \). The subsequent time update predicts the state forward one time step for a new a priori state estimate, and corresponding covariance.

The measurement update may be performed by the set of equations,

\[
S_t = CP_{t|t-1}C^T + R, \tag{6.83}
\]
\[
K_t = P_{t|t-1}C^T S^{-1}, \tag{6.84}
\]
\[
\hat{x}_{t|t} = \hat{x}_{t|t-1} + K_t(y_t - A\hat{x}_{t|t-1} - Du_t), \tag{6.85}
\]
\[
P_{t|t} = (I - K_tC)P_{t|t-1}. \tag{6.86}
\]

The time update is then achieved by the prediction equations,

\[
\hat{x}_{t|t-1} = A\hat{x}_{t-1|t-1} + Bu_t, \tag{6.87}
\]
\[
P_{t|t-1} = AP_{t-1|t-1}A^T + Q. \tag{6.88}
\]

This knowledge of the probability density of the state \( x_t \) may be used to compute the likelihood of an observed output \( y_t \). The measured output \( y_t \) may be represented as a sum of the predicted output with the innovations \( e_t \).

\[
y_t = C\hat{x}_{t|t-1} + Du_t + e_t. \tag{6.89}
\]

Based on the independence of \( \{w_t\} \) and \( \{e_t\} \), the innovations \( e_t \) is a zero-mean Gaussian with covariance,

\[
\mathbb{E}\{e_t e_T\} = S_t, \tag{6.90}
\]

where \( S_t \) is computed from (6.83). The required likelihood may then be computed via,

\[
p(y_t | y_{t-1}, \ldots, y_0, \theta) = p_e(y_t - C\hat{x}_{t|t-1} - Du_t). \tag{6.91}
\]

6.11.2 State-Space Example

To provide a basic example of MCMC identification of a state-space system, a first order linear single-input single-output state-space system is considered. The model structure
chosen is

\[ x_{t+1} = \theta_1 x_t + \theta_2 u_t + \theta_3 w_t, \]  
\[ y_t = x_t + \theta_5 u_t + \theta_6 \varepsilon_t, \]  

(6.92a)

(6.92b)

where the state \( x_t \in \mathbb{R} \), the input \( u_t \in \mathbb{R} \), and the output \( y_t \in \mathbb{R} \). The noise terms \( w_t \) and \( \varepsilon_t \) are normally distributed with

\[
\begin{bmatrix} w_t \\ \varepsilon_t \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right).
\]

(6.93)

The \( \theta_4 \) parameter has been constrained to unity, and is hence absent in (6.92b) to avoid the system becoming unidentifiable.

To generate a dataset to identify, a set of \( N = 40 \) outputs was computed using (6.92) as the true system, with parameter values chosen as

\[ \theta = [0.9, 1.0, 0.1, 0.0, 0.1]^T. \]  

(6.94)

The initial state was set to \( x_0 = 0 \), and the input signal was distributed according to,

\[ u_t \sim \mathcal{N}(0, 1). \]  

(6.95)

Figure 6.38 shows the results of applying the Kalman filter in the MCMC algorithm. The MCMC method of Algorithm 6.1.1 was used, together with the Kalman filter of (6.83)--(6.88) to evaluate the probability ratio required in Step 3 of that algorithm. The proposal used was derived from the sample covariance method described in Section 6.9.3.

At \( 10^4 \) iterations, this result may be compared to the results shown in Figure 6.9 for the first-order transfer function model. The accuracy is slightly inferior to that transfer function model, but this is to be expected given the higher model order of the system considered here.

The use of the Kalman filter does add additional computation burden to the evaluation of the MCMC algorithm. Instead of a simple simulation though the dataset for each iteration, it requires the Kalman filter equations to be run. However, this is not a great cost, and the \( 10^4 \) iterations, as shown here, take only 29 milliseconds to compute on one core of a 2010-era PC.

### 6.12 Non-Linear Models

While the Kalman filter provides an efficient means to compute the required likelihood function \( p(Y | \theta) \) for linear state-space systems, it doesn’t provide a general solution for non-linear systems or those with non-Gaussian noise characteristics. In this section, the particle filter is introduced as a means of computing the likelihood for quite arbitrary
$P_t$, at $10^4$ MCMC Iterations

Figure 6.38: Parameter estimates for the linear system (6.92) after $10^4$ MCMC iterations.
model structures of the form,

\[ x_{t+1} \sim f_\theta(x_{t+1} \mid x_t), \]  
\[ y_t \sim h_\theta(y_t \mid x_t), \]

where \( f_\theta(\cdot) \) and \( h_\theta(\cdot) \) are distributions that are parameterised by the vector \( \theta \).

### 6.12.1 Particle Filter

The particle filter was described in Section 2.8. Its underlying approach is based on using a sampling approach to form a non-parametric probability density function representation. The density function \( p_\theta(x_t \mid Y_t) \) is represented by a set of \( M \) “particles,” which consist of weighted samples drawn from the distribution itself. The \( i \)-th particle is represented by value \( x^i \), and a weight \( w^i \). The set of particles, may then be seen as a representation of the density function \( p_\theta \) according to,

\[
\int_A p_\theta(x) \approx \sum_{\{i : x^i \in A\}} w^i. \tag{6.97}
\]

The aim of the particle filter is to adequately represent the change in probability density of the state over time. The three main stages of the algorithm were represented diagrammatically in Figure 2.8, which is reproduced here as Figure 6.39.

![Particle filter steps](image)

*Figure 6.39: Particle filter steps.*

The utility of the particle filter in the context of MCMC parameter estimation for
(6.96) is that for each $t$, it provides a means of estimating $p(y_t \mid Y_{t-1}, \theta)$ according to

$$p(y_t \mid Y_{t-1}) = \int p(y_t \mid x_t) p(x_t \mid y_{t-1}) \, dx_t$$

$$\approx \sum_{i=1}^{M} w_i p(y_t \mid x_i)$$

Then by applying (5.20), $p(Y \mid \theta)$ may be estimated via,

$$p(Y \mid \theta) = p(y_1 \mid \theta) \prod_{t=2}^{N} p(y_t \mid Y_{t-1}, \theta).$$

This is the quantity that is required in order to compute the acceptance probability of the Metropolis–Hastings algorithm (Step 3 of Algorithm 6.1.1). However, for the convergence proofs presented in Chapter 5, the probability density itself must be computed, and not simply an estimate of it. The accuracy of the state probability estimates that are achieved clearly depend on the number of particles, $M$, that are realised. Despite this, it has been shown[2], that MCMC convergence results are still met for a finite $M$ when the particle filter is used within the Metropolis–Hastings algorithm.

### 6.12.2 Linear Example

The linear state-space model presented in (6.92) is a convenient example to evaluate the performance of the particle filter within the Metropolis algorithm. The same data and methodology used in the Kalman filter example are here applied, but using the particle filter from Algorithm 2.8.1 in place of the Kalman filter.

Figure 6.40 shows the estimated marginal probability density functions for an MCMC run of $10^4$ iterations, using a particle filter to evaluate $p(Y \mid \theta)$. Comparing against the Kalman filter results in Figure 6.38, there is clearly a cost to the quality of the results in using this particular implementation of the particle filter.

In using a particle filter there is a design decision of how many particles to use to represent the state density function. The computational requirements of the particle filter rises approximately linearly with the number of particles used, but yet a larger number of particles will result in a more accurate estimate of the state density functions.

The example results shown in Figure 6.40 were completed using a particle filter with $M = 64$ particles. To better quantify the effect of the number of particles, the identification process was repeated for varying number of particles, between 1 and 512. The speed of convergence of each of these estimates are shown in Figure 6.41. In this figure, for each particle count, the error in the estimated $\theta_1$ distribution is shown as a function of the number of iterations. The total variation norm is used to quantify the difference between the estimate and the “true” density, which was evaluated via parallel runs of $10^7$ iterations of the MCMC algorithm.
6.12. Non-Linear Models

Figure 6.40: Parameter estimates for the linear system (6.93) after $10^4$ MCMC iterations using a particle filter with 64 particles.
As expected, the trend in Figure 6.41 shows improving algorithm convergence for increasing particle count in the particle filter. The “KF” trace demonstrates the result of using the Kalman Filter, as described in the previous section, and this trace is consistent with the limit as the number of particles in the particle filter approaches infinity.

The nature of the particle filter MCMC method is that the estimate may be improved by increasing either the number of particles, or the number of MCMC iterations. Overall computational load is approximately linear in each of these, yet the important question remains of how to optimise this computational trade-off. Figure 6.42 demonstrates the results of performing this parameter estimation with a range of particle counts $M$. The number of particles are presented on the $x$-axis, while the $y$-axis shows the approximate average computational burden required to reach a density estimate with total variation norm $\|p(\cdot) - \pi(\cdot)\|_{tv} = 10^{-1}$.

From this plot, there is clearly an optimum for this system of about 100 particles. For a higher number of particles, the improvement in state estimation that can be gained does not justify the additional cost in the particle filter, and the processing power would be better put into running more MCMC iterations. Clearly also, when the number of particle is below this optimum, excessively large numbers of MCMC iterations are necessary to compensate for the poorer estimates.

### 6.12.3 Nonlinear Example

In order to demonstrate the applicability of the Particle Filter MCMC approach to the identification of non-linear systems, a long-standing benchmark model structure[36] is
6.12. Non-Linear Models

now considered. This takes the form of the following state-space model,

\[
x_{t+1} = \theta_1 x_t + \theta_2 \frac{x_t}{1+x_t^2} + \theta_3 \cos(1.2t) + \theta_4 w_t, \tag{6.101a}
\]

\[
y_t = \theta_5 x_t^2 + \theta_6 e_t, \tag{6.101b}
\]

\[
\begin{bmatrix} w_t \\ e_t \end{bmatrix} \sim \mathcal{N} \left( \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \right). \tag{6.101c}
\]

The true parameters chosen for this example are \( \theta^* = (0.5, 25, 8, 1.0, 0.05, \sqrt{10})^T \). One simulated realisation for these parameters was then used to generate the dataset to identify.

With the same methodology as for the previous linear case, marginal density functions were estimated for all of the six parameters. The noise variance terms \( \theta_4 \) and \( \theta_6 \) were constrained to be positive. Similarly \( \theta_3 \) was also constrained to be positive, as each positive solution for \( \theta_3 \) also has a negative equivalent. The resulting density functions after \( 10^5 \) iterations with \( M = 64 \) particles are shown in Figure 6.43.

Again, the “true density” plot has been constructed simply by using samples from multiple runs of the algorithm, each over \( 10^7 \) iterations. After the \( 10^5 \) iterations, this is quite clearly a very usable result showing the confidence in the parameter estimates given the supplied data.

Again, the convergence properties of the algorithm as a function of the number of particles is of interest. Figure 6.44 shows algorithm convergence in terms of the total variation norm as the number of iterations increase. While there is no Kalman filter...
Figure 6.43: Parameter estimates for the nonlinear system (6.101) after $10^5$ MCMC iterations using a particle filter with 64 particles.
version to compare against in this case, the behaviour is very similar to the linear case. With a particle count from 128 onwards, the costs due to the errors in the particle filter approximation become quite small.

![MCMC Convergence](image)

*Figure 6.44: Total variation norm as a function of iterations for varying particle counts, for $\theta_1$ parameter of the nonlinear system (6.101).*

To find the computationaly optimal number of particles, Figure 6.45 plots a measure of the average computational load required, versus number of particles, in order to reach

![Computational cost](image)

*Figure 6.45: Computational cost as function of particle counts, for $\theta_1$ parameter of the nonlinear system (6.101).*
a density convergence with total variation norm of value $10^{-1}$. Again the optimal choice for quality of distribution per given computational load is about $M = 100$ particles for the particle filter.

6.13 GPGPU Implementation of PF-MCMC

While the MCMC algorithm itself can quite rapidly produce useful results with just the computational resources of a modern personal computer, the same is not necessarily true once it is coupled with a particle filter. The computational burden is based on the product of the number of data samples, the number of particles, and the number of MCMC iterations. The extra approximately two orders or magnitude in computation time needed to run the particle filter, will turn a computation that would take minutes into one that takes hours.

However, sampling algorithms, such as MCMC and particle filters, are generally well suited to parallel computing architectures, and this can offer a means to greatly improve execution times. The quality of the sampling algorithm estimates are determined by the total number of samples drawn. These samples may be drawn from multiple concurrently running samplers as easily as those executed sequentially. As there is little interaction required between each sampler, performance may generally increase linearly with the number of processing cores available.

In recent years Graphics Processing Units (GPU) have emerged as an effective tool for parallel computation. While originally developed for the personal computer gaming market, they offer considerable general-purpose computational capability at a modest cost. These devices provide in the order of 500 parallel processing cores, offering up to the order of an hundred-fold increase in processor speed over a conventional CPU for suited applications. The use of GPU devices for non-graphic computational purposes is known as General-Purpose GPU, or GPGPU.

This thesis will consider the GPU computational architecture currently produced by Nvidia Corporation, the Compute Unified Device Architecture (CUDA), but the principles apply equally for other GPU architectures.

6.13.1 CUDA Architecture

Nvidia use a variant on a Single Instruction Multiple Data (SIMD) processor arrangement. In an SIMD system, there are multiple logic units, each of which process all the same instruction at any given clock cycle. However, each may operate on different data. This is convenient for array processing where the same operations are required across an array of values.

A limitation of SIMD systems is met when conditional branching is needed in algorithm code. The problem in this case is that some of the array elements may require taking a different branch to other elements, resulting in a divergence of instruction exe-
cution. This is overcome in Nvidia’s Single Instruction Multiple Thread (SIMT) architecture. SIMT behaves as SIMD, except serialises the divergent paths, so that different instruction paths can be accommodated for different processing cores. While only one instruction may still be executed at time, flexibility is provided at the cost of additional computational time.

\[
y[i]=x[i]*x[i];
\]

\[
\text{if (i>1)}
\]
\[
x[i]=i-1;
\]

\[
\text{else}
\]
\[
x[i]=0;
\]

\[
z[i]=x[i]*x[i];
\]

*Figure 6.46: Example CUDA C code.*

As an example, consider the CUDA C code in Figure 6.46. Here the index \(i\) denotes the thread index, which will be unique for each processing unit. The first line is easily run in parallel in an SIMD architecture, as each processor is computing the same instructions. However, the if statement splits the processors into two separate paths, which cannot be executed in parallel. The actual execution of this code is shown in Figure 6.47. The different branches of the if statement are serialised, with one branch being executed before the other. Finally, after each path is executed, parallel execution continues across all threads.

\[
1 \quad y[0]=x[0]*x[0]; \quad y[1]=x[1]*x[1]; \quad y[2]=x[2]*x[2]; \quad y[3]=x[3]*x[3];
\]
\[
2 \quad x[2]=2-1; \quad x[3]=3-1;
\]
\[
3 \quad x[0]=0; \quad x[1]=0;
\]
\[
4 \quad z[0]=x[0]*x[0]; \quad z[1]=x[1]*x[1]; \quad z[2]=x[2]*x[2]; \quad z[3]=x[3]*x[3];
\]

*Figure 6.47: Execution of code from Figure 6.46.*

The result of this approach is a compromise. Code may be written such that the underlying SIMD machine may be ignored, and conditional paths taken for different threads. The cost of doing this is that performance may be severely degraded if there is significant branching among the parallel code. Consequently, in order to effectively use this parallel architecture, it’s necessary that the algorithm be coded to minimise branch paths between threads. To do this all cores need to be executing the same code as much as possible.

**Execution Units**

A significant limitation of having all of the processing cores on the device executing the same instructions simultaneously is that there would be significant memory contention problems. When a memory access is required, all of the cores will require a read at the
same time, and each core could require access from a different memory location. A means of overcoming this is to stagger the execution times between different sets of processor cores. This is done by partitioning the total number of cores over independent sets of SIMD coupled processors.

Within the CUDA architecture, Nvidia groups the GPU ALUs (Arithmetic Logic Units) into sets of 32, and denotes the use of each set as a *warp*. Each warp behaves as a SIMT processor, with only one instruction control unit per warp.

Figure 6.48: Nvidia’s SIMT processor architecture.

Figure 6.48 depicts the structure of the Nvidia’s GPUs. There is a control unit associated with each group of 32 ALUs, and on-board memory accessible by all cores. For the case of the GTX560Ti profiled, there are a total of 384 cores, divided into 12 warps of 32 cores each.

### 6.13.2 Thread Abstraction

While it is useful to recognise the physical layout of the cores and instruction control units in the GPU to understand performance, the CUDA software architecture does provide an abstraction that hides some of these details. In particular, warps are not allocated directly by the programmer.

The main programming elements in CUDA are *threads*, *blocks*, *grids*, and *kernels*. A *thread* is used in the conventional sense of the term, indicating one path of execution, for one dataset. A group of threads are launched together in a *block*, as shown in Figure 6.49. Each block of threads will generally execute the same instructions, each operating on different data. Each block may be composed of one or more warps of 32 threads, but the specific number of warps may be left transparent to the user. There may be more threads present in a block than there are cores on the GPU, and the scheduler will time-slice between the different threads for the available resources.

In addition to having multiple threads running concurrently in a block, CUDA also allows multiple blocks to be scheduled to run as a *grid*. All of the blocks in the grid run
6.13. **GPGPU Implementation of PF-MCMC**

Figure 6.49: Thread grouping program interface.

a common *kernel*, which is defined by a function in CUDA C code. The grid essentially allows a greater number of threads to be concurrently run than just those in a single block.

Blocks are limited to around 1000 threads, and while that is more than the number of cores on the GPU, it’s useful to have many more threads available for execution than there are cores, as cores often need to wait on memory access. While one warp of threads is waiting for memory to be available, the CUDA scheduler can run an alternate warp instead.

While all of the blocks in a grid run the same kernel (set of instructions), threads within a block are more tightly bound than those across different blocks. It’s possible to synchronise the threads within a block, and also utilise fast shared memory between threads in a single block. In order to allow any synchronisation between threads of different blocks, it’s necessary to end execution of the kernel, and start a new kernel. Consequently, when there is any dependency between threads, it is better that they execute within a single block if the block size capabilities of the GPU will allow it.

### 6.13.3 Memory Model

Due to its parallel nature, GPU programming brings special requirements to the memory subsystem. In particular, the memory bandwidth increases up to linearly with the number of cores. In an SIMD processor, as each core is working with different data, when a memory access is required, data needs to be fetched for all the cores at once. So in an SIMD processor with hundreds of cores, hundreds of memory locations may need to be read simultaneously.

One way that CUDA overcomes this bottleneck is to have a surplus of threads ready to run, to occupy the processors while other threads are waiting on memory accesses. A consequence of this is that for efficient execution, and an algorithm needs to be significantly more parallelisable than would be indicated by the number of cores on the device. For example on a device with 384 cores, it would be desirable to have several thousand threads ready to run at any given point in time.

As shown in Figure 6.50, the CUDA architecture provides global, shared, and constant memory spaces within the device. These each provide important characteristics that can
be utilised to improve performance. Shared memory is shared within the threads of a single block, and is significantly faster than the general global memory. However it has very limited capacity, which can translate to only tens of bytes per thread within the block.

### 6.14 Parallel GPU Implementation

MCMC and Particle Filters are both inherently quite parallelisable algorithms. In the case of MCMC, instead of one long Markov Chain, a number of shorter parallel chains may be implemented. Apart from some loss of efficiency in reaching chain “burn-in” for all chains, the algorithm performance per kernel evaluation remains largely static. This may be achieved as the individual chains may operate largely in independence of one another.

Similarly, a particle filter is conceptually quite suitable for implementation in parallel hardware. There are many particles that exist in parallel, each of which need to be propagated along a number of time-steps. However there are some interactions between particles that influence how such a filter may be implemented in a parallel fashion.

Given the relative ease of parallel execution of the MCMC algorithm, one could attempt to utilise the parallel resources by simply running many parallel MCMC chains, and running the particle filter serially in each. However, the limitation of this approach is the GPU hardware is requiring thousands of parallel threads for efficient operation, and MCMC becomes more difficult to use effectively over this number of parallel chains.

Alternatively, the parallel hardware could be utilised by running one thread per particle for the particle filter, and executing the MCMC sequentially. However, the optimum number of particles is around one hundred, which is significantly less than the required number of threads.
The best computation solution is one where there are both parallel MCMC chains, and parallel particles in a particle filter all running concurrently. In this situation, 100 particles and 100 chains provides 10,000 concurrent threads for the GPU hardware to schedule from. The primary computation will involve the time and measurement updates across each of the particles in each of the parallel particle filters.

6.15 A Parallel Particle Filter

As described in Section 2.8, there are three main components to the computation for each time-step of a particle filter.

- Propagation
- Weighting
- Resampling

The first two of these, the propagation and weighting are quite straightforward to implement in a parallel manner. With one thread per particle, the values for each particle may be computed at once with a SIMD machine.

6.15.1 Propagation

For propagation, each state moves independently according to the model state update equation. In the example non-linear system, this state update is described in (6.101a), where for the $i$-th particle,

$$x_{t+1} = \theta_1 x_t^i + \theta_2 \frac{x_t^i}{1 + (x_t^i)^2} + \theta_3 u_t + \theta_4 w_t$$  \hspace{1cm} (6.102)

The parameter vector $\theta$, and input value $u_t$, are common to all particles in the filter. Each particle has a unique state value $x_t^i$, which is retained in memory local to the thread. The noise term $w_t^i$, is simply a realisation drawn from the state noise density, with each particle receiving a different realisation from the random number generator.

In this application there are multiple parallel particle filters running, with one for each concurrent MCMC chain. The particles over different filters remain identical and execute concurrently, except for having a different parameter vector $\theta$ for each chain. A much lesser level of synchronisation is required between threads that don’t correspond to the same MCMC chain, so these may appear in different GPU execution blocks.

6.15.2 Weighting

The weighting step of the particle filter algorithm involves computing the likelihood of the measured output for the current state. For the example system, this is according
to (6.101b). The likelihood computation is achieved by first evaluating the residual,

\[ \hat{e}_i = \frac{1}{\theta_6} (y_i - \theta_5 x_i^2) \].

(6.103)

Then \( p_e(\hat{e}_i) \), the probability density function of the noise model, may be evaluated. For zero-mean normally distributed noise, this is simply,

\[ p_e(e_i) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp \left( -\frac{e_i^2}{2\sigma^2} \right). \]

(6.104)

As in the case of the propagation step, this computation may be efficiently performed entirely in parallel.

### 6.15.3 Resampling

The remaining step is that of resampling. This is where a new set of unit-weight particles are drawn according to the weighted distribution of the present particles. Here a number of problems are presented for a parallel implementation. There is now significant interaction between each of the particles, and even the mapping of one thread per particle is in question, as some particles disappear, and others are multiply sampled to produce several new particles.

The conventional sequential approach is to start by forming a cumulative sum of the weights from the existing particles. This performs the dual role of a means for resampling out of the cdf, and also computing a sum of the weights to allow for normalisation. A cumulative sum is very efficient on a scalar processor, simply requiring \( M \) sum and store operations for \( M \) particles. However, in a parallel architecture, this cannot simply be achieved in \( M \) parallel operations. Instead, the addition for each particle requires the result from the addition for the previous particle. The result of a simplistic implementation would then offer no speedup over a scalar processor.

An alternative more suited to parallel implementations is using a tree-structured adder, requiring an order of \( \log_2 M \) iterations over \( M \) parallel threads.

One such approach is illustrated in Figure 6.51. Each of the particle weights are laid along the horizontal axis, and the stages of addition are shown vertically. At the first time step, adjacent weights are accumulated. Over further iterations, different combinations of results are summed to produce the final cumulative sum. The solid lines represent the operations needed to simply find the sum of the values, while the dashed lines show the additional operations required for the intermediate cumulative sum results.

This addition structure uses only 50% of the available resources, and alternate arrangements allow for this computation with half the number of threads. However, as \( M \) threads are executing anyhow, this becomes an efficient way of implementing the sum in this case, without the overheads of altering the structure for a relatively small computational gain. Given that there are only \( \log_2 M \) addition operations, any additional
overheads in setting up loops and pointers can be very costly.

Once the cumulative sum of the particle weights has been calculated, there still remains the task of sampling from the weighted distribution. With a cumulative distribution function now available, Inverse Transform Sampling may be used to convert a uniform distribution into the desired distribution.

The approach used here is systematic resampling[9], which uses a variant of inverse transform sampling, requiring only a single random number draw per set of particles.

After drawing one initial uniform number per particle filter,

\[ \varepsilon_s = U \left( 0, \frac{1}{M} \right), \]  

(6.105)

the uniformly distributed samples for each particle are chosen as

\[ s_i = \varepsilon_s + \frac{i}{M}, \]  

(6.106)
for the $i$-th newly-drawn particle. These uniformly distributed values will now need to be transformed according to the target distribution by mapping to the inverse of the CDF. Using inverse transform scaling, this involves mapping the set $\{s_i\}$ through the inverse of the CDF, as shown in Figure 6.52.

Scalar processor implementations will typically walk through the list of $\{s_i\}$ and the sum of weights, to perform the inverse lookup into the CDF. This will take an order of $M$ steps to move through the $M$ particles and weights. Again this algorithm presents a challenge for parallel implementation, due to the dependence between the successive calculated values.

A solution to this can be reached by now allocating one thread of the parallel processor to each of the new particles to be drawn. Then each of the threads can run a binary search across the CDF to find the appropriate particle to draw as the new sample. Just as the case of the cumulative sum algorithm, the binary search requires $\log_2 M$ iterations.

### 6.16 GPGPU Performance

In order to evaluate the performance advantage possible using a GPU card, a particle filter MCMC sampler was implemented for the example linear state-space system (6.92). Figure 6.53 shows a plot of speed of computation of both the CPU version of the algorithm running on an Intel Core i5-2500, and the GPU version running on an Nvidia GTX560Ti. These are both mid-range devices of 2010 era.

![Figure 6.53: MCMC computation speed for CPU and GPU implementations as a function of the number of particles](image)

This plot shows that the performance with respect to particle filter size is quite different between the two implementations. The trace for the CPU implementation shows that the computation cost increases linearly with the number of particles, as is expected, but the behaviour of the GPU algorithm is quite different. For low particle counts, there is
very little additional cost for computing higher numbers of particles. This is because with low particle counts, it’s difficult for the GPU to efficiently utilise the parallel resources. The GTX560’s minimum warp size of 32 means that there is little difference in the time required to compute one particle versus 32 particles. As the number of particles grow, the ability to effectively use all of the GPU cores is improved.

The two GPU plots in Figure 6.53 show the results for different numbers of parallel MCMC chains, where the iterations count on the y-axis is normalised to show the total number of iterations over all chains. This shows that again by increasing the number of parallel chains, the overall utilisation of the GPU is improved. This is primarily due to the additional thread scheduling flexibility provided to the GPU.

As a point of comparison, the Kalman filter implementation of the MCMC sampler for this system, shown in Figure 6.38, executes at $4.1 \times 10^5$ iterations per second on the same CPU hardware. A CPU based particle filter utilising 128 particles is slower than the Kalman filter version by two orders of magnitude. However, when running in GPU hardware, the particle filter comes close to the speed possible under the Kalman filter. While the Kalman filter will always be a better choice for linear systems with Gaussian noise, these results demonstrate that parallel GPU implementations can bring particle filter approaches to a similar level of performance, but applicable to non-linear and non-Gaussian systems.

![Figure 6.54: Relative speedup of parallel GPU implementation as a function of the number of particles](image)

The overall speedup achieved through using the parallel GPU implementation is shown in Figure 6.54. The peak speedup is $50 \times$, and is reached here with 1024 parallel chains, each with a particle filter of 128 particles. It may be noted that the optimal performance occurs over a relatively narrow range of particles per GPU block. With particle counts much less than 100, the GPU can’t effectively utilise the parallel resources, while for higher particle counts, shared memory resources within the GPU become de-
pleted and efficiency also drops. Fortunately for this application, the region of peak efficiency is well located. By comparing the optimal particle count for the particle filter, as shown in Figure 6.42, the particle filter method is a good match for the GPU capabilities. The GPU may then practically provide the opportunity for significant computation time advantages over a CPU-only approach.

6.17 Conclusion

Markov Chain Monte–Carlo methods are clearly a useful tool in model parameter identification. They provide the means to extract detailed probability density information for model parameters, and quantities that are functions of model parameters. Importantly, unlike existing approaches, there is no requirement for linear Gaussian model structures, or any assumptions about data record length.

Being computational methods, the execution time of MCMC algorithms can potentially be a problem in practical use. However, through a combination of appropriate proposal density design, and the use of parallel computing architectures, execution times can be greatly reduced, to the point where the methods are applicable to systems of practical interest.
Chapter 7

Conclusion

This thesis has demonstrated the role and behaviour of Markov chain Monte–Carlo methods for estimation tasks. Fundamentally, many parameter estimation tasks involve a process of integrating over a potentially quite high-dimensional likelihood function, and stochastic simulation methods can provide an effective means to compute such integrals.

A strong benefit of the Markov chain Monte-Carlo methods is that they have been constructed in a way that is amenable to analysis of their numerical behaviour. Through using the strong body of theory on the properties of Markov chains, proof of the convergence behaviour of the algorithms is made possible. While the actual rate of convergence may still be unknown, proof has been shown that the algorithms presented here produce estimates that eventually converge to the correct value.

In the discrete-state application the Metropolis algorithm and Gibbs sampler were used to perform multi-user detection. These algorithms were shown to provide a good approximation to optimal MAP detection, but with significantly less computational cost than a complete integration. The stochastic methods are still computationally intensive, so additional measures were found to improve the performance. By applying importance sampling, the number of required algorithm iterations were reduced, while an algebraic decomposition allowed much faster execution times per iteration.

The utility of the Metropolis algorithm was also shown through examples in the area of system identification. This application resulted in Markov chains over an uncountable space, and hence variations to the theoretical treatment to complete the proof of convergence. The value of the MCMC methods in this application is that full posterior probability density functions may be estimated for model parameters, and even somewhat arbitrary functions of parameters. While this in itself is a value contribution over existing methods, with the MCMC approach these results are available even for non-linear and non-Gaussian model structures. With the techniques developed in this thesis, these results may also be achieved in a timely manner on readily available computer hardware.
Further Work

The results in this thesis show that there is considerable scope for further investigation into the use of stochastic simulation methods both in communications and control systems. A clear direction of interest in the case of multi-user detection would be to consider the performance of the MCMC detector as part of an iterative receiver design. Given the soft inputs and soft outputs, it is suited to this role, and with the feedback from the decoder, the number of iterations needed may potentially be significantly reduced.

In the area of system identification, there are several areas worthy of further investigation. Some of these are based on the ability that MCMC provides in evaluating probability density functions of arbitrary functions of parameters. In the linear systems case, some possible functions to consider are the gain and phase margins of the system when coupled with a known controller. These new abilities raise the question of how to best utilise them in controller design.

A significant current limitation of MCMC methods in system identification is in the difficulty in implementing the method for each individual task. While the algorithm itself is quite simple, the associated software for result collection and evaluation of performance can be quite considerable. This would be aided greatly by the development of appropriate toolbox software to simplify the process. This is particularly true in the case of the GPGPU hardware implementation, where currently quite low-level programming is required to adapt to new model structures.
Bibliography


