Numerical Methods for Parameter Estimation in Stochastic Systems

By

Cody E. Clifton

Submitted to the Department of Mathematics
and the Graduate Faculty of the University of Kansas
in partial fulfillment of the requirements for the degree of
Master of Arts

Date defended: April 16, 2013

The Thesis Committee for Cody E. Clifton certifies
that this is the approved version of the following thesis:

Numerical Methods for Parameter Estimation in Stochastic Systems

Prof. Bozenna Pasik-Duncan, Chair

Date approved: April 18, 2013

Abstract

The objective of this work is to provide numerical simulations in support of a collection of existing results on estimation in two distinct types of stochastic systems.

In the first chapter, we consider a linear time-invariant higher-order system of order that is subject to white noise perturbation. We numerically illustrate the result that the quadratic variation estimator of the white noise local variance is asymptotically biased when a forward-difference approach is used for numerically approximating the derivatives of the stochastic process, and that the bias can be eliminated by instead applying a specific alternative numerical differentiation scheme. Moreover, we consider the result that the straightforward discretization of a least squares estimation procedure for unknown parameters in the system leads to an asymptotically biased estimate.

In the second chapter, we consider a controlled Markov chain, taking values on a finite state space, whose transition probabilities are assumed to depend on an unknown parameter belonging to a compact set. We first provide numerical illustration of the result that under a particular identifiability condition, the maximum likelihood estimator of this parameter is strongly consistent. Next, we illustrate that under alternative assumptions the sequence of maximum likelihood estimates converges and retains a desirable property relating to the Markov chain's transition probabilities. Additionally, we present a survey of several other related results.

Acknowledgements

I would first like to thank my advisor, Prof. Bozenna Pasik-Duncan, for imparting guidance, wisdom, and encouragement. Her infectious passion for mathematics is truly inspirational.

I wish to thank Prof. Tyrone Duncan and Prof. Zsolt Talata for serving on my Thesis Committee; especially Prof. Duncan, whose suggestions helped to further my research on estimation in higher-order linear stochastic systems.

I would also like to thank the KU Mathematics faculty as a whole for providing instruction and advice.

I would like to recognize Theodore Lindsey for contributing valuable insight toward the development of MATLAB code for simulating observed trajectories of a Markov chain.

I wish to thank the phenomenal Mathematics professors of Whitman College for their efforts in preparing me for graduate school.

I would like to credit Mark Yannotta with originally inspiring me to study mathematics at an advanced level, and to thank him for encouraging me to pursue graduate education at KU and for being my friend and mentor.

Finally, I would like to thank my family for their investment in my education, their support of my varied interests, and their unfailing love.

Contents

1 Estimation in Higher-Order Linear Stochastic Systems 1

1.1 Introduction 2

1.2 Simulating Sampled SDE Solutions 9

1.3 Estimating White Noise Local Variance 11

1.4 Estimating an Unknown Parameter 12

1.5 Future Investigations 16

1.6 Concluding Remarks 16

2 Estimation of Parameters in Markov Chain Transition Probabilities 17

2.1 Introduction 18

2.2 Simulating Markov Chain Trajectories 18

2.3 Three Cases of Parameter Dependence 20

2.4 Parameter Estimation, part I 22

2.5 Parameter Estimation, part II 26

2.6 Survey of Related Results 30

2.7 Future Investigations 33

2.8 Concluding Remarks 33

References 35

A MATLAB code 36

A.1 Code for Chapter 1 36

A.2 Code for Chapter 2 49

Chapter 1

Estimation in Higher-Order Linear Stochastic Systems

Consider a linear time-invariant system of order $d \geq 2$ that is subject to white noise perturbation. The input and output of this stochastic system are assumed to be sampled at regular time intervals, and using only these observations the first $(d-1)$ derivatives are approximated via a numerical differentiation scheme. In turn, these derivative approximations are used to produce a quadratic variation estimate of the white noise local variance.

Duncan *et al.* [2] have shown that the quadratic variation estimator is asymptotically biased when a forward-difference approach is used for numerically approximating the derivatives of the stochastic process, whereas the bias can be eliminated by instead applying a specific alternative numerical differentiation scheme. In this chapter, we provide numerical illustrations in support of this result. Additionally, we consider the result (also from [2]) that the straightforward discretization of a least squares estimation procedure for unknown parameters in the system leads to an asymptotically biased estimate, that this bias is a direct consequence of the inconsistency of the quadratic variation estimate of the white noise local variance, and that it can be eliminated either by the addition of a correction term when defining the estimator or by again incorporating a particular numerical scheme for derivative approximation.

1.1 Introduction

Let $(X(t), t \geq 0)$ be an $\mathbb{R}^n$-valued process that is the solution of the following stochastic differential equation of order $d \geq 2$:

$$\begin{aligned} \mathrm{d}X^{(d-1)}(t) &= \left(\sum_{i=1}^{d} f_i X^{(i-1)}(t) + gU(t) \right) \mathrm{d}t + \mathrm{d}W(t) \\ X^{(i)}(0) &= X_0^{(i)} \end{aligned} \tag{1.1}$$

for $i = 0, \ldots, d-1$, where $t \geq 0$, $X^{(i)}(t) = (\mathrm{d}X^{(i-1)}/\mathrm{d}t)$ for $i = 1, 2, \ldots, d-1$, $X^{(0)}(t) = X(t)$, $(f_1, f_2, \ldots, f_d, g)$ are constant matrices, $U(t) \in \mathbb{R}^q$, $(W(t), t \geq 0)$ is an $\mathbb{R}^n$-valued Wiener process with local variance matrix h, that is, $\mathrm{d}W(t)\mathrm{d}W'(t) = h\mathrm{d}t$, and prime denotes matrix transposition. Initially, the $\mathbb{R}^q$-valued process $(U(t), t \geq 0)$ is the solution of the linear stochastic differential equation

$$\begin{aligned} \mathrm{d}U(t) &= cU(t)\mathrm{d}t + \mathrm{d}W_0(t) \\ U(0) &= U_0, \end{aligned} \tag{1.2}$$

where $t \geq 0$, c is a constant matrix, and $(W_0(t), t \geq 0)$ is an $\mathbb{R}^q$-valued Wiener process with local variance matrix h_0 that is independent of $(W(t), t \geq 0)$. The product $gU(t)$ in (1.1) may depend on only some components of $U(t)$, which are called relevant components and are denoted $U_{\mathrm{rel}}(t)$.

A first-order system of linear stochastic differential equations is obtained from (1.1) and (1.2) by defining the following vector and matrices in block form:

$$\mathbb{X}(t) = \begin{pmatrix} X(t) \\ X^{(1)}(t) \\ \vdots \\ X^{(d-1)}(t) \\ U(t) \end{pmatrix}, \tag{1.3}$$

$$F = \begin{pmatrix} 0 & I & 0 & \cdots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & I & 0 \\ f_1 & f_2 & \cdots & f_d & g \\ 0 & 0 & \cdots & 0 & c \end{pmatrix}, \tag{1.4}$$

$$H = \begin{pmatrix} 0 & \cdots & 0 \\ \vdots & h & 0 \\ 0 & 0 & h_0 \end{pmatrix}, \tag{1.5}$$

where I is the identity in $\mathbb{R}^n$ and the blocks in F and H correspond to the blocks in $\mathbb{X}$. Thus, (1.1) and (1.2) can be expressed as the system of first-order equations

$$\begin{aligned} d\mathbb{X}(t) &= F\mathbb{X}(t)dt + d\mathbb{W}(t) \\ \mathbb{X}(0) &= \mathbb{X}_0, \end{aligned} \tag{1.6}$$

where $t \geq 0$, $\mathbb{X}(t)$ is given by (1.3), F is the constant matrix (1.4), and $(\mathbb{W}(t), t \geq 0)$ is an $\mathbb{R}^{dn+q}$-valued Wiener process with local variance matrix H given by (1.5).

The following assumption is made on F.

(A1) F is a stable linear transformation; that is, its spectrum is contained in the open left half-plane.

If (A1) is satisfied, then $(\mathbb{X}(t), t \geq 0)$ has a limiting Gaussian distribution with zero mean and variance matrix R; that is $R = E[\mathbb{X}\mathbb{X}']$, where $\mathbb{X}$ is a random variable with the limiting distribution and R satisfies the Lyapunov equation

$$FR + RF' + H = 0. \tag{1.7}$$

The variance matrix R is partitioned into blocks that correspond to the block components of $\mathbb{X}(t)$ as follows:

$$R = (r_{ij})$$

for $i, j \in \{1, \dots, d+1\}$, where $r_{ij} = E[X^{(i)} X^{(j)\prime}]$ for $i, j \in \{1, \dots, d\}$, $r_{i,d+1} = E[X^{(i)} U'] = r'_{d+1,i}$ for $i \in \{1, \dots, d\}$, $r_{d+1,d+1} = E[UU']$, and E is expectation with respect to the limiting distribution. The assumption (A1) of the stability of F ensures the validity of applications of the Law of Large Numbers in proofs of the subsequent results.

It is assumed that there are discrete observations of $(X(t), t \geq 0)$ and $(U(t), t \geq 0)$ with the uniform sampling interval $\delta > 0$. This sampling yields the following random variables:

$$(X(m\delta), U_{\mathrm{rel}}(m\delta),\ m = 0, 1, \dots, N + d). \tag{1.8}$$

1.1.1 Forward Difference Approach

Initially, the derivatives $(X^{(i)}(m\delta),\ m = 0, 1, \dots, N + d - 1 - i)$ are approximated by the forward differences

$$X^{(i)}_{m,\delta} = \left(X^{(i-1)}_{m+1,\delta} - X^{(i-1)}_{m,\delta} \right) / \delta \tag{1.9}$$

for $i = 1, 2, \dots, d-1$. For subsequent notational convenience, let $X^{(0)}_{m,\delta} = X(m\delta)$ for $m = 0, 1, \dots, N + d - 1$. Since $X^{(i)}_{m,\delta}$ is not $X^{(i)}(m\delta)$, the ith derivative of $(X(t), t \geq 0)$, a bias for some asymptotic computations is introduced that does not converge to zero as δ tends to zero.

The well-known quadratic variation formula for (1.1) is

$$\lim_{\delta \to 0} \frac{1}{T} \sum_{m=0}^{[T/\delta]} \left(X^{(d-1)}((m+1)\delta) - X^{(d-1)}(m\delta) \right) \left(X^{(d-1)}((m+1)\delta) - X^{(d-1)}(m\delta) \right)' = h, \tag{1.10}$$

where $T > 0$ is fixed and the limit can be taken in $L^2(P)$, i.e. in quadratic mean. The family of random variables on the left-hand side of (1.10) suggests the following family of estimates for h:

$$h^*(N, \delta) = \frac{1}{N\delta} \sum_{m=0}^{N-1} \left(X^{(d-1)}_{m+1,\delta} - X^{(d-1)}_{m,\delta} \right) \left(X^{(d-1)}_{m+1,\delta} - X^{(d-1)}_{m,\delta} \right)', \tag{1.11}$$

where $N \in \mathbb{N}$ and $\delta > 0$.

The following proposition shows that the family of estimates $(h^*(N, \delta), N \in \mathbb{N}, \delta > 0)$

does not converge to h as $N \to \infty$ and $\delta \to 0$, but rather to $C(d)h$, where $C(d)$ is a nontrivial, explicit constant that depends only on the order d of the system.

Proposition 1.1.1. *Assume that (A1) is satisfied. Let $(X^{(d-1)}(t), t \geq 0)$ satisfy* (1.1)*, and let $h^*(N,\delta)$ for $N \in \mathbb{N}$ and $\delta > 0$ be given by* (1.11)*. The following equality is satisfied:*

$$\lim_{\delta\to 0}\lim_{N\to\infty} h^*(N,\delta) = C(d)h \quad a.s., \tag{1.12}$$

where

$$C(d) = \frac{(-1)^d}{(2d-1)!}\sum_{j=1}^{d}(-1)^j j^{2d-1}\binom{2d}{d-j} \tag{1.13}$$

for $d = 2, 3, \ldots$.

Assume now that (1.1) contains a p-dimensional unknown parameter $\alpha = (\alpha^1, \ldots, \alpha^p)$, so that it may be written

$$\mathrm{d}X^{(d-1)}(t) = \left(\sum_{i=1}^{d} f_i(\alpha)X^{(i-1)}(t) + g(\alpha)U(t)\right)\mathrm{d}t + \mathrm{d}W(t), \tag{1.14}$$

where

$$f_i(\alpha) = f_{i0} + \sum_{j=1}^{p}\alpha^j f_{ij}$$

for $i = 1, \ldots, d$,

$$g(\alpha) = g_0 + \sum_{j=1}^{p}\alpha^j g_j,$$

and $(f_{ij}, i \in \{1,\ldots,d\}, j \in \{0,\ldots,p\})$, $(g_j, j \in \{1,\ldots,p\})$ are known fixed matrices. With an unknown parameter in (1.1), the notion of U_{rel} is extended to the linear span of g_jU for $j = 0,\ldots,p$. Let α_0 denote the true value of the parameter in (1.14). It is assumed that (A1) is satisfied with $f_i = f_i(\alpha_0)$ for $i = 1,2,\ldots,d$ and $g = g(\alpha_0)$. The least-squares estimate of α_0 is obtained from the observations $(X(t), t \in [0,T])$ by minimizing the quadratic functional

$$\int_0^T \left[\left(X^{(d)} - \sum_{i=1}^{d} f_i(\alpha)X^{(i-1)} - g(\alpha)U\right)' \ell \left(X^{(d)} - \sum_{i=1}^{d} f_i(\alpha)X^{(i-1)} - g(\alpha)U\right) - X^{(d)\prime}\ell X^{(d)}\right]\mathrm{d}t, \tag{1.15}$$

where ℓ is a positive, semidefinite matrix. In (1.15), the undefined term $X^{(d)\prime}\ell X^{(d)}$ is cancelled, and $X^{(d)}\mathrm{d}t$ is defined as $\mathrm{d}X^{(d-1)}$. The minimization of (1.15) yields the following family of equations for the least-squares estimate, $\alpha^*(T) = (\alpha^{*1}(T), \ldots, \alpha^{*P}(T))$, of α_0:

$$\begin{aligned}
&\sum_{k=1}^{p} \frac{1}{T} \int_0^T \left(\sum_{i=1}^{d} f_{ij} X^{(i-1)} - g_j U \right)' \ell \left(\sum_{i=1}^{d} f_{ik} X^{(i-1)} + g_k U \right) \mathrm{d}t\, \alpha^{*k}(T) \\
&= \frac{1}{T} \int_0^T \left(\sum_{i=1}^{d} f_{ij} X^{(i-1)} + g_j U \right)' \ell \left(\mathrm{d}X^{(d-1)} - \left(\sum_{i=1}^{d} f_{i0} X^{(i-1)} + g_0 U \right) \mathrm{d}t \right)
\end{aligned} \tag{1.16}$$

for $j = 1, 2, \ldots, p$. Using (1.14) with $\alpha = \alpha_0$, (1.16) can be rewritten as

$$\begin{aligned}
&\sum_{k=1}^{p} \frac{1}{T} \int_0^T \left(\sum_{i=1}^{d} f_{ij} X^{(i-1)} - g_j U \right)' \ell \left(\sum_{i=1}^{d} f_{ik} X^{(i-1)} + g_k U \right) \mathrm{d}t \left(\alpha^{*k}(T) - \alpha_0^k \right) \\
&= \frac{1}{T} \int_0^T \left(\sum_{i=1}^{d} f_{ij} X^{(i-1)} + g_j U \right)' \ell \mathrm{d}W
\end{aligned} \tag{1.17}$$

for $j = 1, 2, \ldots p$. If (A1) is satisfied, then

$$\lim_{T\to\infty} \frac{1}{T} \int_0^T \left(\sum_{i=1}^{d} f_{ij} X^{(i-1)} - g_j U \right)' \ell \left(\sum_{i=1}^{d} f_{ik} X^{(i-1)} + g_k U \right) \mathrm{d}t = \mathrm{tr}(F_j' \ell F_k R),$$

where $\mathrm{tr}(\cdot)$ is the trace, $F_j = (f_{1j}, \ldots, f_{dj}, g_j)$ for $j = 1, \ldots, p$, and R satisfies (1.7).

The following assumption is used subsequently.

(A2) The matrix $Q := (\mathrm{tr}(F_j' \ell F_k R))$ for $j, k \in \{1, \ldots, p\}$ is nonsingular.

Since the right-hand side of (1.17) converges to zero a.s. as $T \to \infty$, it follows from (A2) that

$$\lim_{T\to\infty} \alpha^*(T) = \alpha_0 \quad \text{a.s.},$$

so the family of least-squares estimates $(\alpha^*(T), T > 0)$ is strongly consistent.

Now it is assumed that instead of a continuous observation of the state $(X(t), t \geq 0)$ and the input $(U(t), t \geq 0)$, there is only (1.8) from which (1.9) is computed. Equation

(1.16) is replaced by the discrete analog:

$$\frac{1}{N\delta}\sum_{k=1}^{p}\delta\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}X_{m,\delta}^{(i-1)} - g_jU(m\delta)\right)'\ell\left(\sum_{i=1}^{d} f_{ik}X_{m,\delta}^{(i-1)} + g_kU(m\delta)\right)\hat{\alpha}_{N\delta}^{k}$$
$$= \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}X_{m,\delta}^{(i-1)} + g_jU(m\delta)\right)'\ell\left(\left(X_{m+1,\delta}^{(d-1)} - X_{m,\delta}^{(d-1)}\right) - \delta\left(\sum_{i=1}^{d} f_{i0}X_{m,\delta}^{(i-1)} + g_0U(m\delta)\right)\right) \tag{1.18}$$

for $j = 1, 2, \ldots, p$, where $\hat{\alpha}_{N\delta} = \left(\hat{\alpha}_{N\delta}^{1}, \ldots, \hat{\alpha}_{N\delta}^{p}\right)$ is an estimate of α_0 and is the solution of (1.18).

The next proposition establishes that the family of least-squares estimates given by (1.18) is asymptotically biased, whereas the family of estimates given by the following modified discrete least squares equations is consistent:

$$\frac{1}{N\delta}\sum_{k=1}^{p}\delta\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}X_{m,\delta}^{(i-1)} - g_jU(m\delta)\right)'\ell\left(\sum_{i=1}^{d} f_{ik}X_{m,\delta}^{(i-1)} + g_kU(m\delta)\right)\hat{\alpha}_{N\delta}^{k}$$
$$= \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}X_{m,\delta}^{(i-1)} + g_jU(m\delta)\right)'\ell\left(\left(X_{m+1,\delta}^{(d-1)} - X_{m,\delta}^{(d-1)}\right) - \delta\left(\sum_{i=1}^{d} f_{i0}X_{m,\delta}^{(i-1)} + g_0U(m\delta)\right)\right)$$
$$-\frac{1}{N\delta}\frac{C(d)-1}{2C(d)}\sum_{m=0}^{N-1}\left(X_{m+1,\delta}^{(d-1)} - X_{m,\delta}^{(d-1)}\right)' f_{dj}'\,\ell\left(X_{m+1,\delta}^{(d-1)} - X_{m,\delta}^{(d-1)}\right) \tag{1.19}$$

for $j = 1, 2, \ldots, p$, where $C(d)$ is given by (1.13).

Proposition 1.1.2. *Assume that (A1) and (A2) are satisfied for $\alpha = \alpha_0$. For $N \in \mathbb{N}$ and $\delta > 0$, let $\hat{\alpha}_{N\delta} = \left(\hat{\alpha}_{N\delta}^{1}, \ldots, \hat{\alpha}_{N\delta}^{p}\right)$ be the solution of* (1.19)*. Then*

$$\lim_{\delta\to 0}\ \text{p-}\lim_{N\to\infty}\hat{\alpha}_{N\delta} = \alpha_0,$$

where p-lim$_{N\to\infty}\hat{\alpha}_{N\delta}$ *is the nonrandom limit in probability.*

1.1.2 Alternate Numerical Derivative Approximation

For the next propositions in [2], we consider the following numerical differentiation scheme:

$$\tilde{X}_{m,\delta}^{(i)} = X_{m,\delta}^{(i)} + A(d)\left(X_{m+1,\delta}^{(i)} - X_{m,\delta}^{(i)}\right) \tag{1.20}$$

for $i = 1, 2, \ldots, d-1$, where $X^{(i)}_{m,\delta}$ is given by (1.9). Let the estimate of h formed from $\left(\tilde{X}^{(d-1)}_{m,\delta}, m = 0, 1, \ldots, N-1\right)$ be denoted $\tilde{h}(N, \delta, A(d))$, that is

$$\tilde{h}(N,\delta,A(d)) = \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(\tilde{X}^{(d-1)}_{m+1,\delta} - \tilde{X}^{(d-1)}_{m,\delta}\right)\left(\tilde{X}^{(d-1)}_{m+1,\delta} - \tilde{X}^{(d-1)}_{m,\delta}\right)' \tag{1.21}$$

For suitable values of $A(d)$, the family of estimates $\left(\tilde{h}(N,\delta,A(d)), N \in \mathbb{N}, \delta > 0\right)$ is strongly consistent, as described in the following proposition.

Proposition 1.1.3. *Let (A1) be satisfied, let $B(d)$ be given by*

$$B(d) = \frac{(-1)^d}{(2d-1)!}\sum_{j=1}^{d+1}(-1)^j j^{2d-1}\binom{2d+2}{d+1-j}, \tag{1.22}$$

and let $A(d)$ be the positive root of the equation

$$a^2 + a + (C(d) - 1)/B(d) = 0. \tag{1.23}$$

Then the family of estimates $\left(\tilde{h}(N,\delta,A(d)), N \in \mathbb{N}, \delta > 0\right)$, where $\tilde{h}(N,\delta,A(d))$ is given by (1.21) is strongly consistent, that is

$$\lim_{\delta\to 0}\lim_{N\to\infty}\tilde{h}(N,\delta,A(d)) = h \quad a.s., \tag{1.24}$$

for $d = 2, 3, \ldots$.

Now, the discretized version of (1.16) using the numerical differentiation rule (1.20) is given by

$$\begin{aligned}\frac{1}{N\delta}\sum_{k=1}^{p}\delta\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}\tilde{X}^{(i-1)}_{m,\delta} - g_j U(m\delta)\right)'\ell\left(\sum_{i=1}^{d} f_{ik}\tilde{X}^{(i-1)}_{m,\delta} + g_k U(m\delta)\right)\tilde{\alpha}^k_{N\delta} \\ = \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(\sum_{i=1}^{d} f_{ij}\tilde{X}^{(i-1)}_{m,\delta} + g_j U(m\delta)\right)'\ell\left(\left(\tilde{X}^{(d-1)}_{m+1,\delta} - \tilde{X}^{(d-1)}_{m,\delta}\right) - \delta\left(\sum_{i=1}^{d} f_{i0}\tilde{X}^{(i-1)}_{m,\delta} + g_0 U(m\delta)\right)\right)\end{aligned} \tag{1.25}$$

for $j = 1, 2, \ldots, p$, where $\tilde{\alpha}_{N\delta} = \left(\tilde{\alpha}^1_{N\delta}, \ldots, \tilde{\alpha}^p_{N\delta}\right)$ is an estimate of α_0 and is the solution of (1.25). The final proposition asserts that the least squares estimator $\tilde{\alpha}_{N\delta}$ is consistent, without need for the addition of any correction term in the equation above.

Proposition 1.1.4. *Assume that (A1) and (A2) are satisfied for* $\alpha = \alpha_0$*. For* $N \in \mathbb{N}$ *and* $\delta > 0$*, let* $\tilde{\alpha}_{N\delta} = \left(\tilde{\alpha}^1_{N\delta}, \dots, \tilde{\alpha}^p_{N\delta}\right)$ *be the solution of* (1.25)*. Then, for* $d \geq 2$*,*

$$\lim_{\delta \to 0} \operatorname*{p-lim}_{N \to \infty} \tilde{\alpha}_{N\delta} = \alpha_0.$$

1.2 Simulating Sampled SDE Solutions

Before illustrating any estimation procedures, we first numerically simulate the sampled observations of the continuous-time scalar-valued process $(X(t), t \geq 0)$ by applying the Euler-Maruyama method to (1.6), as described in [4]. Supposing that $\alpha_0 = 1$ is the true value of the unknown parameter, we choose $f_{10} = f_{21} = -2$ and $f_{11} = f_{20} = 1$, such that $f_1(\alpha) = f_2(\alpha) = -1$ and the matrix F satisfies assumption (A1).

For the purpose of numerical illustration, the following simplifying assumptions are made to hold throughout the remainder of this chapter. Fix $d = 2$, $n = 1$, $p = 1$, and $U(t) \equiv 0$. Then (1.14) becomes

$$\mathrm{d}X^{(1)}(t) = \left(f_1(\alpha)X(t) + f_2(\alpha)X^{(1)}(t)\right)\mathrm{d}t + \mathrm{d}W(t), \tag{1.26}$$

where $f_1(\alpha) = f_{10} + f_{11}\alpha$ and $f_2(\alpha) = f_{20} + f_{21}\alpha$ for known fixed scalars $(f_{10}, f_{11}, f_{20}, f_{21})$. Moreover, this second-order linear SDE can be written as a first-order system of linear SDEs by simplifying (1.3) - (1.5), respectively, as follows:

$$\mathbb{X}(t) = \begin{pmatrix} X(t) \\ X^{(1)}(t) \end{pmatrix} \tag{1.27}$$

$$F = \begin{pmatrix} 0 & 1 \\ f_1(\alpha) & f_2(\alpha) \end{pmatrix} \tag{1.28}$$

$$H = \begin{pmatrix} 0 & 0 \\ 0 & h \end{pmatrix}. \tag{1.29}$$

The system (1.26) is then equivalent to (1.6), where $t \geq 0$, $\mathbb{X}(t)$ is given by (1.27), F is the constant matrix (1.28), and $(\mathbb{W}(t), t \geq 0)$ is an $\mathbb{R}^2$-valued Wiener process with local variance matrix H given by (1.29).

Let $\mu >> 1$ be the factor by which the discretization increment of the estimation procedure varies from the discretization increment of the SDE solution. That is, the increment δ will be used in computing the numerical derivative approximations, so to simulate observations of a continuous process we employ the Euler-Maruyama recursion with the smaller increment $\rho := \delta/\mu$. Here, we select $\mu = 100$.

Let η denote the number of process values that will be used to compute the derivative approximations. In particular, $\eta := N + d$ for the forward difference approach, while $\eta := N + d + 1$ in the case of the alternate numerical differentiation method. Then $M := \mu(\eta - 1)$ is the required number of simulated observations of the process via the Euler-Maruyama method.

Fix the initial value of the process to be $\mathbb{X}_0 = (0, 1)'$, and let $\eta_{m,\rho h} \in N(0, \rho h)$ for $m = 0, 1, \ldots, M - 1$ denote normal random values with mean zero and variance ρh. Then the Euler-Maruyama recursion for the process $(\mathbb{X}(t), t \geq 0)$ is given by

$$\mathbb{X}((m+1)\rho) = \mathbb{X}(m\rho) + F\mathbb{X}(m\rho)\rho + \Delta W_{m,\rho h}$$

for $m = 0, 1, \ldots, M - 1$, where

$$\Delta W_{m,\rho h} = \begin{pmatrix} 0 \\ \eta_{m,\rho h} \end{pmatrix}.$$

We assume that only the process $(X(t), t \geq 0)$ is observed; hence the simulated second component of $(\mathbb{X}(t), t \geq 0)$ is disregarded and will be replaced by one of the numerical derivative approximations described above. The MATLAB code for the Euler-Maruyama method is given in Appendix A.1.1.

1.3 Estimating White Noise Local Variance

With the simplifications described in the preceding section, (1.11) becomes

$$h^*(N,\delta) = \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(X^{(1)}_{m+1,\delta} - X^{(1)}_{m,\delta}\right)^2. \tag{1.30}$$

The value $C(2) = 2/3$, given in [2], is easily computed directly from (1.13). The MATLAB code for a function that returns the value of any $C(d)$, $d \geq 2$, is given in Appendix A.1.4, while the forward difference derivative approximations

$$X^{(1)}_{m,\delta} = \left(X_{m+1,\delta} - X_{m,\delta}\right)/\delta, \qquad m = 0,\ldots,N \tag{1.31}$$

are produced by the code in Appendix A.1.2.

Suppose that $h = 1$ is the true value of the local variance of the scalar-valued Wiener process $(W(t), t \geq 0)$. Given the simulated sampled observations of $(\mathbb{X}(t), t \geq 0)$, as described in Section 1.2, the estimation of h is performed by explicitly calculating the value of $h^*(N,\delta)$ for large N and small $\delta > 0$. Performing this computation within nested loops corresponding to $\lim_{N\to\infty}$ and $\lim_{\delta\to 0}$, respectively, we obtain a mesh of values that are seen to converge to the value $C(2)h = 2/3$ (see Figure 1.1), which supports Proposition 1.1.1 along with the numerical results provided in [3]. The MATLAB code for this estimation procedure appears in Appendix A.1.7.

N δ	10^1	10^2	10^3	10^4
10^{-1}	0.98	0.66	0.60	0.66
10^{-2}	0.32	0.73	0.71	0.67
10^{-3}	0.79	0.66	0.65	0.66
10^{-4}	0.37	0.58	0.68	0.67

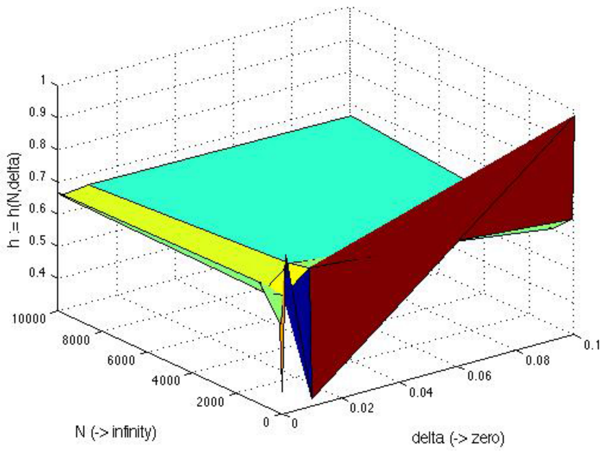

Figure 1.1: The estimator $h^*(N,\delta)$ converges to the value $C(2)h = 2/3$, as predicted.

We now proceed to estimate h via the alternate numerical differentiation method described in Section 1.1.2. With the assumed simplifications, (1.21) becomes

$$\tilde{h}(N,\delta,A(d)) = \frac{1}{N\delta}\sum_{m=0}^{N-1}\left(\tilde{X}^{(1)}_{m+1,\delta} - \tilde{X}^{(1)}_{m,\delta}\right)^2. \tag{1.32}$$

The value $B(2) = 1$ is easily computed directly from (1.22). Given $C(2) = 2/3$ as above, $A(2)$ is the positive root of (1.23) with $d = 2$, i.e.

$$A(2) = \frac{-1+\sqrt{1-4(C(2)-1)/B(2)}}{2} = \frac{-1+\sqrt{1+4/3}}{2} \approx 0.2638.$$

The MATLAB code for functions that compute any $B(d)$ and $A(d)$, $d \geq 2$, is given in Appendices A.1.5 and A.1.6, respectively.

If we assume that the alternate numerical derivative approximations $\left(\tilde{X}^{(1)}_{m,\delta}, m = 0,\dots,N\right)$ are given by (1.20) with $i = 1$, then using (1.32) to produce an array of $\tilde{h}(N,\delta,A(d))$-values for large N and small $\delta > 0$, we repeatedly observe[1] convergence to $h/2$, rather than to h. Consequently, we use the modified approximations

$$\tilde{X}^{(1)}_{m,\delta} = \sqrt{2}\cdot\left[X^{(1)}_{m,\delta} + A(d)\left(X^{(1)}_{m+1,\delta} - X^{(1)}_{m,\delta}\right)\right], \tag{1.33}$$

where $\left(X^{(1)}_{m,\delta}, m = 0,\dots,N\right)$ is still the collection of forward differences given by (1.31). With the included correction factor of $\sqrt{2}$, we obtain a mesh of values that are seen to converge to the value $h = 1$ (see Figure 1.2). The MATLAB code for producing $\left(\tilde{X}^{(1)}_{m,\delta}, m = 0,\dots,N\right)$ is given in Appendix A.1.3, while the code for the estimation procedure appears in Appendix A.1.8.

1.4 Estimating an Unknown Parameter

We now wish to illustrate the estimation of the unknown parameter α. With the simplifications described in Section 1.2 and the assumption that $\ell = 1$, (1.19) reduces to the

[1]Here, only the case $h = 1$ is illustrated, but the MATLAB code was tested for a variety of scalar h-values, i.e. assuming $h = 0.5$ or $h = 2$ to be the true value instead of $h = 1$.

δ \ N	10^1	10^2	10^3	10^4
10^{-1}	0.82	0.93	0.96	0.93
10^{-2}	0.81	1.08	1.02	0.94
10^{-3}	0.77	0.71	0.95	0.94
10^{-4}	1.34	0.95	0.96	0.98

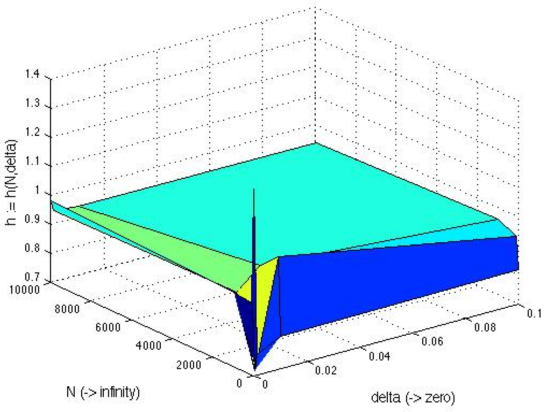

Figure 1.2: The estimator $\tilde{h}(N, \delta, A(d))$ converges to $h = 1$ when the $\sqrt{2}$ factor is used.

single equation

$$\begin{aligned}
&\frac{1}{N}\sum_{m=0}^{N-1}\left(f_{11}X(m\delta)+f_{21}X^{(1)}_{m,\delta}\right)^2\hat{\alpha}^k_{N\delta} \\
&=\frac{1}{N\delta}\sum_{m=0}^{N-1}\left(f_{11}X(m\delta)+f_{21}X^{(1)}_{m,\delta}\right)\left(\left(X^{(1)}_{m+1,\delta}-X^{(1)}_{m,\delta}\right)-\delta\left(f_{10}X(m\delta)+f_{20}X^{(1)}_{m,\delta}\right)\right) \qquad (1.34)\\
&-\frac{1}{N\delta}\frac{C(2)-1}{2C(2)}\sum_{m=0}^{N-1}f_{21}\left(X^{(1)}_{m+1,\delta}-X^{(1)}_{m,\delta}\right)^2.
\end{aligned}$$

To solve for the estimate $\hat{\alpha}_{N\delta}$ in this scalar case, it is sufficient to compute the ratio S_1/S_2, where S_1 is the right-hand side of (1.34) and

$$S_2 := \frac{1}{N}\sum_{m=0}^{N-1}\left(f_{11}X(m\delta)+f_{21}X^{(1)}_{m,\delta}\right)^2.$$

As before, the sampled observations of the process $(X(t), t \geq 0)$ are simulated via the Euler-Maruyama recursion, while the forward difference derivative approximations $\left(X^{(1)}_{m,\delta}, m = 0, \ldots, N\right)$ are given by (1.31). The values of S_1 and S_2 are computed explicitly, and their ratio is given as the estimated value of α for a particular N and δ.

The above estimate calculation is repeated within nested loops corresponding to $\lim_{N\to\infty}$ and $\lim_{\delta\to 0}$, respectively, and the MATLAB code for the entire procedure is provided in Appendix A.1.9. Figures 1.3 and 1.4 illustrate two successive executions of this code. While the former appears to imply the consistency of the family of estimates $(\hat{\alpha}_{N,\delta}, N \in \mathbb{N}, \delta > 0)$, the later does not. Thus, the numerical results do not presently support the claim of Proposition 1.1.2. We remark that omitting the correction term, denoted by `COR` in the MATLAB code, does not alter the behavior of the simulated results.

δ \ N	10^1	10^2	10^3	10^4
10^{-1}	0.96	0.73	0.74	0.76
10^{-2}	0.90	0.17	0.74	0.70
10^{-3}	-1.08	1.68	0.46	0.66
10^{-4}	10.6	-5.58	-0.50	1.06

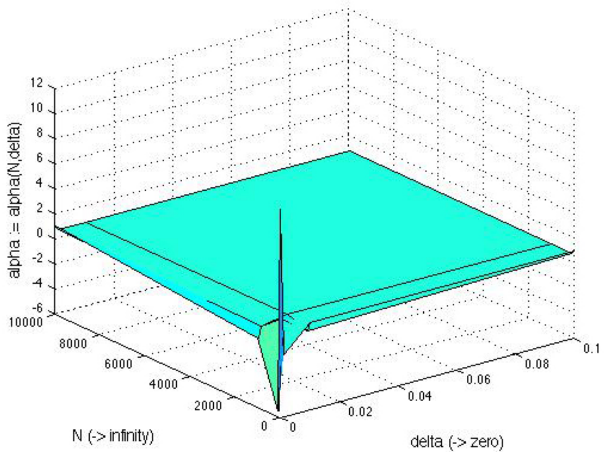

Figure 1.3: The corrected estimator $\hat{\alpha}_{N,\delta}$ appears to converge to the true value $\alpha_0 = 1$.

δ \ N	10^1	10^2	10^3	10^4
10^{-1}	0.38	0.94	0.79	0.76
10^{-2}	8.36	0.93	0.82	0.78
10^{-3}	-2.05	2.35	1.70	0.75
10^{-4}	2.10	6.11	1.77	0.44

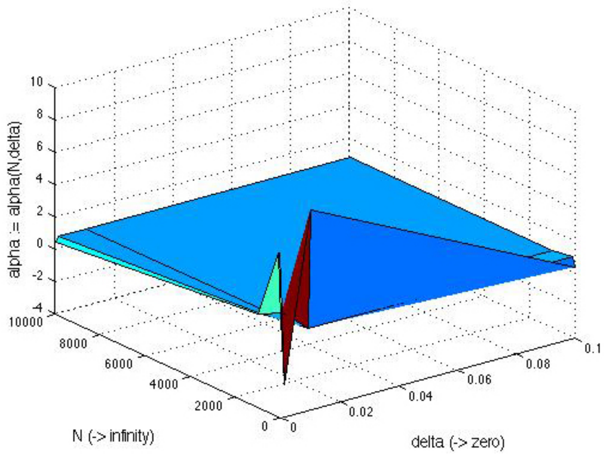

Figure 1.4: The corrected estimator $\hat{\alpha}_{N,\delta}$ does not converge to the true value $\alpha_0 = 1$.

Applying the alternate numerical differentiation scheme (1.20), with or without the $\sqrt{2}$ factor discussed in Section 1.3, provides similarly inconclusive results. In particular, Figures 1.5 and 1.6 illustrate that consistency of the family of estimates $(\tilde{\alpha}_{N,\delta}, N \in \mathbb{N}, \delta > 0)$ is sometimes observed and sometimes not observed, respectively. Thus, the numerical results do not presently support the claim of Proposition 1.1.4. The MATLAB code for this alternate estimation procedure is given in Appendix A.1.10.

N δ	10^1	10^2	10^3	10^4
10^{-1}	1.51	0.69	0.81	0.77
10^{-2}	2.22	1.10	0.61	0.77
10^{-3}	9.39	-0.03	1.74	0.80
10^{-4}	3.74	3.49	-0.42	1.01

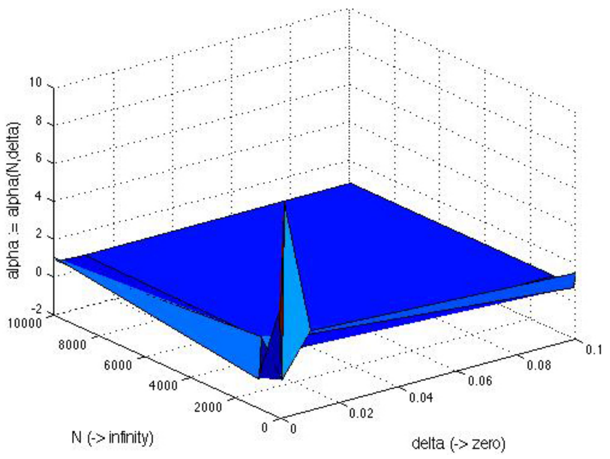

Figure 1.5: The estimator $\tilde{\alpha}_{N,\delta}$ appears to converge to the true value $\alpha_0 = 1$.

N δ	10^1	10^2	10^3	10^4
10^{-1}	2.08	1.04	0.76	0.77
10^{-2}	3.27	1.59	0.73	0.70
10^{-3}	-3.74	1.36	1.36	0.73
10^{-4}	3.25	5.72	1.92	1.87

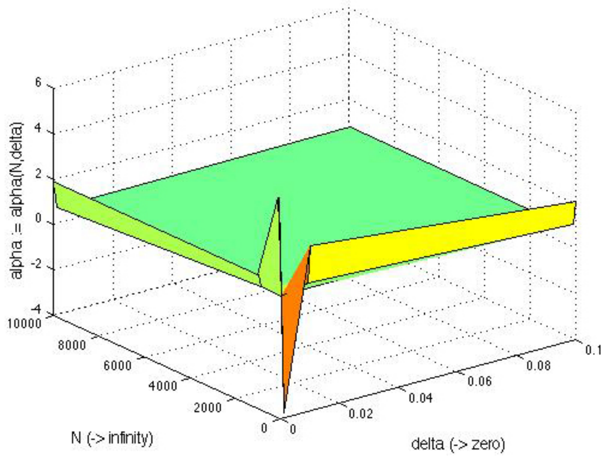

Figure 1.6: The estimator $\tilde{\alpha}_{N,\delta}$ does not converge to the true value $\alpha_0 = 1$.

1.5 Future Investigations

It remains to understand why the simulated quadratic variation estimates do not converge to the true value of the white noise local variance when the alternate numerical differentiation approach is used, as described in Section 1.3. In particular, why the correction factor of $\sqrt{2}$ is needed in order to observe the strong consistency of the estimates guaranteed by Proposition 1.1.3. Additionally, we would like reconcile the discrepancy between the theoretical results of Propositions 1.1.2 and 1.1.4 and the numerical simulations presented in Section 1.4.

1.6 Concluding Remarks

The numerical illustrations above support the result that a precise asymptotic bias is observed in the quadratic variation estimation of the local variance h when forward differences are used to approximate the derivative of the continuous-time scalar-valued process $(X(t), t \geq 0)$ that is the solution of the stochastic differential equation (1.1). On the other hand, in the case of the alternate numerical differentiation scheme, the simulated estimates $\tilde{h}(N, \delta, A(d))$ were observed to converge to $h/2$, rather than to the true value h, barring the insertion of a correction factor of $\sqrt{2}$.

Chapter 2

Estimation of Parameters in Markov Chain Transition Probabilities

Consider a controlled Markov chain, taking values on a finite state space, whose transition probabilities are assumed to depend on an unknown parameter belonging to a compact set. Our goal is to numerically analyze the long-term behavior of a sequence of maximum likelihood estimates of this unknown parameter. Motivated by the work of both Sagalovksy and Pasik-Duncan, we consider one- and two-dimensional parameter cases. We further distinguish between cases where the transition probabilities depend linearly on the parameter and cases where they depend quadratically.

Mandl [6] has shown that, under a specific identifiability condition, the maximum likelihood estimator is strongly consistent. Similarly, Borkar and Varaiya [1] have shown that that under alternative assumptions the sequence maximum likelihood estimates converges and retains desirable (although not ideal) property relating to the Markov chain's transition probabilities. In this chapter, we provide numerical illustrations in support of both of these results. Additionally, we present a survey of related results by Kumar [5], Sagalovsky [8], and Pasik-Duncan [7].

2.1 Introduction

Consider a Markov chain, $(X_n, n = 0, 1, \dots)$, taking values on the finite state space $S = \{1, 2, \dots, s\}$. The state transition probabilities $\mathbb{P}(X_{n+1} = j \,|\, X_n = i)$ are assumed to depend on both an unknown parameter, α, belonging to the compact set A, and a control action, $(u_n, n = 1, 2, \dots)$, belonging to the finite set U. Thus, we denote these probabilities by

$$p(i, j; u_n, \alpha) := \mathbb{P}(X_{n+1} = j \,|\, X_n = i), \qquad n = 0, 1, \dots.$$

At each time n, the maximum likelihood estimate of α is given by $\hat{\alpha}_n := \operatorname{argmax}_{\alpha \in A} L_n(\alpha)$, where $L_n(\alpha) := \sum_{m=0}^{n-1} \log p(x_m, x_{m+1}; u_m, \alpha)$ is the control-dependent log-likelihood function. An important question is whether or not the sequence $(\hat{\alpha}_n, n = 1, 2, \dots)$ converges and, if it does, whether or not it converges to the true value of α. We will see that under different assumptions, the maximum likelihood estimator may or may not be consistent.

First, the simulation of trajectories of a general finite-state Markov chain is addressed in Section 2.2. We identify three distinct cases of parameter dependence in Section 2.3: linear dependance on a one-dimensional parameter, quadratic dependence on a one-dimensional parameter, and linear dependence on a two-dimensional parameter. In Section 2.4, we demonstrate numerical simulations that support a result by Mandl [6]. Similarly, in Section 2.5, we exhibit both simulations and counter-examples to concretely illustrate some of the work of Borkar and Varaiya [1]. Finally, we present a survey of related results by Kumar [5], Sagalovsky [8], and Pasik-Duncan [7] in Section 2.6.

2.2 Simulating Markov Chain Trajectories

We begin by addressing the general problem of using MATLAB to numerically simulate an observed trajectory of the Markov chain $(X_n, n = 0, 1, \dots)$. To this end, we assume knowledge of the true value, α^0, of the unknown parameter α. Initially, we suppose that the control, u, is identically zero. Then, at the end of this section, we discuss the case involving a non-trivial feedback control action.

For simplicity, we let $P = (P(i,j))$ denote the constant (since α^0 is assumed to be known and $u \equiv 0$) transition probability matrix of the Markov chain. When $S = \{1,2\}$, simulating successive states of the chain is particularly straightforward. Indeed, a trajectory $x_0, x_1, \ldots$ can be generated in MATLAB through a weighted coin-flip procedure as follows.

We describe only the initial iteration of the algorithm, as subsequent iterations follow analogously by the memoryless Markov property. Fix $x_0 \in S$. First, a number r is selected at random from the uniform distribution on the interval $[0,1]$. If $0 \leq r \leq P(x_0,1) \leq 1$, then set $x_1 = 1$; otherwise, set $x_1 = 2$. Repeating this simple procedure, we produce a Markov chain trajectory $x_0, x_1, \ldots$ in accordance with the transition probability matrix P.

We now extend the above algorithm to treat the case of a Markov chain with the arbitrary finite state space $S = \{1,2,\ldots,s\}$. As before, it suffices to describe only the initial iteration of the algorithm. Fix $x_0 \in S$. First, a number r is selected at random from the uniform distribution on the interval $[0,1]$. The idea is then to divide the interval $[0,1]$ into s subintervals whose lengths are equal to $P(x_0,1), P(x_0,2), \ldots, P(x_0,s)$, and to select the state x_1 according to the subinterval in which r lies. This task is easily accomplished in MATLAB via cumulative row-sums of the transition probability matrix.

The generalized algorithm described above is formalized in Appendices A.2.1 and A.2.2. In particular, two variations of the code are provided: the first is optimized to require minimal processing power, while the second is optimized to require minimal memory. The distinction is subtle but important from a numerical standpoint; however, the relative computational advantage of each case compared to the other is negligible in the applications that are considered here.

Suppose now, as is relevant to the maximum likelihood estimation problems considered throughout the remainder of this chapter, that the transition probabilities of the Markov chain depend on a non-trivial feedback control action of the form $u_n = \phi(\hat{\alpha}_n, x_n)$. In order to simulate an observed trajectory of the chain in such a case, it suffices to modify the method in which the code in Appendix A.2.1 or A.2.2 is applied (rather than to modify the code itself). In particular, at each time n, the transition probability matrix depends on the control, u_n. The code uses this fixed matrix, along with the known value of the present state, x_n, to produce the immediately following state, x_{n+1}. Then, the next maximum likelihood estimate $\hat{\alpha}_{n+1}$ is computed, the next control u_{n+1} is determined, and the procedure repeats.

2.3 Three Cases of Parameter Dependence

Motivated by the work of both Sagalovsky [8] and Pasik-Duncan [7], we now describe the three distinct cases of parameter dependence that will be considered throughout the numerical simulations in Sections 2.4 and 2.5. For simplicity, we henceforth consider a two-state Markov chain by fixing $s = 2$, although the following results hold for an arbitrary positive integer s.

2.3.1 Case I: linear dependence, 1D parameter.

Let $\alpha^0 = 0.02$ be the true value of the one-dimensional unknown parameter α, and consider a Markov chain whose transition probability matrix, $P = (p(i,j;u,\alpha))$, is defined by

$$P := \begin{array}{c} \\ 1 \\ 2 \end{array} \begin{array}{c} \begin{array}{cc} \qquad 1 \qquad\qquad & \qquad 2 \qquad \end{array} \\ \begin{pmatrix} (u-2)\alpha + 0.5 & (2-u)\alpha + 0.5 \\ 1 & 0 \end{pmatrix} \end{array}.$$

Thus, the transition probabilities depend linearly on α; in particular, they have the form

$$p(i,j;u,\alpha) = a(i,j;u)\alpha + b(i,j;u),$$

for $i,j = 1,2$, where $a = (a(i,j;u))$ and $b = (b(i,j;u))$ are the matrices given by

$$a := \begin{pmatrix} u-2 & 2-u \\ 0 & 0 \end{pmatrix}, \quad b := \begin{pmatrix} 0.5 & 0.5 \\ 1 & 0 \end{pmatrix}.$$

2.3.2 Case II: quadratic dependence, 1D parameter.

Again, let $\alpha^0 = 0.02$ be the true value of the one-dimensional unknown parameter α. We now consider a Markov chain whose transition probability matrix, $P = (p(i,j;u,\alpha))$, is defined by

$$P := \begin{array}{c c} & \begin{array}{cc} 1 & 2 \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} (u-2)\alpha^2 + (u-2)\alpha + 0.5 & (2-u)\alpha^2 + (2-u)\alpha + 0.5 \\ 1 & 0 \end{pmatrix} \end{array}.$$

Thus, the transition probabilities depend quadratically on α; in particular, they have the form

$$p(i,j;u,\alpha) = a(i,j;u)\alpha^2 + b(i,j;u)\alpha + c(i,j;u),$$

for $i,j = 1,2$, where $a = (a(i,j;u))$, $b = (b(i,j;u))$, and $c = (c(i,j;u))$ are the matrices given by

$$a := \begin{pmatrix} u-2 & 2-u \\ 0 & 0 \end{pmatrix}, \quad b := \begin{pmatrix} u-2 & 2-u \\ 0 & 0 \end{pmatrix}, \quad c := \begin{pmatrix} 0.5 & 0.5 \\ 1 & 0 \end{pmatrix}.$$

2.3.3 Case III: linear dependence, 2D parameter.

Finally, suppose that $\alpha^0 = (0.01, -0.02)$ is the true value of the two-dimensional unknown parameter $\alpha = (\alpha_1, \alpha_2)$. We now consider a Markov chain whose transition probability matrix $P = (p(i,j;u,\alpha))$ is defined by

$$P := \begin{array}{c c} & \begin{array}{cc} 1 & 2 \end{array} \\ \begin{array}{c} 1 \\ 2 \end{array} & \begin{pmatrix} (u-2)\alpha_1 + (u-2)\alpha_2 + 0.5 & (2-u)\alpha_1 + (2-u)\alpha_2 + 0.5 \\ 1 & 0 \end{pmatrix} \end{array}.$$

Thus, the transition probabilities depend linearly on α; in particular, they have the form

$$p(i,j;u,\alpha) = a(i,j;u)\alpha_1 + b(i,j;u)\alpha_2 + c(i,j;u),$$

for $i, j = 1, 2$, where $a = (a(i, j; u))$, $b = (b(i, j; u))$, and $c = (c(i, j; u))$ are the matrices given by

$$a := \begin{pmatrix} u-2 & 2-u \\ 0 & 0 \end{pmatrix}, \quad b := \begin{pmatrix} u-2 & 2-u \\ 0 & 0 \end{pmatrix}, \quad c := \begin{pmatrix} 0.5 & 0.5 \\ 1 & 0 \end{pmatrix}.$$

2.3.4 Notation

For convenience, let $a_m := a(x_m, x_{m+1}; u_m)$, $b_m := b(x_m, x_{m+1}; u_m)$, and $c_m := c(x_m, x_{m+1}; u_m)$ for $m = 0, 1, \ldots$. Then, at each time n, the log-likelihood function, L_n, is given by:

$$\begin{aligned} &\text{Case I.} && L_n(\alpha) = \sum_{m=0}^{n-1} \log(a_m\alpha + b_m), \\ &\text{Case II.} && L_n(\alpha) = \sum_{m=0}^{n-1} \log(a_m\alpha^2 + b_m\alpha + c_m), \\ &\text{Case III.} && L_n(\alpha) = \sum_{m=0}^{n-1} \log(a_m\alpha_1 + b_m\alpha_2 + c_m). \end{aligned}$$

Moreover, we let $a_m^j = a(x_m, j; u_m)$, $b_m^j = b(x_m, j; u_m)$, and $c_m^j = c(x_m, j; u_m)$, for $j \in S$. This notation for a_m, b_m, c_m, a_m^j, b_m^j, c_m^j and $L_n(\cdot)$ is originally due to Sagalovsky [8], and will be used where appropriate throughout this chapter and especially in the corresponding MATLAB code (see Appendix A.2).

2.4 Parameter Estimation, part I

We now examine the result by Mandl [6] that, under a specific identifiability condition, the maximum likelihood estimator for the unknown parameter α is strongly consistent. The identifiability condition in question is the following.

(IC) For each pair $\alpha, \alpha' \in A$, if $\alpha \neq \alpha'$ then exists $i \in S$ such that

$$[p(i, 1; u, \alpha), \ldots, p(i, s; u, \alpha)] \neq [p(i, 1; u, \alpha'), \ldots, p(i, s; u, \alpha')]$$

for every control $u \in U$.

Under this assumption, the result is established.

Theorem 2.4.1 (Mandl). *If (IC) is satisfied, then the sequence of maximum likelihood estimates* $(\hat{\alpha}_n, n = 0, 1, \dots)$ *converges almost surely to the true parameter value* α^0.

We explore this result via numerical simulations of Cases I - III for both finite and compact sets A. It is apparent that (IC) is satisfied in all three cases when the control is taken to be identically zero. Thus, we assume $u \equiv 0$ throughout this section.

2.4.1 Case I

It is first assumed that the unknown parameter α belongs to the known finite set $A := \{0.01, 0.02, 0.03\}$. This case is illustrated in Figure 2.1, and the accompanying MATLAB code is provided in Appendix A.2.8.

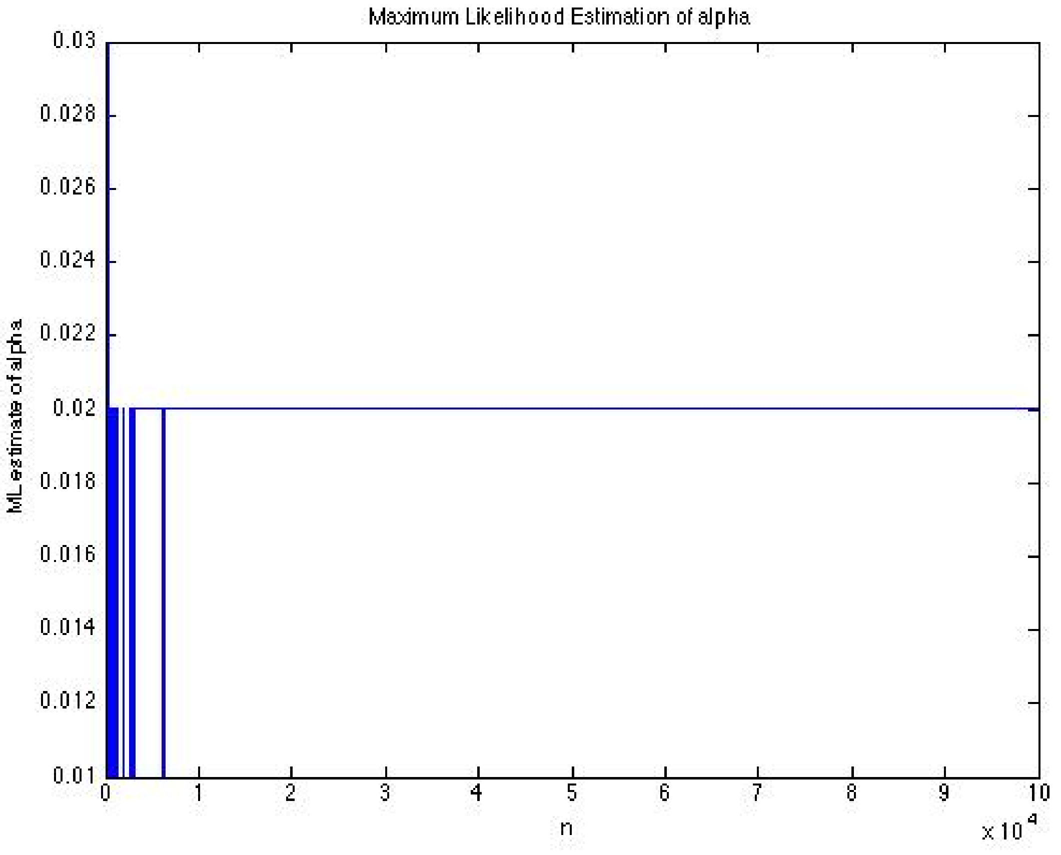

Figure 2.1: a.s. convergence to $\alpha^0 = 0.02$ in Case I for the finite set $A = \{0.01, 0.02, 0.03\}$.

We next suppose that the unknown parameter α belongs to the known compact set $A := [0, 0.1]$. The corresponding simulation behaves analogously to that in the finite case described above. In particular, the maximum likelihood estimates converge almost surely to $\alpha^0 = 0.02$, as seen in Figure 2.2. The accompanying MATLAB code is provided in Appendix A.2.9.

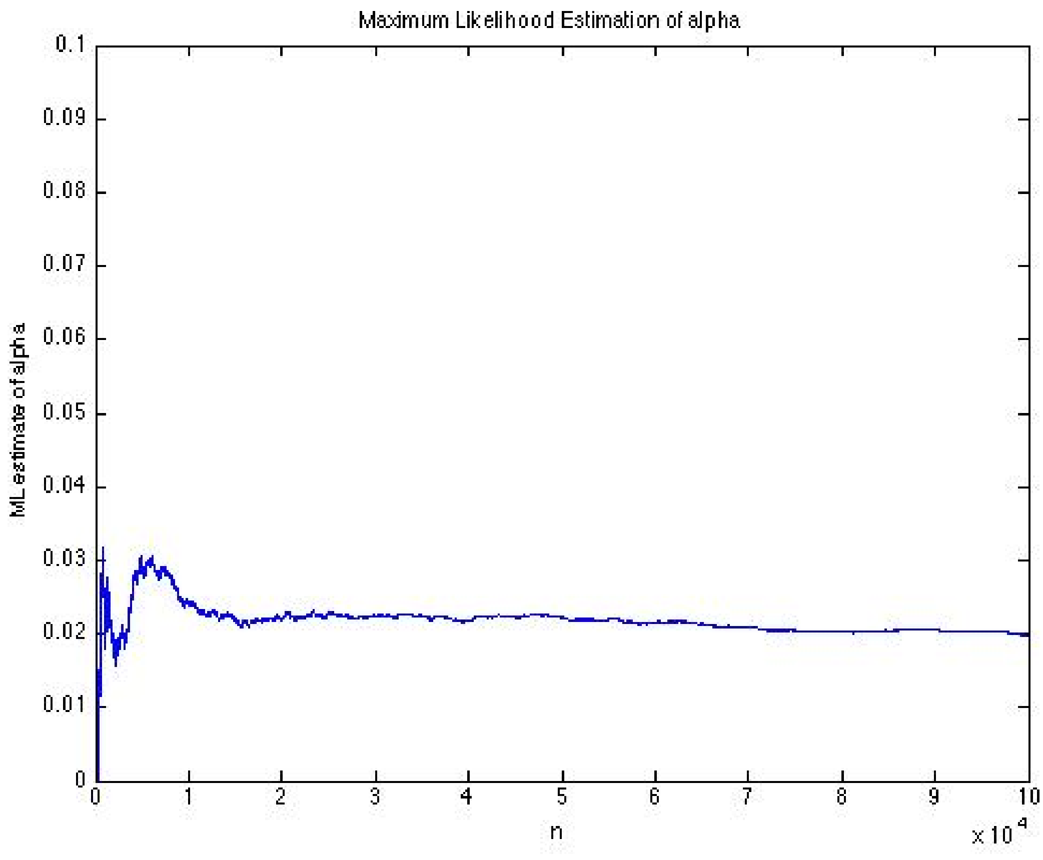

Figure 2.2: a.s. convergence to $\alpha^0 = 0.02$ in Case I for the compact set $A = [0, 0.1]$.

2.4.2 Case II

Again, we first assume that the unknown parameter α belongs to the known finite set $A := \{0.01, 0.02, 0.03\}$, and again, Mandl's result guarantees almost sure convergence to the true value $\alpha^0 = 0.02$. Indeed, this can be seen in Figure 2.3, and the accompanying MATLAB code is provided in Appendix A.2.10.

We next suppose that the unknown parameter α belongs to the known compact set $A := [0, 0.1]$. Again, the simulation behaves analogously to that in the finite case, in that almost sure convergence to $\alpha^0 = 0.02$ is clearly observed. This is illustrated in Figure 2.4, and the accompanying MATLAB code is provided in Appendix A.2.11.

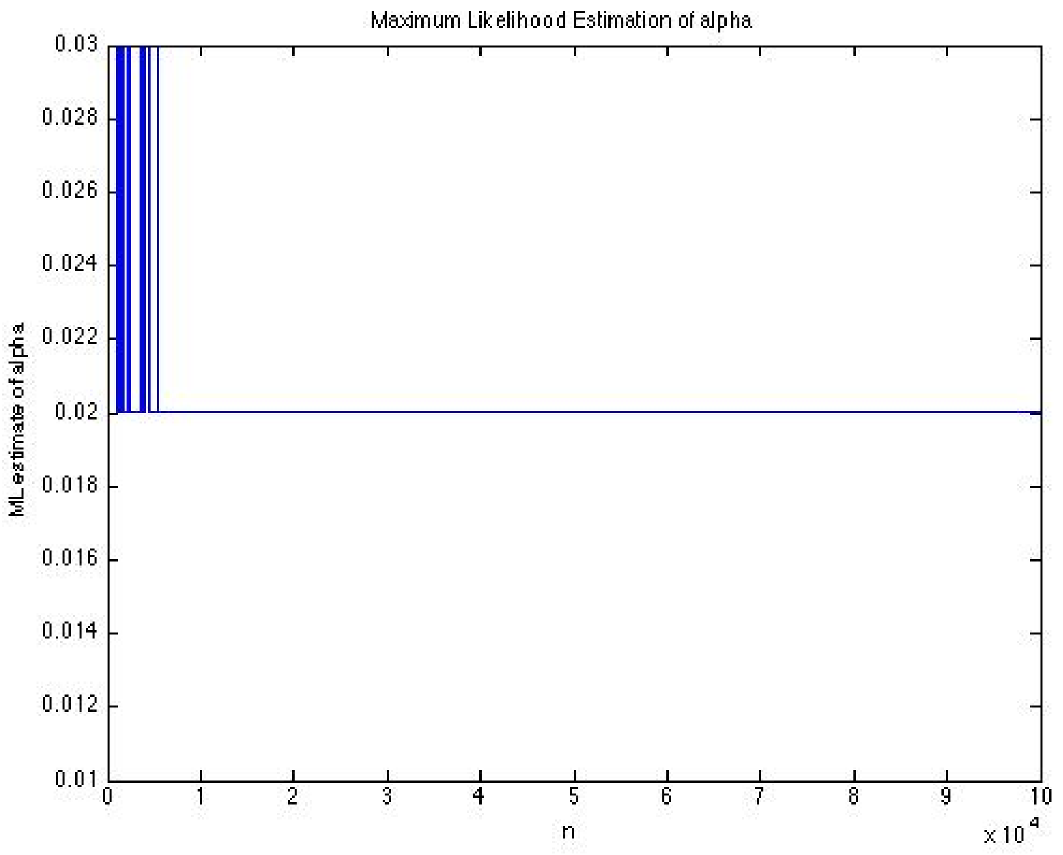

Figure 2.3: a.s. convergence to $\alpha^0 = 0.02$ in Case II for the finite set $A = \{0.01, 0.02, 0.03\}$.

2.4.3 Case III

Lastly, it is assumed that the two-dimensional unknown parameter $\alpha := (\alpha_1, \alpha_2)$ belongs to a known finite set $A := A_1 \times A_2$. In particular, we suppose that

$$\alpha_1 \in A_1 := \{0.01, 0.02, 0.03\},$$
$$\alpha_2 \in A_2 := \{-0.01, -0.02, -0.03\}.$$

Mandl's result guarantees almost sure convergence to the true value $\alpha^0 = (0.01, -0.02)$, as illustrated in Figure 2.5. The accompanying MATLAB code is provided in Appendix A.2.12.

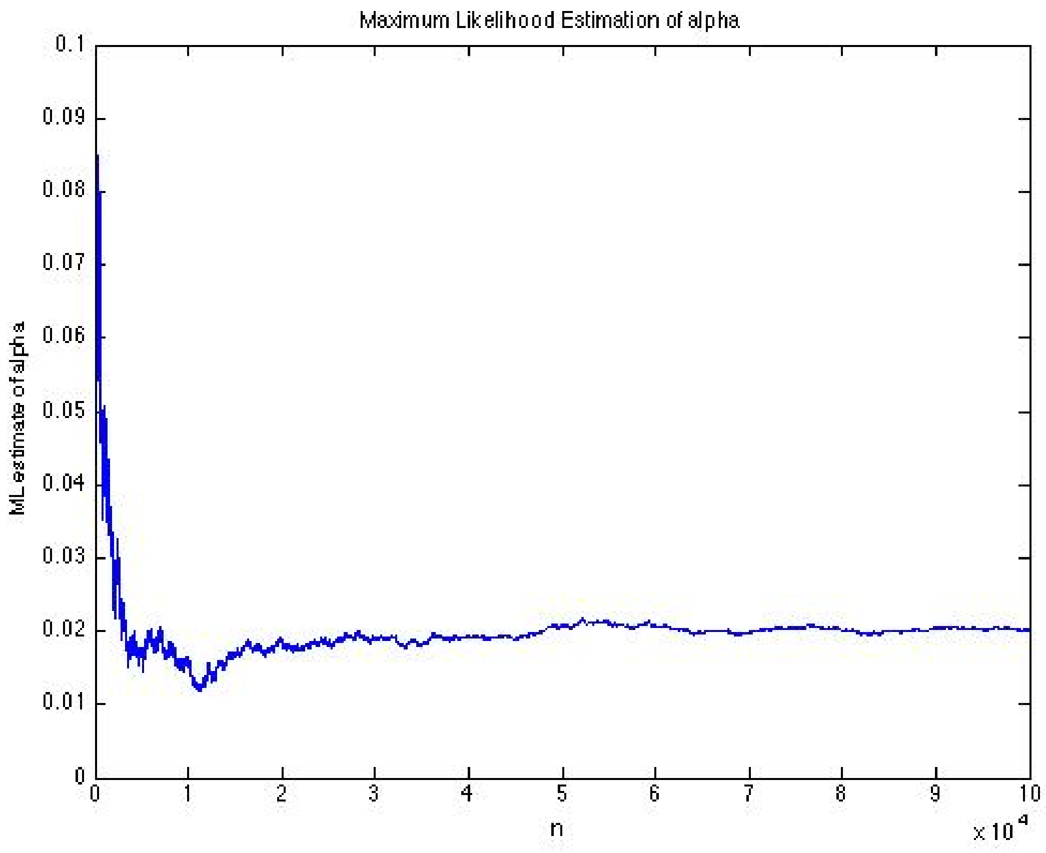

Figure 2.4: a.s. convergence to $\alpha^0 = 0.02$ in Case II for the compact set $A = [0, 0.1]$.

2.5 Parameter Estimation, part II

We now consider the work of Borkar and Varaiya [1], which treats the case in which Mandl's (IC) does not hold. Suppose that the set A is restricted to being finite, and that in place of (IC) we impose two alternative assumptions:

(A3) There exists $\epsilon > 0$ such that for all $i, j \in S$ either $p(i, j; u, \alpha) > \epsilon$ for all $u \in U$ and $\alpha \in A$ or $p(i, j; u, \alpha) = 0$ for all $u \in U$ and $\alpha \in A$.

(A4) The chain is irreducible in the sense that for all $i, j \in S$, there exists a sequence $i = i_0, i_1, \ldots, i_r = j$ such that $p(i_{s-1}, i_s; u, \alpha) > 0$ for all $s = 1, 2, \ldots, r$.

Assuming (A3), (A4), and a feedback control law of the form $u_n = \phi(\alpha_n, x_n)$, the following result is established.

Theorem 2.5.1 (Borkar-Varaiya). *There is a set N of zero measure, a random variable*

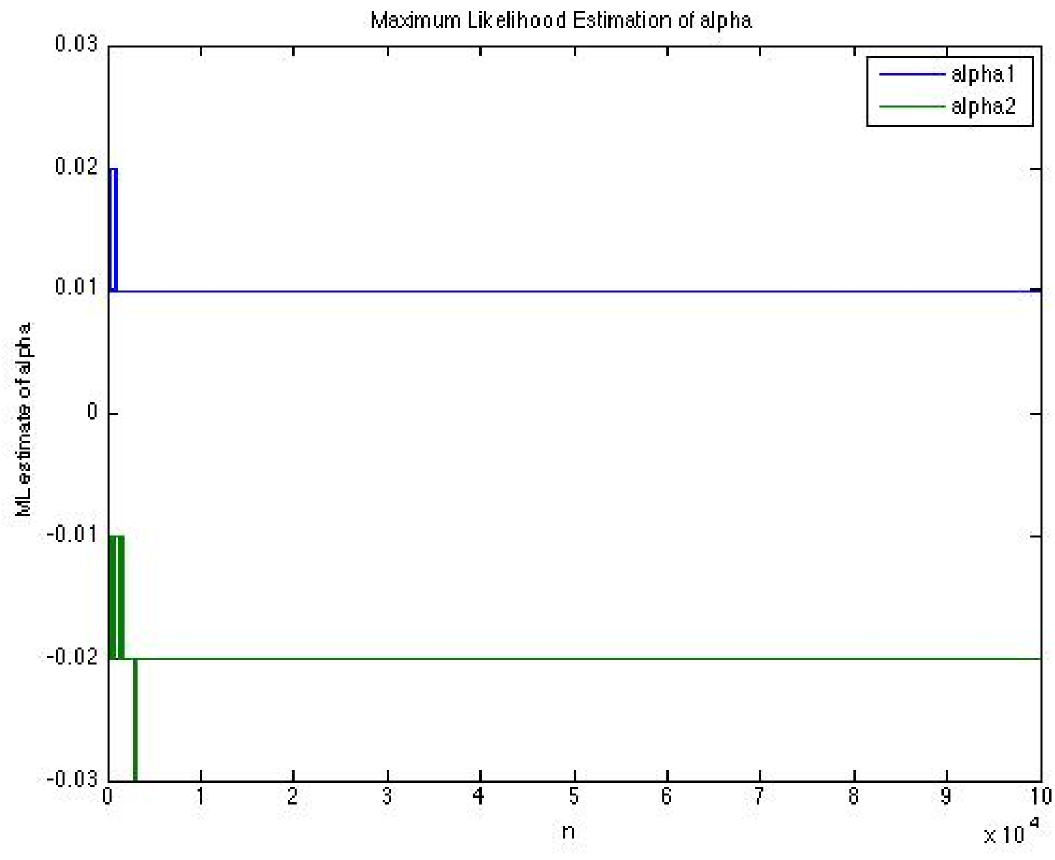

Figure 2.5: a.s. convergence to $\alpha^0 = (0.01, -0.02)$ in Case III for the finite set $A = A_1 \times A_2$.

$\alpha^* \in A$, and a finite random time T such that for $\omega \notin N$, $t \geq T(\omega)$,

$$\alpha_t(\omega) = \alpha^*(\omega), \quad u_t(\omega) = \phi(\alpha^*(\omega), x_t(\omega)),$$

$$p(i,j;\phi(\alpha^*(\omega),i),\alpha^*(\omega)) = p(i,j;\phi(\alpha^*(\omega),i),\alpha^0), \tag{2.1}$$

for all $i, j \in S$.

That is, α^* is indistinguishable from α^0 under the control law induced by α^*. A more desirable result, where (2.1) is replaced by

$$p(i,j;\phi(\alpha^*(\omega),i),\alpha^*(\omega)) = p(i,j;\phi(\alpha^0,i),\alpha^0), \tag{2.2}$$

is not guaranteed. Indeed, the following simulations illustrate both the conclusion of Theorem 2.5.1 and a counter-example to (2.2).

2.5.1 Case I

It is assumed that the unknown parameter α belongs to the known finite set $A := \{0.01, 0.02, 0.03\}$. The following example is due to Borkar and Varaiya [1]. Consider a simple, but non-trivial, feedback control law of the form $u_n = \phi(\hat{\alpha}_n, x_n)$, where $\phi(0.01, i) = \phi(0.03, i) = 2$ and $\phi(0.02, i) = 1$ for $i = 1, 2$. Suppose that the initial state of the chain is $x_0 = 1$ and the initial control is $u_0 = 1$.

It is straightforward to verify that if $x_1 = 1$, then the maximum likelihood estimate is $\hat{\alpha}_1 = 0.01$, while if $x_1 = 2$, then $\hat{\alpha}_1 = 0.03$. In either case, the feedback control law gives $u_1 = 2$. Since $p(i, j; 2, \alpha)$ does not depend on α for any $i, j = 1, 2$, it follows that the maximum likelihood estimate will subsequently remain unchanged. That is, $\hat{\alpha}_n \equiv 0.01$ if $x_1 = 1$ and $\hat{\alpha}_n \equiv 0.03$ if $x_1 = 2$. Figures 2.6 and 2.7 illustrate the two possibilities: convergence of the estimates to either $\alpha^* = 0.01$ or to $\alpha^* = 0.03$, respectively. The accompanying MATLAB code is provided in Appendix A.2.8. This example concretely establishes that convergence to the true value of the unknown parameter is not guaranteed.

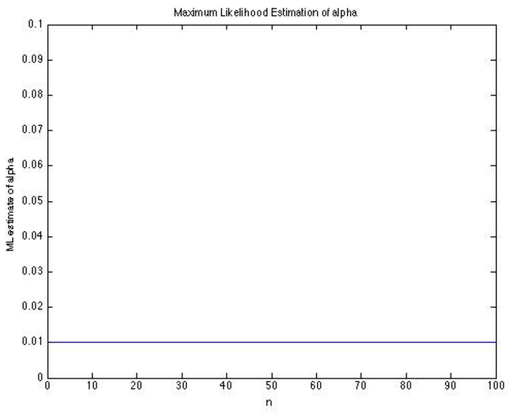

Figure 2.6: convergence to $\alpha^* = 0.01$.

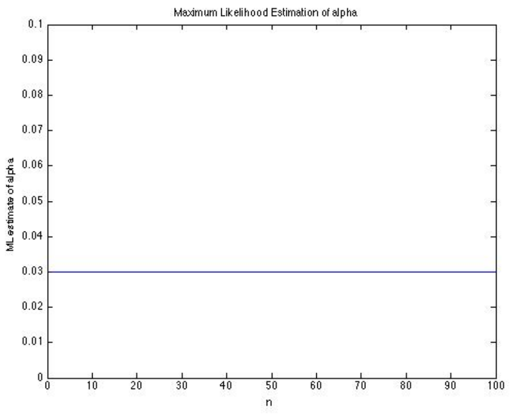

Figure 2.7: convergence to $\alpha^* = 0.03$.

2.5.2 Case II

Again, we assume that the unknown parameter α belongs to the known finite set $A := \{0.01, 0.02, 0.03\}$. The earlier example of Borkar and Varaiya is applicable without modification in this case. Indeed, consider again the feedback control law $u_n = \phi(\hat{\alpha}_n, x_n)$, where $\phi(0.01, i) = \phi(0.03, i) = 2$ and $\phi(0.02, i) = 1$ for $i = 1, 2$. Suppose that the initial state of the chain is $x_0 = 1$ and the initial control is $u_0 = 1$.

It is straightforward to verify that if $x_1 = 1$, then the maximum likelihood estimate is $\hat{\alpha}_1 = 0.01$, while if $x_1 = 2$, then $\hat{\alpha}_1 = 0.03$. In either case, the feedback control law gives $u_1 = 2$. Since $p(i,j;2,\alpha)$ does not depend on α for any $i,j = 1,2$, it follows that the maximum likelihood estimate will subsequently remain unchanged. That is, $\hat{\alpha}_n \equiv 0.01$ if $x_1 = 1$ and $\hat{\alpha}_n \equiv 0.03$ if $x_1 = 2$. Again Figures 2.6 and 2.7 illustrate the two possibilities, and again we see that convergence to the true value of the unknown parameter is not guaranteed. The accompanying MATLAB code is provided in Appendix A.2.10.

2.5.3 Case III

Finally, it is assumed that the two-dimensional unknown parameter $\alpha := (\alpha_1, \alpha_2)$ belongs to a known finite set $A := (A_1, A_2)$. In particular, we suppose that

$$\alpha_1 \in A_1 := \{0.01, 0.02, 0.03\},$$
$$\alpha_2 \in A_2 := \{-0.01, -0.02, -0.03\}.$$

The example of Borkar and Varaiya can be extended to address this two-dimensional parameter case. Indeed, consider again the feedback control law $u_n = \phi(\hat{\alpha}_n, x_n)$, where $\phi((0.01,-0.03),i) = \phi((0.03,-0.01),i) = 2$ and $\phi((\cdot,\cdot),i) = 1$ otherwise for $i = 1,2$. Suppose that the initial state of the chain is $x_0 = 1$ and the initial control is $u_0 = 1$.

It is straightforward to verify that if $x_1 = 1$, then the maximum likelihood estimate is $\hat{\alpha}_1 = (0.01,-0.03)$, while if $x_1 = 2$, then $\hat{\alpha}_1 = (0.03,-0.01)$. In either case, the feedback control law gives $u_1 = 2$. Since $p(i,j;2,\alpha)$ does not depend on α for any $i,j = 1,2$, it follows that the maximum likelihood estimate will subsequently remain unchanged. That is, $\hat{\alpha}_n \equiv (0.01,-0.03)$ if $x_1 = 1$ and $\hat{\alpha}_n \equiv (0.03,-0.01)$ if $x_1 = 2$. Figures 2.8 and 2.9 illustrate the two possibilities: convergence of the estimates to either $\alpha^* = (0.01,-0.03)$ or to $\alpha^* = (0.03,-0.01)$, respectively. The accompanying MATLAB code is provided in Appendix A.2.12. Again, we see that convergence to the true value of the unknown parameter is not guaranteed.

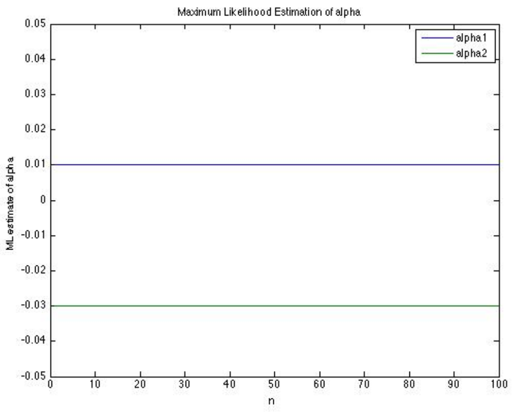

Figure 2.8: convergence to $\alpha^* = (0.01, -0.03)$.

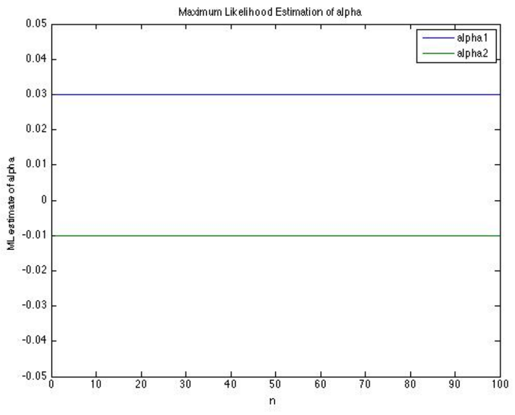

Figure 2.9: convergence to $\alpha^* = (0.03, -0.01)$.

2.6 Survey of Related Results

Kumar [5] considers a more general problem than that of Borkar and Varaiya [1]. Namely, the case in which Mandl's (IC) does not hold and yet the set A of admissible parameter values is permitted to be compact rather than simply finite. As before, the maximum likelihood estimates are defined by $\hat{\alpha}_n := \operatorname{argmax}_{\alpha \in A} L_n(\alpha)$, where L_n is the likelihood function, and the control has the form $u_n = \phi(\hat{\alpha}_n, x_n)$. Kumar shows, via counter-example, that the convergence result of [1] cannot be extended to this case without additional assumptions. In particular, assuming only (A3), (A4), and that $p(\cdot,\cdot;\cdot,\cdot)$ and $\phi(\cdot,\cdot)$ are continuous, the sequence $(\hat{\alpha}_n, n = 1, 2, \dots)$ of maximum likelihood estimates may diverge with probability one.

One example of additional assumptions that guarantee convergence of the maximum likelihood estimates in this generalized case is given by Sagalovsky [8]. Here, the transition probabilities are assumed to depend linearly on a one-dimensional unknown parameter α; that is,

$$p(i, j; u, \alpha) = a(i, j; u)\alpha + b(i, j; u),$$

where $a(\cdot,\cdot;\cdot)$ and $b(\cdot,\cdot;\cdot)$ are known real functions. Note carefully that the linearity of the parameter dependence is necessary in order to ensure the validity of the following theorems, whereas this same linearity was assumed only for convenience throughout Sections 2.3, 2.4, and 2.5.

Taking A to be a compact set, the following results are established.

Theorem 2.6.1 (Sagalovsky)**.** *Under (A3), except for a P-null set of realizations, if the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$, then*

$$\sum_{m=0}^{\infty}\left(\sum_{j=1}^{s}\left(a_m^j\right)^2\right) < \infty. \tag{2.3}$$

Corollary 2.6.2 (Sagalovsky)**.** *Except for a P-null set of realizations, if the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$, then*

$$a_m^j \to 0 \tag{2.4}$$

as $m \to \infty$ for all $j \in S$.

In fact, since A is compact, we see that $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$ if and only if $(\hat{\alpha}_n, n = 1, 2\dots)$ does not converge to α^0. Hence, (2.3) and (2.4) hold almost surely whenever $(\hat{\alpha}_n, n = 1, 2\dots)$ does not converge to α^0.

The following result verifies that, under assumption (A3), convergence of the sequence maximum likelihood estimates is always guaranteed.

Theorem 2.6.3 (Sagalovsky)**.** *If assumption (A3) holds, then the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ converges almost surely to some $\alpha^* \in A$.*

Moreover, with the additional assumption that

(A5) $u_n = \phi(\alpha_n, x_n)$, and $a(i, j; \phi(\alpha, i))$ is continuous in α for all $i, j \in S$,

the main result of Borkar and Varaiya [1] is generalized to the compact case as follows.

Theorem 2.6.4 (Sagalovsky)**.** *Assuming (A3), (A4), (A5), the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2, \dots)$ converges almost surely to a random variable $\alpha^* \in A$ that satisfies* (2.1) *for all $i, j \in S$.*

The results of Sagalovsky are extended by Pasik-Duncan [7]. Here, the transition probabilities are assumed to depend linearly on a two-dimensional unknown parameter

$\alpha = (\alpha_1, \alpha_2)$; that is,

$$p(i,j;u,\alpha) = a(i,j;u)\alpha_1 + b(i,j;u)\alpha_2 + c(i,j;u),$$

where $a(\cdot,\cdot;\cdot)$, $b(\cdot,\cdot;\cdot)$, and $c(\cdot,\cdot;\cdot)$ are known real functions. Again, the linearity of the dependance is vital. Taking $A = A_1 \times A_2$ to be a two-dimensional compact set, and imposing the assumption that

(A6) for all m, either $a_m, b_m \geq 0$ or $a_m, b_m \leq 0$,

the following results are established.

Theorem 2.6.5 (Pasik-Duncan). *Under (A3), (A6), except for a P-null set of realizations, if the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$, then*

$$\sum_{m=0}^{\infty}\left(\sum_{j=1}^{s}\left(a_m^j\right)^2 + a_m^j b_m^j\right) < +\infty, \tag{2.5}$$

$$\sum_{m=0}^{\infty}\left(\sum_{j=1}^{s}\left(b_m^j\right)^2 + a_m^j b_m^j\right) < +\infty. \tag{2.6}$$

Corollary 2.6.6 (Pasik-Duncan). *Except for a P-null set of realizations, if the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$, then*

$$a_m^j \to 0 \tag{2.7}$$

$$b_m^j \to 0 \tag{2.8}$$

as $m \to \infty$ for all $j \in S$.

Again, we remark that, since A is compact, $(\hat{\alpha}_n, n = 1, 2\dots)$ has an accumulation point $\alpha^* \neq \alpha^0$ if and only if $(\hat{\alpha}_n, n = 1, 2\dots)$ does not converge to α^0. Hence, (2.5), (2.6), (2.7), and (2.8) hold almost surely whenever $(\hat{\alpha}_n, n = 1, 2\dots)$ does not converge to α^0.

Theorem 2.6.7 (Pasik-Duncan). *If assumptions (A3), (A6) hold, then the sequence of maximum likelihood estimates $(\hat{\alpha}_n, n = 1, 2\dots)$ converges almost surely to some $\alpha^* \in A$.*

Moreover, with the additional assumption (analogous to (A5)) that

(A7) $u_n = \phi(\alpha_n, x_n)$, and $a(i,j;\phi(\alpha,i))$, $b(i,j;\phi(\alpha,i))$ are continuous in α for all $i,j \in S$,

the result of Sagalovsky is generalized to the two-dimensional parameter case as follows.

Theorem 2.6.8 (Pasik-Duncan). *Assuming (A3), (A4), (A6), (A7), the sequence of maximum likelihood estimates* $(\hat{\alpha}_n, n = 1, 2, \dots)$ *converges almost surely to a random variable* $\alpha^* \in A$ *that satisfies* (2.1) *for all* $i, j \in S$.

2.7 Future Investigations

It remains to extend the numerical simulations of Sections 2.4 and 2.5 to illustrate the results of Sagalovsky and Pasik-Duncan presented in Theorems 2.6.4 and 2.6.8, respectively. Specifically, the MATLAB code for should be modified so as to incorporate a non-trivial feedback control action in all cases (rather than simply when A is a finite set). This would allow for the illustration of convergence behavior in the case where Mandl's (IC) does not hold but where A is assumed to be compact and non-finite.

Moreover, we would like to simulate the results of Theorem 2.6.1 (or Corollary 2.6.2) and Theorem 2.6.5 (or Corollary 2.6.6), which provide an easily verifiable condition for the strong consistency of the maximum likelihood estimator in the cases of linear dependence on a one- and two-dimensional unknown parameter, respectively. We could then explore a similar condition in the related case of quadratic (more generally, polynomial) dependence on a one-dimensional unknown parameter when Mandl's (IC) does not hold; this would be of particular interest, since for this generalized case a convergence result analogous to Theorem 2.6.4 has been conjectured by Pasik-Duncan but not proved.

2.8 Concluding Remarks

In considering a controlled Markov chain whose transition probabilities are assumed to depend on both an unknown parameter, α, and a control action, u, we provided numerical simulations for maximum likelihood estimation of α in a variety of cases. Specifically, we first considered Mandl's [6] result regarding the strong consistency of the maximum likelihood estimator under the assumption of an identifiability condition. Next, we addressed a similar result by Borkar and Varaiya [1]; in particular, that under alternative assumptions the sequence of maximum likelihood estimates converges and retains a desirable (although not ideal) property relating to the Markov chain's transition probabilities. Finally, we surveyed related results by Kumar [5], Sagalovsky [8], and Pasik-Duncan [7].

References

[1] V. Borkar and P. Varaiya. Adaptive Control of Markov Chains, I: Finite Parameter Set. *IEEE Trans. on Autom. Control*, 24(6):953–957, Dec 1979.

[2] T. E. Duncan, P. Mandl, and B. Pasik-Duncan. Numerical Differentiation and Parameter Estimation in Higher-Order Linear Stochastic Systems. *IEEE Trans. on Autom. Control*, 41(4):522–532, Apr 1996.

[3] T. E. Duncan, P. Mandl, and B. Pasik-Duncan. On Statistical Sampling for System Testing. *IEEE Trans. on Autom. Control*, 39(1):118–122, Jan 1994.

[4] D. J. Higham. An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations. *SIAM Review*, 43(3):525–546, Aug 2001.

[5] P. R. Kumar. Adaptive Control with a Compact Parameter Set. *SIAM J. of Control and Optim.*, 20(1):9–13, Jan 1982.

[6] P. Mandl. Estimation and Control in Markov Chains. *Adv. in Applied Prob.*, 6(1):40–60, Mar 1974.

[7] B. Pasik-Duncan. *On Adaptive Control.* SGPiS-Publishers, Warsaw, 1986.

[8] B. Sagalovsky. Adaptive Control and Parameter Estimation in Markov Chains: A Linear Case. *IEEE Trans. on Autom. Control*, 27(2):414–419, Apr 1982.

Appendix A

MATLAB code

Here, we provide the MATLAB code used to produce the numerical simulations presented throughout Chapters 1 and 2.

A.1 Code for Chapter 1

- Appendix A.1.1 – function for simulating the solution of a stochastic differential equation via the recursive Euler-Maruyama method.
- Appendices A.1.2, A.1.3 – functions for computing numerical derivative approximations via the forward difference method and the alternate (linear combination of forward differences) method, respectively.
- Appendices A.1.4, A.1.5, A.1.6 – functions for computing the values $C(d)$, $B(d)$, and $A(d)$, respectively, where $d \geq 0$ is the order of the stochastic system.
- Appendices A.1.7, A.1.8 – code for performing quadratic variation estimation of the local variance matrix of a Brownian motion via the forward difference approach and the alternate numerical differentiation approach, respectively.
- Appendices A.1.9, A.1.10 – code for performing least squares estimation of an unknown parameter in the higher-order stochastic system via the forward difference approach and the alternate numerical differentiation approach, respectively.

A.1.1 E-M recursion for SDE solution approximation

```
function X = em(F,d,mu,n,delta,true_h)

% ------------------------------------------------------------------ %
% Function returns the approximate solution of a stochastic          %
% differential equation (SDE) via the Euler-Maruyama recursion       %
% method.                                                            %
%                                                                    %
% Written by Cody E. Clifton, 9-25-2012.                             %
% ------------------------------------------------------------------ %

M = mu*(n-1)+1; % fix the number of desired SDE solution values
rho = delta/mu; % fix the step-size for Euler-Maruyama recursion

Y = zeros(d,M); % initialize the process Y(t)=[X(t);...;X^{(d-1)}(t)]
Y(:,1) = [0 1]'; % fix the initial value of the process
for i=2:M % recursively assign values to Y(t)
    dW = [0; sqrt(rho*true_h)*randn];
    Y(:,i) = Y(:,i-1) + (F*Y(:,i-1))*rho + dW;
end % for

X = zeros(n,1); % initalize the matrix process X(t)
for i=1:n % assign values to X(t)
    X(i,1) = Y(1,mu*(i-1)+1);
end % for

end % function
```

A.1.2 Forward difference method for derivative approximation

```
function X_diff = fwd_diff(X,n,delta)

% ------------------------------------------------------------------ %
% Function returns the forward difference numerical derivative       %
% approximations of the stochastic process X, with step-size delta.  %
%                                                                     %
% Written by Cody E. Clifton, 9-25-2012.                              %
% ------------------------------------------------------------------- %

X_diff = zeros(n,1); % initialize the matrix of derivative approx's
for i=1:n % compute the forward difference derivative approximations
    X_diff(i,1) = (X(i+1,1) - X(i,1))/delta;
end % for

end % function
```

A.1.3 Alternate method for derivative approximation

The following code requires the MATLAB function provided in Appendix A.1.2.

```
function X_tilde = num_deriv(X,n,delta,a)

% ------------------------------------------------------------------- %
% Function returns a numerical derivative approximation based on      %
% three stochastic process values, a step size (delta), and a         %
% constant (a).                                                       %
%                                                                     %
% Requires the function: fwd_diff.m                                   %
%                                                                     %
% Written by Cody E. Clifton, 10-22-2012.                             %
% ------------------------------------------------------------------- %

X_tilde = zeros(n,1); % initialize the matrix of derivative approx's
for i=1:n % compute the derivative approximations
    X_tilde(i,1) = sqrt(2)*((X(i+1,1)-X(i,1))/delta + ...
                   a*((X(i+2,1)-X(i+1,1))/delta - ...
                      (X(i+1,1)-X(i,1))/delta));
end % for

end % function
```

A.1.4 Computation of the value $C(d)$

```
function C = C_value(d)

% ------------------------------------------------------------------ %
% Function returns the value C(d), where d is the dimension of the   %
% system.                                                            %
%                                                                    %
% Written by Cody E. Clifton, 6-25-2012.                             %
% ------------------------------------------------------------------ %

C = 0;
for j=1:d
    C = C + ((-1)^j)*(j^(2*d-1))*(nchoosek(2*d,d-j));
end % for
C = (((-1)^d)/(factorial(2*d-1)))*C;

end % function
```

A.1.5 Computation of the value $B(d)$

```
function B = B_value(d)

% ------------------------------------------------------------------ %
% Function returns the value B(d), where d is the dimension of the   %
% system.                                                            %
%                                                                    %
% Written by Cody E. Clifton, 6-25-2012.                             %
% ------------------------------------------------------------------ %

```

```
B = 0;
for j=1:d+1
    B = B + ((-1)^j)*(j^(2*d-1))*(nchoosek(2*d+2,d+1-j));
end % for
B = (((-1)^d)/(factorial(2*d-1)))*B;

end % function
```

A.1.6 Computation of the value $A(d)$

```
function A = A_value(B,C)

% ------------------------------------------------------------------ %
% Function returns the value A(d)>0, using the values B=B(d) and     %
% C=C(d), where d is the dimension of the system.                    %
%                                                                    %
% Written by Cody E. Clifton, 6-25-2012.                             %
% ------------------------------------------------------------------ %

A = (-1 + sqrt(1 - 4*((C-1)/B)))/2;

end % function
```

A.1.7 Estimation of h using the forward difference approach

The following code requires the MATLAB functions provided in Appendices A.1.1, A.1.2, and A.1.4.

```
% ------------------------------------------------------------------ %
% This code performs quadratic variation estimation on the local     %
% variance matrix h of the Brownian motion, and displays a three-    %
% dimensional plot of the convergence of this estimator as N ->      %
% infinity and delta -> 0.                                           %
%                                                                        %
% Required functions: em.m, fwd_diff.m, C_value.m                       %
%                                                                        %
% Written by Cody E. Clifton, 11-08-2012.                                %
% -------------------------------------------------------------------- %

clear % clear all variables

true_h = 1; % record the true value of the local variance matrix h
true_alpha = 1; % record the true value of the parameter

d = 2; % define the dimension of the system
C = C_value(d); % generate the value of the constant C(d)

% fix the factor by which the discretization of the estimation
% procedure varies from the discretization of the SDE solution
mu = 100;

N_size = 4; % fix the number of iterations of N to be tested
delta_size = 4; % fix the number of iterations of delta to be tested

h = zeros(N_size,delta_size); % initialize the estimate array
N = zeros(N_size,1); % initialize the array of N-values
delta = zeros(delta_size,1); % initialize the array of delta-values

% define f_ij (with f1=f2=-1) such that f_i = f_i0 + alpha*f_i1
f10=-2; f11=1; f20=1; f21=-2;

F = [0 1; (f10 + true_alpha*f11) (f20 + true_alpha*f21)];

for i=1:N_size
    for j=1:delta_size
        N(i,1) = 10^i; % fix the number of estimation increments
        delta(j,1) = 10^(-j); % fix the SDE solution step size

        % simulate the SDE solution via Euler-Maruyama
        X = em(F,d,mu,N(i,1)+d,delta(j,1),true_h);

        % compute the forward difference derivative approximations.
        X_diff = fwd_diff(X,N(i,1)+d-1,delta(j,1));

        for k=1:N(i,1)
            h(i,j) = h(i,j) + (X_diff(k+1) - X_diff(k))^2;
        end % for
        h(i,j) = (1/(N(i,1)*delta(j,1)))*h(i,j);
    end % for
end % for

disp(h) % display the array of estimated h-values

surf(delta,N,h) % display a surface plot illustrating convergence
xlabel('delta (-> zero)','Fontsize',13,'Rotation',0)
ylabel('N (-> infinity)','Fontsize',13,'Rotation',0)
zlabel('h := h(N,delta)','Fontsize',13,'Rotation',90)
```

A.1.8 Estimation of h using the alternate approach

The following code requires the MATLAB functions provided in Appendices A.1.1, A.1.3, A.1.4, A.1.5, and A.1.6.

```
% ------------------------------------------------------------------ %
% This code performs quadratic variation estimation on the local     %
% variance matrix h of the Brownian motion with the alternate        %
% numerical differentiation scheme given by num_deriv.m, and         %
% displays a three-dimensional plot of the convergence of this       %
% estimator as N -> infinity and delta -> 0.                         %
%                                                                    %
% Required functions: em.m, num_deriv.m,                             %
%                     A_value.m, B_value.m, C_value.m                %
%                                                                         %
% Written by Cody E. Clifton, 11-08-2012.                                 %
% ---------------------------------------------------------------------- %

clear

true_h = 1; % record the true value of the local variance matrix h
true_alpha = 1; % record the true value of the parameter

d = 2; % define the dimension of the system
B = B_value(d); % generate the value of the constant B(d)
C = C_value(d); % generate the value of the constant C(d)
A = A_value(B,C); % generate the value of the constant A(d)

% fix the factor by which the discretization of the estimation
% procedure varies from the discretization of the SDE solution
mu = 100;

N_size = 4; % fix the number of iterations of N to be tested
delta_size = 4; % fix the number of iterations of delta to be tested

h = zeros(N_size,delta_size); % initialize the estimate array
N = zeros(N_size,1); % initialize the array of N-values
delta = zeros(delta_size,1); % initialize the array of delta-values

% define f_ij (with f1=f2=-1) such that f_i = f_i0 + alpha*f_i1
f10=-2; f11=1; f20=1; f21=-2;

F = [0 1; (f10 + true_alpha*f11) (f20 + true_alpha*f21)];

for i=1:N_size
    for j=1:delta_size
        N(i,1) = 10^i; % fix the number of estimation increments
        delta(j,1) = 10^(-j); % fix the SDE solution step size

        % simulate the SDE solution via Euler-Maruyama
        X = em(F,d,mu,N(i,1)+d+1,delta(j,1),true_h);

        % compute the numerical derivative approximations
        X_tilde = num_deriv(X,N(i,1)+d-1,delta(j,1),A);

        for k=1:N(i,1)
            h(i,j) = h(i,j) + (X_tilde(k+1) - X_tilde(k))^2;
        end % for
        h(i,j) = (1/(N(i,1)*delta(j,1)))*h(i,j);
    end % for
end % for

disp(h) % display the array of estimated h-values

surf(delta,N,h) % display a surface plot illustrating convergence
xlabel('delta (-> zero)','Fontsize',13,'Rotation',0)
ylabel('N (-> infinity)','Fontsize',13,'Rotation',0)
zlabel('h := h(N,delta)','Fontsize',13,'Rotation',90)
```

A.1.9 Estimation of α using the forward difference approach

The following code requires the MATLAB functions provided in Appendices A.1.1, A.1.2, and A.1.4.

```
% ------------------------------------------------------------------ %
% This code performs least squares estimation on the unknown         %
% parameter in the stochastic differential equation, and displays a  %
% three-dimensional plot of the convergence of this estimator as     %
% N -> infinity and delta -> 0.                                      %
%                                                                    %
% Required functions: em.m, fwd_diff.m, and C_value.m                %
%                                                                    %
% Written by Cody E. Clifton, 11-08-2012.                            %
% ------------------------------------------------------------------ %

clear

true_h = 1; % record the true value of the local variance matrix h
true_alpha = 1; % record the true value of the parameter

d = 2; % define the dimension of the system
B = B_value(d); % generate the value of the constant B(d)
C = C_value(d); % generate the value of the constant C(d)
A = A_value(B,C); % generate the value of the constant A(d)

% fix the factor by which the discretization of the estimation
% procedure varies from the discretization of the SDE solution
mu = 100;

N_size = 4; % fix the number of iterations of N to be tested
delta_size = 4; % fix the number of iterations of delta to be tested

alpha = zeros(N_size,delta_size); % initialize the estimate array
N = zeros(N_size,1); % initialize the array of N-values
delta = zeros(delta_size,1); % initialize the array of delta-values

% define f_ij (with f1=f2=-1) such that f_i = f_i0 + alpha*f_i1
f10=-2; f11=1; f20=1; f21=-2;

F = [0 1; (f10 + true_alpha*f11) (f20 + true_alpha*f21)];

for i=1:N_size
    for j=1:delta_size
        N(i,1) = 10^i; % fix the number of estimation increments
        delta(j,1) = 10^(-j); % fix the SDE solution step size

        % simulate the SDE solution via Euler-Maruyama
        X = em(F,d,mu,N(i,1)+d,delta(j,1),true_h);

        % compute the forward difference derivative approximations
        X_diff = fwd_diff(X,N(i,1)+d-1,delta(j,1));

        S2 = 0; % initialize the left-hand side of LSE equation
        for k=1:N(i,1)
            S2 = S2 + (f11*X(k,1) + f21*X_diff(k,1))^2;
        end % for
        S2 = (1/N(i,1))*S2;

        S1 = 0; % initialize the right-hand side of LSE equation
        for k=1:N(i,1)
            S1 = S1 + (f11*X(k,1) + f21*X_diff(k,1))* ...
                      ((X_diff(k+1,1) - X_diff(k,1)) - ...
                      delta(j,1)*(f10*X(k,1) + f20*X_diff(k,1)));
        end % for
        S1 = (1/(N(i,1)*delta(j,1)))*S1;

        COR = 0; % initialize the bias correction term
        for k=1:N(i,1)
            COR = COR + f21*(X_diff(k+1,1) - X_diff(k,1))^2;
        end % for
        COR = (1/(N(i,1)*delta(j,1)))*((C-1)/(2*C))*COR;
        S1 = S1 - COR;

        alpha(i,j) = S1/S2;
    end % for
end % for

disp(alpha) % display the array of estimated parameter values

% display a surface plot illustrating parameter convergence
surf(delta,N,alpha)
xlabel('delta (-> zero)','Fontsize',13,'Rotation',0)
ylabel('N (-> infinity)','Fontsize',13,'Rotation',0)
```

```
zlabel('alpha := alpha(N,delta)','Fontsize',13,'Rotation',90)
```

A.1.10 Estimation of α using the alternate approach

The following code requires the MATLAB functions provided in Appendices A.1.1, A.1.3, A.1.4, A.1.5, and A.1.6.

```
% -------------------------------------------------------------------- %
% This code performs least squares estimation on the unknown           %
% parameter in the stochastic differential equation, and displays a    %
% three-dimensional plot of the convergence of this estimator as       %
% N -> infinity and delta -> 0.                                        %
%                                                                      %
% Required functions: em.m, num_deriv.m,                               %
%                      A_value.m, B_value.m, C_value.m                 %
%                                                                      %
% Written by Cody E. Clifton, 11-08-2012.                              %
% -------------------------------------------------------------------- %

clear

true_h = 1; % record the true value of the local variance matrix h
true_alpha = 1; % record the true value of the parameter

d = 2; % define the dimension of the system
B = B_value(d); % generate the value of the constant B(d)
C = C_value(d); % generate the value of the constant C(d)
A = A_value(B,C); % generate the value of the constant A(d)

% fix the factor by which the discretization of the estimation
% procedure varies from the discretization of the SDE solution
mu = 100;

N_size = 4; % fix the number of iterations of N to be tested
```

```
delta_size = 4; % fix the number of iterations of delta to be tested

alpha = zeros(N_size,delta_size); % initialize the estimate array
N = zeros(N_size,1); % initialize the array of N-values
delta = zeros(delta_size,1); % initialize the array of delta-values

% define f_ij (with f1=f2=-1) such that f_i = f_i0 + alpha*f_i1
f10=-2; f11=1; f20=1; f21=-2;

F = [0 1; (f10 + true_alpha*f11) (f20 + true_alpha*f21)];

for i=1:N_size
    for j=1:delta_size
        N(i,1) = 10^i; % fix the number of estimation increments
        delta(j,1) = 10^(-j); % fix the SDE solution step size

        % simulate the SDE solution via Euler-Maruyama
        X = em(F,d,mu,N(i,1)+d+1,delta(j,1),true_h);

        % compute the numerical derivative approximations
        X_tilde = num_deriv(X,N(i,1)+d-1,delta(j,1),A);

        S2 = 0; % initialize the left-hand side of LSE equation
        for k=1:N(i,1)
            S2 = S2 + (f11*X(k,1) + f21*X_tilde(k,1))^2;
        end % for
        S2 = (1/N(i,1))*S2;

        S1 = 0; % initialize the right-hand side of LSE equation
        for k=1:N(i,1)
            S1 = S1 + (f11*X(k,1) + f21*X_tilde(k,1))* ...
                      ((X_tilde(k+1,1) - X_tilde(k,1)) - ...
                      delta(j,1)*(f10*X(k,1) + f20*X_tilde(k,1)));
        end % for
        S1 = (1/(N(i,1)*delta(j,1)))*S1;

        alpha(i,j) = S1/S2;
    end % for
end % for

disp(alpha) % display the array of estimated parameter values

% display a surface plot illustrating parameter convergence
surf(delta,N,alpha)
xlabel('delta (-> zero)','Fontsize',13,'Rotation',0)
ylabel('N (-> infinity)','Fontsize',13,'Rotation',0)
zlabel('alpha := alpha(N,delta)','Fontsize',13,'Rotation',90)
```

A.2 Code for Chapter 2

- Appendix A.2.1, A.2.2 – functions for simulating an observed trajectory of a Markov chain: the first is optimized to require minimal processing power, while the second is optimized to require minimal memory.

- Appendices A.2.3, A.2.4, A.2.5 – functions for computing the control-dependent matrices $a = (a(i,j;u))$, $b = (b(i,j;u))$, and $c = (c(i,j;u))$, respectively.

- Appendices A.2.6, A.2.7 – functions for computing a feedback control, $u_n = \phi(\hat{\alpha}_n, x_n)$, when the unknown parameter is one-dimensional and two-dimensional, respectively.

- Appendices A.2.8, A.2.9 – code for simulating maximum likelihood estimation in the case of linear dependence on a one-dimensional parameter belonging to a finite set and to a compact set, respectively.

- Appendices A.2.10, A.2.11 – code for simulating maximum likelihood estimation in the case of quadratic dependence on a one-dimensional parameter belonging to a finite set and to a compact set, respectively.

- Appendix A.2.12 – code for simulating maximum likelihood estimation in the case of linear dependence on a two-dimensional parameter belonging to a finite set.

A.2.1 Simulation of successive states of a Markov chain, v.1

```
function X = MarkovSim1(N,P,X0)

% ------------------------------------------------------------------ %
% This function is optimized to require minimal processing power.    %
% ------------------------------------------------------------------ %
%                                                                    %
% Let 'P' denote the transition probability matrix of a Markov       %
% chain.  This function produces a vector containing 'N' simulated   %
% observations of the state of the chain.                            %
%                                                                    %
% Inputs:    P  = transition probability matrix of a Markov chain.   %
%            N  = desired number of simulated observations.          %
%            X0 = initial state                                      %
%                                                                    %
% Outputs:   X = vector containing N simulated observations of the   %
%                state of a Markov chain with transition probability %
%                matrix P.                                           %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-22.                            %
% ------------------------------------------------------------------ %

S = size(P,1); % let 's' be the size of the Markov chain's state space

X = zeros(N,1); % initialize the vector of simulated observations
X(1) = X0; % fix the initial state of the chain to be 'X0'

% The distinguishing feature of this function, as compared with
% MarkovSim2, is the following preallocation of cumulative row-sum
% probabilities from the matrix P, which causes the function to
% require less processing power but more memory.

Q = cumsum(P,2);

for i=1:N-1
    r = rand; % produce a random value from the Uniform[0,1] distr.
    for j=1:S
        if (r < Q(X(i),j))
            X(i+1) = j; % fix the next simulated state of the chain
            break; % exit the for-loop containing the if statement
        end % if
    end % for
end % for

end % function
```

A.2.2 Simulation of successive states of a Markov chain, v.2

```
function X = MarkovSim2(N,P,X0)

% ------------------------------------------------------------------ %
% This function is optimized to require minimal memory.              %
% ------------------------------------------------------------------ %
%                                                                    %
% Let 'P' denote the transition probability matrix of a Markov       %
% chain. This function produces a vector containing 'N' simulated    %
% observations of the state of the chain.                            %
%                                                                    %
% Inputs:    P = transition probability matrix of a Markov chain.    %
%            N = desired number of simulated observations.           %
%            X0 = initial state                                      %
%                                                                    %
% Outputs:   X = vector containing N simulated observations of the   %
%                state of a Markov chain with transition probability %
%                matrix P.                                          %
%                                                                   %
% Written by Cody E. Clifton, 2013-03-22.                           %
% ------------------------------------------------------------------ %

S = size(P,1); % let 's' be the size of the Markov chain's state space

X = zeros(N,1); % initialize the vector of simulated observations
X(1) = X0; % fix the initial state of the chain to be 'X0'

% The distinguishing feature of this function, as compared with
% MarkovSim1, is that the cumulative row-sum probabilities from the
% matrix P are not preallocated, which causes the function to require
% less memory but more processing power.

for i=1:N-1
    r = rand; % produce a random value from the Uniform[0,1] distr.
    for j=1:S
        q = 0;
        for k=1:j
            q = q + P(X(i),j); % sum the X(i)th row of P from 1 to j
        end % for
        if (r < q)
            X(i+1) = j; % fix the next simulated state of the chain
            break; % exit the for-loop containing the if statement
        end % if
    end % for
end % for

end % function
```

A.2.3 Computation of the control-dependent matrix $a = (a(i,j;u))$

```
function a = a_values(u)
```

```

% ------------------------------------------------------------------ %
% This function returns the control-dependent matrix a = (a(i,j;u)). %
%                                                                    %
% Inputs:   u = control.                                             %
%                                                                    %
% Outputs:  a = matrix of values (a(i,j;u)).                         %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-22.                            %
% ------------------------------------------------------------------ %

a = zeros(2); % initialize a

% assign values to the known function a
a(1,1) = u - 2; a(1,2) = 2 - u;
a(2,1) = 0;     a(2,2) = 0;

end % function
```

A.2.4 Computation of the control-dependent matrix $b = (b(i, j; u))$

```
function b = b_values(u)

% ------------------------------------------------------------------ %
% This function returns the control-dependent matrix b = (b(i,j;u)). %
%                                                                    %
% Inputs:   u = control.                                             %
%                                                                    %
% Outputs:  b = matrix of values (b(i,j;u)).                         %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-22.                            %
% ------------------------------------------------------------------ %

b = zeros(2); % initialize b
```

```

% assign values to the known function b
b(1,1) = 0.5; b(1,2) = 0.5;
b(2,1) = 1;   b(2,2) = 0;

end % function
```

A.2.5 Computation of the control-dependent matrix $c = (c(i,j;u))$

```
function c = c_values(u)

% ------------------------------------------------------------------ %
% This function returns the control-dependent matrix c = (c(i,j;u)). %
%                                                                    %
% Inputs:   u = control.                                             %
%                                                                    %
% Outputs:  c = matrix of values (c(i,j;u)).                         %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-22.                            %
% ------------------------------------------------------------------ %

c = zeros(2); % initialize c

% assign values to the known function c
c(1,1) = 0.5; c(1,2) = 0.5;
c(2,1) = 1;   c(2,2) = 0;

end % function
```

A.2.6 Computation of feedback control for a 1D parameter

```
function u = control1D(alpha_hat)

```

```
% ------------------------------------------------------------------ %
% This function returns a feedback control that depends on the       %
% maximum likelihood estimate (MLE) of a one-dimensional unknown     %
% parameter in the transition probabilities of a Markov chain.       %
%                                                                    %
% Inputs:   alpha_hat = one-dimensional MLE                          %
%                                                                    %
% Outputs:  u = feedback control, dependent on the MLE alpha_hat.    %
%                                                                    %
% Written by Cody E. Clifton, 2013-04-03.                            %
% ------------------------------------------------------------------ %

if (alpha_hat==0.02)
        u = 1;
else
        u = -1;
end % if/else

end % function
```

A.2.7 Computation of feedback control for a 2D parameter

```
function u = control2D(alpha1_hat,alpha2_hat)

% ------------------------------------------------------------------ %
% This function returns a feedback control that depends on the       %
% maximum likelihood estimate (MLE) of a two-dimensional unknown     %
% parameter in the transition probabilities of a Markov chain.       %
%                                                                    %
% Inputs:   alpha_hat = two-dimensional MLE                          %
%                                                                    %
% Outputs:  u = feedback control, dependent on the MLE alpha_hat.    %
%                                                                    %
% Written by Cody E. Clifton, 2013-04-03.                            %
% ------------------------------------------------------------------ %

if ((alpha1_hat==0.02)&&(alpha2_hat==0.02))
    u = 1;
else
    u = -1;
end % if/else

end % function
```

A.2.8 MLE: linear dependence, 1D, finite set

The following code requires the MATLAB functions provided in Appendices A.2.3, A.2.4, A.2.6, and either A.2.1 or A.2.2.

```
% ------------------------------------------------------------------ %
% Consider a Markov chain whose transition probabilities depend      %
% linearly on a one-dimensional unknown parameter alpha from a       %
% finite set A and a feedback control u from a finite set U. This    %
% code first assumes knowledge of the true value of alpha in order   %
% to successively simulate states of the chain and perform maximum   %
% likelihood estimation of alpha.                                    %
%                                                                    %
% Required functions: a_values.m, b_values.m, control1D.m,           %
%                       MarkovSim1.m or MarkovSim2.m                 %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-28.                            %
% ------------------------------------------------------------------ %

clear % clear all variables

true_alpha = 0.02; % fix the true value of the parameter
A = [0.01, 0.02, 0.03]; % define the set of possible parameter values
N = 100000; % fix the number of simulated observations of the chain
```

```

X = zeros(N+1,1); % initialize the vector of simulated states
alpha_hat = zeros(N,1); % initialize the vector of ML estimates
P = zeros(2); % initialize the transition probability matrix

u = 1; % initialize the control u

X(1) = 1; % fix the initial state of the chain to be '1'

s = size(A,2);
L = zeros(s,1);

for i=1:N
    % assign values to the matrices a and b
    a = a_values(u); b = b_values(u);

    % set P(j,k) = a(j,k)*alpha + b(j,k), j,k=1,2
    P(1,1) = a(1,1)*true_alpha + b(1,1);
    P(1,2) = a(1,2)*true_alpha + b(1,2);
    P(2,1) = a(2,1)*true_alpha + b(2,1);
    P(2,2) = a(2,2)*true_alpha + b(2,2);

    % simulate the "next" state of the Markov chain
    Y = MarkovSim1(2,P,X(i));
    X(i+1) = Y(2);

    a_m = 0; b_m = 0;

    for j=1:2
        for k=1:2
            if ((X(i)==j)&&(X(i+1)==k))
                a_m = a(j,k); b_m = b(j,k);
                break; % exit the loop containing the if-statement
            end % if
        end % for
    end % for

    for j=1:s
        L(j) = L(j) + log(a_m*A(j) + b_m);
    end % for

    % find indices corresponding to values from A that maximize L
    m = find(L==max(L(:)));

    % fix the alpha_hat as the first from A that maximizes L
    alpha_hat(i) = A(m(1));

    u = control1D(alpha_hat(i));
end % for

disp(alpha_hat) % display the vector of ML estimates

n = linspace(1,N,N);
plot(n,alpha_hat) % plot the ML estimates against the vector n
title('Maximum Likelihood Estimation of alpha');
xlabel('n');
ylabel('ML estimate of alpha');
```

A.2.9 MLE: linear dependence, 1D, compact set

The following code requires the MATLAB function provided in either Appendix A.2.1 or Appendix A.2.2.

```
% ------------------------------------------------------------------ %
% Consider a Markov chain whose transition probabilities depend      %
% linearly on a one-dimensional unknown parameter alpha.  This code  %
% first assumes knowledge of the true value of alpha in order to     %
% simulate an observed trajectory of the chain, and then uses this   %
% trajectory to perform maximum likelihood estimation of alpha.      %
%                                                                       %
% Required functions: MarkovSim1.m or MarkovSim2.m                      %
%                                                                       %
% Written by Cody E. Clifton, 2013-04-03.                               %
% --------------------------------------------------------------------- %

clear % clear all variables

true_alpha = 0.02; % fix the true value of the parameter
N = 100000; % fix the number of simulated observations of the chain

X = zeros(N+1,1); % initialize the vector of simulated states
alpha_hat = zeros(N,1); % initialize the vector of ML estimates
P = zeros(2); % initialize the transition probability matrix

u = 0; % initialize the control u, in this case identically zero

X(1) = 1; % fix the initial state of the chain to be '1'

% assign values to the matrices a and b
a = a_values(u); b = b_values(u);

% set P(j,k) = a(j,k)*alpha + b(j,k) for j,k = 1,2
P(1,1) = a(1,1)*true_alpha + b(1,1);
P(1,2) = a(1,2)*true_alpha + b(1,2);
P(2,1) = a(2,1)*true_alpha + b(2,1);
P(2,2) = a(2,2)*true_alpha + b(2,2);

% generate N+1 simulated observations of the state of the chain
X = MarkovSim1(N+1,P,X(1));

for i=1:N
    M = zeros(2); % initialize "index" matrix
    for j=1:i
        if ((X(j)==1)&&(X(j+1)==1))
```

```
            M(1,1) = M(1,1) - 1;
        elseif ((X(j)==1)&&(X(j+1)==2))
            M(1,2) = M(1,2) - 1;
        elseif ((X(j)==2)&&(X(j+1)==1))
            M(2,1) = M(2,1) - 1;
        else
            M(2,2) = M(2,2) - 1;
        end % if/else
    end % for
    Likelihood = @(alpha) (M(1,1)*log(a(1,1)*alpha+b(1,1)) + ...
                           M(1,2)*log(a(1,2)*alpha+b(1,2)) + ...
                           M(2,1)*log(a(2,1)*alpha+b(2,1)));
                           % M(2,2)*log(a(2,2)*alpha+b(2,2))
    alpha_hat(i) = fminbnd(Likelihood,0,0.1);
end % for

disp(alpha_hat) % display the vector of ML estimates

n = linspace(1,N,N);
plot(n,alpha_hat) % plot the ML estimates against the vector n
title('Maximum Likelihood Estimation of alpha');
xlabel('n');
ylabel('ML estimate of alpha');
```

A.2.10 MLE: quadratic dependence, 1D, finite set

The following code requires the MATLAB functions provided in Appendices A.2.3, A.2.4, A.2.6, and either A.2.1 or A.2.2.

```
% ------------------------------------------------------------------ %
% Consider a Markov chain whose transition probabilities depend      %
% quadratically on a one-dimensional unknown parameter alpha from a  %
% finite set A and a feedback control u from a finite set U. This    %
% code first assumes knowledge of the true value of alpha in order   %
% to successively simulate states of the chain and perform maximum   %
% likelihood estimation of alpha.                                    %
%                                                                    %
% Required functions: a_values.m, b_values.m, c_values.m,            %
%                     control1D.m, MarkovSim1.m or MarkovSim2.m      %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-28.                            %
% ------------------------------------------------------------------ %

clear % clear all variables

true_alpha = 0.02; % fix the true value of the parameter

A = [0.01, 0.02, 0.03]; % define the set of possible parameter values
N = 100000; % fix the number of simulated observations of the chain

X = zeros(N+1,1); % initialize the vector of simulated states
alpha_hat = zeros(N,1); % initialize the vector of ML estimates
P = zeros(2); % initialize the transition probability matrix

u = 1; % initialize the control u

X(1) = 1; % fix the initial state of the chain to be '1'

s = size(A,2);
L = zeros(s,1);

for i=1:N
    % assign values to the matrices a, b, and c
    a = a_values(u); b = a_values(u); c = b_values(u);

    % set P(j,k) = a(j,k)*alpha^2 + b(j,k)*alpha + c(j,k), j,k=1,2
    P(1,1) = a(1,1)*true_alpha^2 + b(1,1)*true_alpha + c(1,1);
    P(1,2) = a(1,2)*true_alpha^2 + b(1,2)*true_alpha + c(1,2);
    P(2,1) = a(2,1)*true_alpha^2 + b(2,1)*true_alpha + c(2,1);
    P(2,2) = a(2,2)*true_alpha^2 + b(2,2)*true_alpha + c(2,2);

    % simulate the "next" state of the Markov chain
    Y = MarkovSim1(2,P,X(i));
    X(i+1) = Y(2);

    a_m = 0; b_m = 0; c_m = 0;

    for j=1:2
        for k=1:2
            if ((X(i)==j)&&(X(i+1)==k))
                a_m = a(j,k); b_m = b(j,k); c_m = c(j,k);
                break; % exit the loop containing the if-statement
            end % if
        end % for
    end % for

    for j=1:s
        L(j) = L(j) + log(a_m*A(j)^2 + b_m*A(j) + c_m);
    end % for

    % find indices corresponding to values from A that maximize L
    m = find(L==max(L(:)));

    % fix the alpha_hat as the first value from A that maximizes L
    alpha_hat(i) = A(m(1));

    u = control1D(alpha_hat(i));
end % for

disp(alpha_hat) % display the vector of ML estimates

n = linspace(1,N,N);
plot(n,alpha_hat) % plot the ML estimates against the vector n
title('Maximum Likelihood Estimation of alpha');
```

```
xlabel('n');
ylabel('ML estimate of alpha');
```

A.2.11 MLE: quadratic dependence, 1D, compact set

The following requires the MATLAB function provided in either Appendix A.2.1 or Appendix A.2.2.

```
% ------------------------------------------------------------------ %
% Consider a Markov chain whose transition probabilities depend      %
% quadratically on a one-dimensional unknown parameter alpha.  This  %
% code first assumes knowledge of the true value of alpha in order   %
% to simulate an observed trajectory of the chain, and then uses     %
% this trajectory to perform maximum likelihood estimation of alpha. %
%                                                                    %
% Required functions: MarkovSim1.m or MarkovSim2.m                   %
%                                                                    %
% Written by Cody E. Clifton, 2013-04-03.                            %
% ------------------------------------------------------------------ %

clear % clear all variables

true_alpha = 0.02; % fix the true value of the parameter
N = 100000; % fix the number of simulated observations of the chain

X = zeros(N+1,1); % initialize the vector of simulated states
alpha_hat = zeros(N,1); % initialize the vector of ML estimates
P = zeros(2); % initialize the transition probability matrix

u = 0; % initialize the control u, in this case identically zero

X(1) = 1; % fix the initial state of the chain to be '1'

% assign values to the matrices a, b, and c
```

```
a = a_values(u); b = a_values(u); c = b_values(u);

% set P(j,k) = a(j,k)*alpha^2 + b(j,k)*alpha + c(j,k), j,k=1,2
P(1,1) = a(1,1)*true_alpha^2 + b(1,1)*true_alpha + c(1,1);
P(1,2) = a(1,2)*true_alpha^2 + b(1,2)*true_alpha + c(1,2);
P(2,1) = a(2,1)*true_alpha^2 + b(2,1)*true_alpha + c(2,1);
P(2,2) = a(2,2)*true_alpha^2 + b(2,2)*true_alpha + c(2,2);

% generate N+1 simulated observations of the state of the chain
X = MarkovSim1(N+1,P,X(1));

for i=1:N
    M = zeros(2); % initialize "index" matrix
    for j=1:i
        if ((X(j)==1)&&(X(j+1)==1))
            M(1,1) = M(1,1) - 1;
        elseif ((X(j)==1)&&(X(j+1)==2))
            M(1,2) = M(1,2) - 1;
        elseif ((X(j)==2)&&(X(j+1)==1))
            M(2,1) = M(2,1) - 1;
        else
            M(2,2) = M(2,2) - 1;
        end % if/else
    end % for
    Likelihood = @(alpha)  ...
                (M(1,1)*log(a(1,1)*alpha^2+b(1,1)*alpha+c(1,1)) + ...
                 M(1,2)*log(a(1,2)*alpha^2+b(1,2)*alpha+c(1,2)) + ...
                 M(2,1)*log(a(2,1)*alpha^2+b(2,1)*alpha+c(2,1)));
                 % M(2,2)*log(a(2,2)*alpha^2+b(2,2)*alpha+c(2,2))
    alpha_hat(i) = fminbnd(Likelihood,0,0.1);
end % for

disp(alpha_hat) % display the vector of ML estimates

n = linspace(1,N,N);
```

```
plot(n,alpha_hat) % plot the ML estimates against the vector n
title('Maximum Likelihood Estimation of alpha');
xlabel('n');
ylabel('ML estimate of alpha');
```

A.2.12 MLE: linear dependence, 2D, finite set

The following requires the MATLAB functions provided in Appendices A.2.3, A.2.4, A.2.7, and either A.2.1 or A.2.2.

```
% ------------------------------------------------------------------- %
% Consider a Markov chain whose transition probabilities depend on a %
% two-dimensional unknown parameter alpha.  This code first assumes  %
% knowledge of the true value of alpha in order to simulate an       %
% observed trajectory of the chain, and then uses this trajectory to %
% perform maximum likelihood estimation of alpha.                    %
%                                                                    %
% Required functions: FiniteArgMax2D.m and                           %
%                      MarkovSim1.m or MarkovSim2.m                  %
%                                                                    %
% Written by Cody E. Clifton, 2013-03-28.                            %
% ------------------------------------------------------------------- %

clear % clear all variables

true_alpha = [0.01, -0.02]; % fix the true value of the parameter

% define the set of admissible values for the 1st component of alpha
A1 = [0.01, 0.02, 0.03];
% define the set of admissible values for the 2nd component of alpha
A2 = [-0.01, -0.02, -0.03];

N = 100000; % fix the number of simulated observations of the chain

```

```
X = zeros(N+1,1); % initialize the vector of simulated states
alpha_hat = zeros(N,2); % initialize the vector of ML estimates
P = zeros(2); % initialize the transition probability matrix

u = 1; % initialize the control u

X(1) = 1; % fix the initial state of the chain to be '1'

s1 = size(A1,2);
s2 = size(A2,2);
L = zeros(s1,s2);

for i=1:N
    % assign values to the matrices a, b, and c
    a = a_values(u); b = a_values(u); c = b_values(u);

    % set P(j,k)=a(j,k)*alpha(1)+b(j,k)*alpha(2)+c(j,k), for j,k=1,2
    P(1,1) = a(1,1)*true_alpha(1) + b(1,1)*true_alpha(2) + c(1,1);
    P(1,2) = a(1,2)*true_alpha(1) + b(1,2)*true_alpha(2) + c(1,2);
    P(2,1) = a(2,1)*true_alpha(1) + b(2,1)*true_alpha(2) + c(2,1);
    P(2,2) = a(2,2)*true_alpha(1) + b(2,2)*true_alpha(2) + c(2,2);

    % simulate the "next" state of the Markov chain
    Y = MarkovSim1(2,P,X(i));
    X(i+1) = Y(2); %

    a_m = 0; b_m = 0; c_m = 0;

    for j=1:2
        for k=1:2
            if ((X(i)==j)&&(X(i+1)==k))
                a_m = a(j,k); b_m = b(j,k); c_m = c(j,k);
                break; % exit the loop containing the if-statement
            end % if
        end % for
    end % for

    for j=1:s1
        for k=1:s2
            L(j,k) = L(j,k) + log(a_m*A1(j) + b_m*A2(k) + c_m);
        end % for
    end % for

    % find pairs of indices of values from A1, A2 that maximize L
    [m1,m2] = find(L==max(L(:)));

    % fix the first pair of values from A1, A2 that maximizes L
    alpha_hat(i,1) = A1(m1(1));
    alpha_hat(i,2) = A2(m2(1));

    u = control2D(alpha_hat(i,1),alpha_hat(i,2));
end % for

disp(alpha_hat) % display the vector of ML estimates

n = linspace(1,N,N);
plot(n,alpha_hat) % plot the ML estimates against the vector n
title('Maximum Likelihood Estimation of alpha');
xlabel('n');
ylabel('ML estimate of alpha');
legend('alpha1','alpha2');
```