University College London

MSc CSML

Methodology for Joint Estimation of Static and Dynamic Parameters in Bayesian CVAR models

---

Author:
Jonathan T. J. Cox

Supervisor:
Dr. Gareth. W. Peters

31st August, 2013

Abstract

The high-dimensional, generally non-linear, matrix-variate setting of cointegrated Vector Autoregressive (CVAR) models presents a challenging context for performing inference. The development, implementation and testing of several state-of-art Particle Markov Chain Monte Carlo (PMCMC) methods are discussed for the purpose of joint estimation of latent states and static parameters in Bayesian CVAR models. The effectiveness of the estimation performed by the samplers in a variety of static and dynamic linear and non-linear systems is explored. The use of PMCMC samplers is demonstrated for a class of cointegration model with innovation error covariance structure which dynamically evolves according to a latent square Bessel process.

Contents

Chapter 1

Introduction

Over the last 25 years the concept of cointegration has been widely applied in the field of financial econometrics in the areas of multivariate time series analysis and macroeconomics. As well as finding applications there, the field in which the concept originated, it has gone on to be applied in the apparently unrelated fields of biology, political science, and, increasingly, quantitative finance and trading. The concept is important because it provides a way to isolate and identify the influence of common, stable, long-term stochastic trends on time-series variables. The variables are allowed to deviate from their inherent relationships in the short term, but to retain the associations in the long term. Examples of financial variables which display a cointegration relationship abound, with obvious examples being spot and futures prices for a particular asset, and interest rate yields for the same currency but varying maturities.

The finance industry has in general been slow to replace the outdated (but ingrained) use of the limited concept of correlation with the more powerful, appropriate and stable concept of cointegration. The ubiquitous use of short-term correlation measures between de-trended time-series in investment risk/return analyses guarantees the loss of valuable information contained in the common trends of the prices. An analysis based on cointegration is more likely to be stable in the long-run and to provide a better understanding of risk. Similarly a trading strategy based on the concept is likely to require reduced rebalancing in its hedging strategy and thus its transaction costs, and have more attractive long-run performance characteristics.

This thesis focuses on the development of sophisticated stochastic simulation methodologies for the estimation of cointegrated models in a Bayesian setting, with the aim of applying these in future work to financial time-series. The reason that such sophisticated methods are necessary is that interest lies not only in performing inference for the unknown matrix-variate static model parameters, but in elucidating the hidden, latent, vector-valued states driving the cointegrated systems. This is a very challenging problem in the context of cointegrated models both due to the very high-dimensionality of the systems, the matrix-variate formulation which is highly sensitive to changes in certain model components and the potential for the underlying systems to have non-linear specifications.

By representing the cointegrated models in state-space form the deep underlying connection with the techniques of filtering becomes clear. Filtering is concerned with estimating the likely states of a system from noisy observations. There are many varieties of filter and the most efficient or effective type for a particular state-space system depends on the underlying dynamics of that system. A very general and powerful class of techniques, which can be applied to the filtering problem in both linear and non-linear situations, is that of *Sequential Monte Carlo* (SMC) methods. For situations involving non-linearity these are far superior to their traditional linear Gaussian cousins, the Kalman Filters.

Markov Chain Monte Carlo (MCMC) methods are well established techniques for sampling from otherwise intractable distributions. Recently, state-of-the-art methods have been developed which embed SMC filters within MCMC samplers for the *joint* estimation of static parameters and latent

states in complex non-linear dynamical systems. These *Particle Markov Chain Monte Carlo* (PMCMC) methods appeared in the literature in 2010 and many aspects of their behaviour in complex practical applications remain open research questions.

This thesis records the development, implementation and testing of a series of PMCMC samplers capable of performing the required joint estimation. In one sense the thesis, although a complete description of the samplers, their construction and practical implementation, represents the first phase of a larger project which will culminate in a paper detailing the methodology for estimation of an extended class of cointegrated models[1] that incorporate stochastic volatility via inverse Wishart processes. Future work will also focus on the application of these models in a quantitative trading scenario.

[1] A simple example of which is included in the project as **Model $\mathbf{M}_2$**.

Chapter 2

Models for Cointegration

2.1 Introduction

This chapter introduces multivariate models for time series and the key related concepts of integrated series and cointegrated vector autoregressive (CVAR) models. Ordinary VAR models are stationary and thus do not admit trends or shifts in the mean or the covariances, nor deterministic seasonal patterns. We will consider nonstationary processes of a very specific type; these will be allowed to have stochastic trends and are then called *integrated*. Under the constraint that some of the variables move together in the long-run, although they have stochastic trends, they are driven by a common source of stochasticity and are called *cointegrated*.

Section 2.2 describes the concept of cointegration and the conditions necessary for its existence in the context of VAR models. Section 2.3 briefly describes the reasons for the origin of cointegration and the applications the concept has found in the finance industry. Section 2.4 gives details of two popular formats for representing cointegrated models: Phillips' triangular representation and the Vector Error Correction Model (VECM) representation of Engle and Granger. In section 2.5 the Johansen maximum likelihood method is briefly described for the purpose of parameter estimation in CVAR models. Finally, the chapter concludes with a short case study (Section 2.6) that uses the Johansen method to determine the cointegration rank for each of five synthetic VECMs.

2.2 Cointegrated VAR models

A *vector autoregressive* (VAR) process is the multivariate analogue of a univariate autoregressive (AR) process. The VAR model of order p, or VAR(p), can be written:

$$\mathbf{x}_t = \boldsymbol{\mu} + A_1\mathbf{x}_{t-1} + \cdots + A_p\mathbf{x}_{t-p} + \boldsymbol{\epsilon}_t, \tag{2.1}$$

where $\mathbf{x}_t = (x_{1t}, \ldots, x_{nt})'$ is a $(n \times 1)$ random vector, the A_i are fixed $(n \times n)$ coefficient matrices, $\boldsymbol{\mu} = (\mu_1, \ldots, \mu_n)'$ is a fixed $(n \times 1)$ vector of intercept terms and $\boldsymbol{\epsilon}_t = (\epsilon_{1t}, \ldots, \epsilon_{nt})'$ is an n-dimensional *white noise* or *innovation process*[1].

A *stationary process*[2] is one that has time invariant first and second moments, and thus does not have trends or changing variances. A VAR process is stationary if the determinantal polynomial of the VAR operator has all its roots outside the complex unit circle. Features such as trending means (and heteroskedasticity) are common in real financial time series, therefore models built from stationary processes are insufficient to capture some of the main features of interest. Typically the technique of taking successive differences may be used to remove trends and create a stationary differenced series; however, if we are required to analyse the original variables rather than these differences, models will be needed that can accommodate the nonstationarity.

[1] A white noise process has: $\mathbb{E}(\epsilon_t) = 0$, $\mathbb{E}(\epsilon_t, \epsilon_t') = \Sigma_\epsilon$, and $\mathbb{E}(\epsilon_t, \epsilon_s) = 0$ for $s \neq t$.

[2] Here we are discussing *covariance* stationary processes, there are stronger forms of this concept, see [Lütkepohl, 2007] for details.

Random walk-like behaviour is observed for univariate AR processes such as

$$x_t = \mu + a_1 y_{t-1} + \cdots + a_p x_{t-p} + \epsilon_t,$$

if $1 - a_1 z - \cdots - a_p z^p$ has a root for $z = 1$; that is, an *integrated* process of order 1. The presence of d unit roots in the AR operator of a univariate process x_t will create an integrated process of order d, denoted $x_t \sim I(d)$. Generally, such a process can be made stationary by differencing d times: $\Delta^d x_t = (1 - L)^d x_t$ is stationary (the differencing operator, Δ, is expressed in terms of the lag operator, L, as $\Delta = (1 - L)$, and $Lx_t = x_{t-1}$).

The VAR in (2.1) can be re-written as

$$A(L)\mathbf{x}_t = \boldsymbol{\epsilon}_t,$$

where $A(L) = I_n - A_1 L - \cdots - A_p L^p$, and L is the lag operator. A VAR process can generate deterministic and stochastic trends if the determinantal polynomial of the VAR operator has roots on the complex unit circle. It is sufficient to allow for unit roots ($z = 1$) to obtain trending behaviour of the variables. In the case of a vector process having unit roots some of the processes can have common trends so that they move together to some extent. These are then called *cointegrated*.

Cointegration represents the presence of equilibrium relationships between sets of time series variables. Writing the set of variables as a vector $x_t = (x_{1t}, \ldots, x_{nt})'$ the long-run equilibrium can be expressed as

$$\boldsymbol{\beta}'\mathbf{x}_t = \boldsymbol{\beta}_1 x_{1t} + \ldots \boldsymbol{\beta}_n x_{nt} = 0, \tag{2.2}$$

where $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots \boldsymbol{\beta}_n)'$. For a particular time period the relationship may not be satisfied exactly, but we may have $\boldsymbol{\beta}'\mathbf{x}_t = \mathbf{e}_t$, where $\mathbf{e}_t$ is a stochastic variable representing deviations from the equilibrium. It is possible in this arrangement that the variables wander as a group, i.e. they are driven by a common stochastic trend. In other words that each of them individually may be integrated and therefore nonstationary, but that there exists a linear combination of the variables which is stationary; such a group is termed cointegrated. A synthetically simulated cointegrated bivariate time series is shown below in Figure 2.1.

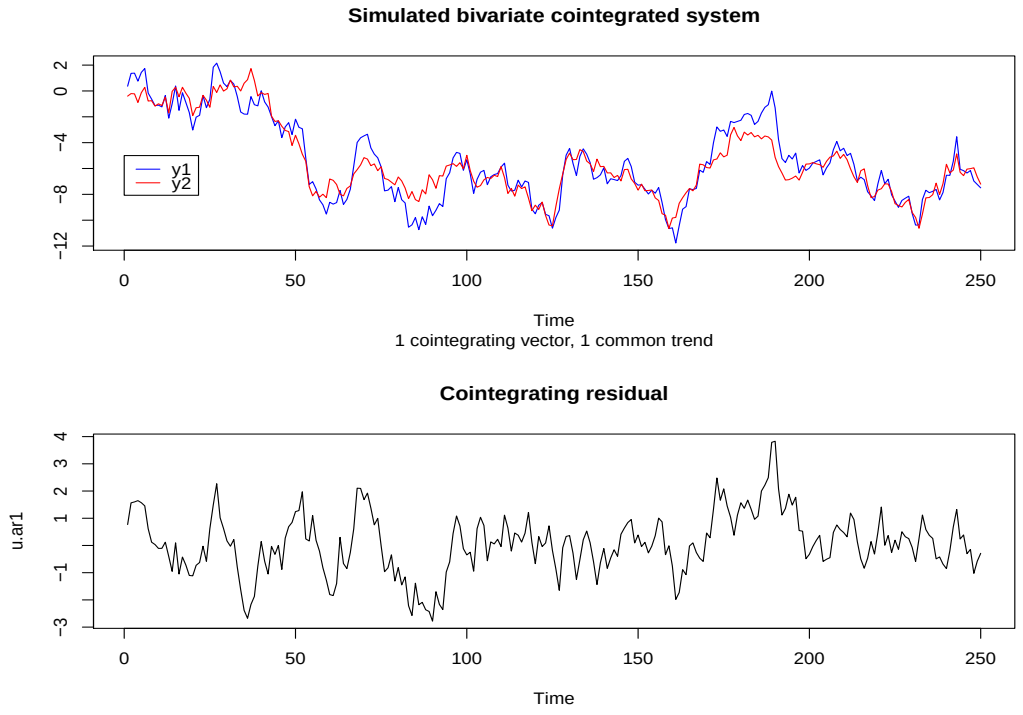


Figure 2.1: Simulated bivariate cointegrated time system: the lower plot shows the cointegrating residual ($\mathbf{e}_t$).

The concept of cointegration was introduced by Engle and Granger [Engle and Granger, 1987]. The linear combination written in (2.2) is termed the *cointegrating relationship*, and the vector $\boldsymbol{\beta}$ is called the *cointegrating vector*. Usually, the variables in an n-dimensional process $\mathbf{x}_t$ are called *cointegrated of order* (d, b), or $\mathbf{x}_t \sim CI(d, b)$, if all elements of $\mathbf{x}_t$ are $I(d)$ and a linear combination $z_t := \boldsymbol{\beta}'\mathbf{x}_t$ with $\boldsymbol{\beta} \neq 0$ such that z_t is $I(d-b)$. For example, if all components of $\mathbf{x}_t$ are $I(1)$ and $\boldsymbol{\beta}'\mathbf{x}_t$ is stationary $I(0)$, then $\mathbf{x}_t \sim CI(1, 1)$.

A cointegrating vector $\boldsymbol{\beta}$ is not unique; multiplying by a non-zero constant produces a further cointegrating vector. There may be various linearly independent cointegrating vectors: for example, given 6 variables under study there may be cointegrating relationships between the first and last sub-groups containing 3 variables. Alternatively there may be cointegrating relationships involving the whole group of 6 variables.

2.3 Cointegration in Finance

By far the most common application domains for the cointegration concept are econometrics and finance. Any system of financial asset prices with a mean-reverting spread will have some degree of cointegration, despite the fact that the causalities revealed by these relationships contradict the efficient financial markets hypothesis. *Mean reverting* is the term finance practitioners use to refer to stationary processes. If spreads are mean-reverting, assets will be tied together in the long-term by a common stochastic trend.

Equilibrium relationships are expected to exist between financial variables of various categories and for various reasons, for example: between spot and futures prices for commodities or assets, between exchange rates and relative prices in various countries, between equity prices and dividends, and between the equities within an index. The origin of these relationships is often clear: spot and futures prices represent prices for the same underlying asset at different points in time, relative prices and exchange rates are tied to future expectations of interest rates in various countries, and dividends and equity prices refer to different measures of value of the the same underlying company. The result of these underlying connections is that new information will affect the prices in similar ways.

2.3.1 Cointegration and correlation

Despite the fact that the concept of cointegration has been written about extensively since its introduction in 1987, and can be considered a key concept of the field of modern time series econometrics, its application to 'real-world' applications in finance and investment management has been relatively slow to gain ground. The reason for this is that, traditionally, the starting point for portfolio risk/return management has been the analysis of *correlation* amongst the historical security returns, whereas cointegration considers raw prices, yields or rates. The familiar concept of correlation, although related, is not the same thing as cointegration. Highly correlated returns do not imply securities with high cointegration and vice versa.

Correlation is a typically unstable, short-term measure of the co-movements of returns; hedging or quantitative investment strategies which depend upon measures of correlation typically require frequent rebalancing. Conversely, since cointegration measures long-term co-movements between prices (which may occur even when short term correlations are low), portfolio hedging strategies which depend on cointegration will require less rebalancing and be more effective in the long term. In a sense cointegration generalises the concept of correlation to the case of non-stationary time series [Alexander, 1999]

2.3.2 Applications of Cointegration in Investment

A number of opportunities exist to apply knowledge of the existence of cointegration to the tasks of reducing investment risk and increasing investment performance in specific situations in finance. Simple examples are provided by the classic *index tracking* strategies, or the *long-short* equity market neutral strategies, see for example [Alexander and Dimitriu, 2002]. We will describe both of these to give a flavour of the possible applications of cointegration to investment management.

Index Tracking Strategy: This strategy aims to replicate a benchmark index in terms of return and volatility. A cointegration analysis can be the starting point for constructing an index replicating portfolio; such portfolios are expected to have similar returns, volatilities, and high correlation with the index. [Alexander and Dimitriu, 2002] give three reasons for using cointegration (rather than correlation) to construct an equity index tracking portfolio:

- The tracking error (between replicating portfolio and the benchmark) is, by construction, mean reverting.
- The stock weights in the portfolio are more stable, and thus require less frequent re-balancing.
- Better use of information, in particular the information contained in the stock prices.

Implementing the index tracking strategy is a two stage process; the first stage consists of selecting the stocks to include in the tracking portfolio, and the second stage involves determining the holding in each stock based - typically on a correlation analysis or - here on a cointegration analysis. The selection stage is carried out by a portfolio manager using proprietary methods or technical analysis; this stage does not have special features in a cointegration based approach. However, the second stage is different in a cointegration based approach compared to the standard correlation based approaches. The stock weights in the portfolio are estimated based on ordinary least squares (OLS) coefficients of the cointegration equation that regresses the index log price on the selected stock log prices over a given period [Alexander and Dimitriu, 2002]:

$$\log I_t = c_1 + \underbrace{\sum_{k=1}^{N} c_{k+1} \log P_{k,t}}_{\text{Tracking Portfolio}} + \mathsf{e}_t,$$

where I_t and $P_{k,t}$ are the index price and the price of stock k at time t respectively. The residuals e_t, above, are stationary if and only if the tracking portfolio and the index are cointegrated. The procedure provides a unique portfolio of stocks for each selected holding and calibration period.

Long-short Market Neutral Strategy: This strategy builds on the cointegration-constructed index tracking portfolio, and aims to generate returns under all market conditions. The combination of index tracking and long-short market neutral strategies is designed to improve upon the properties of the basic component strategies. The strategy also depends on the tracking ability of cointegrated portfolios, but uses two replicating portfolios: a plus and a minus portfolio which track 'enhanced' benchmarks. Simply put, one of these, the plus portfolio, is expected to increase in value overall, where the minus is expected to decrease in value. By holding the plus portfolio long, and selling the minus portfolio short the strategy is expected to generate returns on the spread between the two portfolios.

2.4 Representation

The literature on estimation and inference of cointegrated systems contains three extensively used representations for cointegrating models: Phillips' *triangular representation*, Stock and Watson's *common trends representation* and Engle and Granger's *vector error correction model* (VECM)

(see for example [Phillips, 1991], [Stock and Watson, 1988] and [Engle and Granger, 1987]). In the following section the first and last of these are explained in more detail.

2.4.1 Triangular Representation

The triangular representation was first introduced in [Phillips, 1991]. Consider a cointegrating vector with true rank r, i.e. $\text{rank}(\boldsymbol{\beta}) = r$, whereby there are r linearly independent rows in $\boldsymbol{\beta}$. By rearranging the order of the variables we can always ensure that the first r rows of $\boldsymbol{\beta}$ are linearly independent. Therefore, the upper $(r \times r)$ sub-matrix consisting of the first r rows of $\boldsymbol{\beta}$ is non-singular:

$$\boldsymbol{\beta}^{\star} = \begin{bmatrix} I_r \\ \boldsymbol{\beta}_{(n-r)} \end{bmatrix}, \tag{2.3}$$

where $\boldsymbol{\beta}_{(n-r)}$ has dimension $((n-r) \times r)$. This normalisation ensures a unique cointegration matrix and will be discussed in more detail in later sections.

If the normalization in (2.3) is made we can re-write the cointegrating system as

$$\begin{aligned} \mathbf{x}_t^{(1)} &= -\boldsymbol{\beta}'_{(n-r)}\mathbf{x}_t^{(2)} + \mathbf{e}_t^{(1)} \\ \Delta\mathbf{x}_t^{(2)} &= \mathbf{e}_t^{(2)}, \end{aligned} \tag{2.4}$$

where $\mathbf{x}_t^{(1)}$ and $\mathbf{e}_t^{(1)}$ have dimension $(r \times 1)$, $\mathbf{x}_t^{(2)}$ and $\mathbf{e}_t^{(2)}$ have dimension $((n-r) \times 1)$, and $\mathbf{e}_t = (\mathbf{e}_t^{(1)'}, \mathbf{e}_t^{(2)'})'$ is a stationary process. There cannot be any cointegrating relationships between the components of the $\mathbf{x}_t^{(2)}$ subsystem, since otherwise there would be more than r linearly independent cointegrating relationships (and this would make the rank larger than r). The variables in the $\mathbf{x}_t^{(2)}$ part represent stochastic trends in the system, see [Lütkepohl, 2007] chapter 6.

2.4.2 Vector Error Correcting Model

Before the introduction of the concept of cointegration, the closely related *error correction models* were reported in the econometrics literature: see [Phillips, 1954] which introduced the error correction concept to the field of econometrics, [Sargan, 1964], and also [Hendry and Richard, 1983]. In error correction models the changes in a particular variable are dependent upon deviations from an equilibrium relation. A simple example concerns the price of an asset in two different markets, x_{1t} and x_{2t}, where the equilibrium relationship between these prices is $x_{1t} = \boldsymbol{\beta} x_{2t}$. Changes to x_{1t} depend on deviations from this equilibrium in the preceding time period, $t-1$:

$$\Delta x_{1t} = \boldsymbol{\alpha}_1(x_{1,t-1} - \boldsymbol{\beta}_1 x_{2,t-1}) + \epsilon_{1t},$$

and similarly changes in x_{2t} may also be dependent on deviations from the same equilibrium:

$$\Delta x_{2t} = \boldsymbol{\alpha}_2(x_{1,t-1} - \boldsymbol{\beta}_1 x_{2,t-1}) + \epsilon_{2t}.$$

For a mode general ECM the values of Δx_{it} may also depend upon lagged changes in both variables:

$$\begin{aligned} \Delta x_{1t} &= \boldsymbol{\alpha}_1(x_{1,t-1} - \boldsymbol{\beta}_1 x_{2,t-1}) + \boldsymbol{\gamma}_{11,1}\Delta x_{1,t-1} + \boldsymbol{\gamma}_{12,1}\Delta x_{1,t-1}\Delta x_{2,t-1} + \epsilon_{1t} \\ \Delta x_{2t} &= \boldsymbol{\alpha}_2(x_{1,t-1} - \boldsymbol{\beta}_1 x_{2,t-1}) + \boldsymbol{\gamma}_{21,1}\Delta x_{1,t-1} + \boldsymbol{\gamma}_{22,1}\Delta x_{1,t-1}\Delta x_{2,t-1} + \epsilon_{2t}. \end{aligned} \tag{2.5}$$

If both $x_{1,t}$ and x_{2t} are $I(1)$ variables the above represents a cointegrating relationship for $x_{1t} - \boldsymbol{\beta}_1 x_{2t}$.

We can re-write (2.5) using vector-matrix notation to give:

$$\Delta\mathbf{x}_t = \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{x}_{t-1} + \boldsymbol{\Gamma}_1\Delta\mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t.$$

It is straightforward to see that this can also be re-written in a VAR(2) form (by writing out $\Delta \mathbf{x}_t = \mathbf{x}_t - \mathbf{x}_{t-1}$ etc).

The *Granger representation theorem* [Engle and Granger, 1987] states that a multivariate integrated process is cointegrated if and only if it can be represented in the ECM form with certain restrictions. The general form for the *vector error correction model* (VECM) representation of an n-dimensional VAR process of order p is

$$\Delta \mathbf{x}_t = \mathbf{\Pi} \mathbf{x}_{t-1} + \sum_{i=1}^{p-1} \mathbf{\Gamma}_i \Delta \mathbf{x}_{t-1} + \boldsymbol{\phi} \mathbf{d}_t + \boldsymbol{\epsilon}_t, \tag{2.6}$$

where:

$\Delta \mathbf{x}_t \in \mathbb{R}^n$	$= \mathbf{x}_t - \mathbf{x}_{t-1}$,
$\mathbf{\Pi}$	is the $(n \times n)$ long-run multiplier matrix,
$\mathbf{\Gamma}_i$	is the ith$(n \times n)$ lag matrix,
$\mathbf{d}_t$	is an $(n \times 1)$ vector of deterministic terms, defined as a polynomial in time t,
$\boldsymbol{\phi}$	is an $(n \times n)$ matrix,
$\boldsymbol{\epsilon}_t$	is an $(n \times 1)$ i.i.d. multivariate, correlated vector of errors.

The cointegration properties of the vector model depend upon the rank, r, of the *long-run multiplier matrix*, $\mathbf{\Pi}$. If the rank, r is equal to zero, then $\mathbf{x}_t$ has a stable VAR$(p-1)$ representation and exhibits no cointegrating relationships. If $r = n$, that is the matrix $\mathbf{\Pi}$ is full rank, then the VAR operator has no unit roots and $\mathbf{x}_t$ is a stable VAR(p) stationary process. If, however, the rank, r, is intermediate, $0 < r < n$, then the VECM process is cointegrated. In this case $\mathbf{\Pi}$ can be written as a matrix product $\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$ with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ both of dimension $(n \times r)$ and both of rank r. The matrix $\boldsymbol{\beta}$ is still called the *cointegrating matrix* and its columns are the *cointegrating vectors* of the process. The matrix $\boldsymbol{\alpha}$ is called the *loading matrix*, and the $\boldsymbol{\beta}'\mathbf{x}_t$ are called the *common trends*.

Under these conditions it is possible to re-write (2.6) as:

$$\Delta \mathbf{x}_t = \boldsymbol{\alpha}\boldsymbol{\beta}'\mathbf{x}_{t-1} + \sum_{i=1}^{p-1} \mathbf{\Gamma}_i \Delta \mathbf{x}_{t-1} + \boldsymbol{\phi} \mathbf{d}_t + \boldsymbol{\epsilon}_t. \tag{2.7}$$

It is important to note that the decomposition of the $(n \times n)$ long-run multiplier matrix $\mathbf{\Pi}$ into the product of two $(n \times r)$ matrices, $\mathbf{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$ is not unique. For any non-singular $(r \times r)$ matrix Q we can define $\boldsymbol{\alpha}^* = \boldsymbol{\alpha}Q'$ and $\boldsymbol{\beta}^* = \boldsymbol{\beta}Q^{-1}$ and get $\mathbf{\Pi} = \boldsymbol{\alpha}^*\boldsymbol{\beta}^*$. This demonstrates that the cointegration relationships are not unique. However, as mentioned in the previous section, it is possible to impose restrictions on the $\boldsymbol{\alpha}$ and/or $\boldsymbol{\beta}$ to provide unique relations.

In situations where cointegration exists, the VECM representation will generate superior forecasts than the corresponding first-differenced form representation, especially over medium and long time horizons. The reason for this is that under cointegration, $\boldsymbol{z}_t = \boldsymbol{\beta}'\mathbf{x}_t$, will have finite forecast error variance, whereas any other linear combination of the forecasts of the series in $\mathbf{x}_t$ will have infinite variance [Engle and Yoo, 1987].

2.5 Parameter Estimation

Various methods for estimating the parameters of a multivariate cointegration model have been developed, including a number based on least squares estimators. Here we focus on the most popular maximum likelihood based estimator: the Johansen Procedure.

2.5.1 Johansen Maximum Likelihood Procedure

Maximum Likelihood (ML) estimation for the parameters of Vector Autoregressive (VAR) models was first developed by Søren Johansen in 1991 [Johansen, 1991] and as such the ML method is often known as the Johansen method.

Consider an Vector Error Correction Model of order p,

$$\Delta \mathbf{x}_t = \mathbf{\Pi} \mathbf{x}_{t-1} + \sum_{i=1}^{p-1} \mathbf{\Gamma}_i \Delta \mathbf{x}_{t-1} + \boldsymbol{\phi} \mathbf{d}_t + \boldsymbol{\epsilon}_t, \qquad \boldsymbol{\epsilon}_t \sim MVN(0, \Sigma), \tag{2.8}$$

where, the parameters of the VECM are as defined in Section (2.4.2). The aim of the Johansen method is to determine a linear combination $Z_t = \beta' \mathbf{x}_t$ which is $I(0)$. Equation (2.8) can also be represented by,

$$Z_{0t} = \alpha \beta' Z_{1t} + \Psi Z_{2t} + \epsilon_t, \tag{2.9}$$

$$\epsilon_t = Z_{0t} - \alpha \beta' Z_{1t} + \Psi Z_{2t}, \tag{2.10}$$

where:

$$\begin{array}{ll} Z_{0t} & = \Delta \mathbf{x}_t, \\ Z_{1t} & = \mathbf{x}_{t-1}, \\ \Psi & = (\mathbf{\Gamma}_1, \cdots, \mathbf{\Gamma}_{p-1}, \boldsymbol{\phi}), \\ Z_{2t} & = (\Delta \mathbf{x}_t, \cdots, \Delta \mathbf{x}_{t-p+1}, \mathbf{d}_t)', \\ \epsilon_t & \text{is iid multivariate Gaussian noise with covariance } \Sigma. \end{array}$$

By assuming that the ϵ_t are iid samples from a multivariate Gaussian distribution with zero mean and covariance Σ, we can solve for the model parameters by maximizing the log likelihood function for a data sample size T of Equation (2.9):

$$\begin{aligned} L(\alpha, \beta, \Psi, \Sigma) &= \frac{1}{\sqrt{2\pi\Sigma}} \exp\left(-\frac{1}{2} \sum_{t=1}^{T} \epsilon_t' \Sigma^{-1} \epsilon_t \right) \\ \ln \mathcal{L}(\alpha, \beta, \Psi, \Sigma) &= -\frac{nT}{2} \ln 2\pi - \frac{T}{2} \ln |\Sigma| - \frac{1}{2} \sum_{t=1}^{T} \epsilon_t' \Sigma^{-1} \epsilon_t \\ &= -\frac{nT}{2} \ln 2\pi - \frac{T}{2} \ln |\Sigma| \\ &\quad - \frac{1}{2} \sum_{t=1}^{T} \left(Z_{0t} - \alpha \beta' Z_{1t} + \Psi Z_{2t} \right)' \Sigma^{-1} \left(Z_{0t} - \alpha \beta' Z_{1t} + \Psi Z_{2t} \right). \end{aligned} \tag{2.11}$$

By working with this form we can arrive at ML estimators for the ECM parameters. The ML estimators are summarised in the following table, and the derivation of these is provided in Appendix A.

Estimator	Form
$\hat{\alpha}$	$S_{01}\beta(\beta' S_{11} \beta)^{-1}$
$\hat{\beta}$	$\left[\mathbf{v}^1, \ldots, \mathbf{v}^r\right]' S_{11}^{-1/2}$
$\hat{\Omega}$	$S_{00} - S_{01}\beta(\beta' S_{11}' \beta)^{-1} \beta' S_{01}$
$\hat{\Psi}$	$M_{02} M_{12}^{-1} - \alpha \beta' M_{12} M_{22}^{-1}$
Maximum Likelihood	$\propto \lvert S_{00} \rvert \prod_{i=1}^{r} (1 - \hat{\lambda}_i)$

Table 2.1: ECM parameter estimators from the Johansen Maximum Likelihood Method

Where we have defined:

- Product moment matrices: $M_{ij} = \frac{1}{T}\sum_{t=1}^{T} Z_{it}Z'_{jt}$, for $i, j = 0, 1, 2$.
- Auxilliary regressions: $Z_{0t} = M_{02}M_{22}^{-1}Z_{2t} + R_{0t}$, and $Z_{1t} = M_{12}M_{22}^{-1}Z_{2t} + R_{1t}$, where R_{0t} and R_{1t} are the residuals.
- Residual sum of squares matrices: $S_{ij} = \frac{1}{T}\sum_{t=1}^{T} R_{it}R'_{jt}$ for $i, j = 0, 1$.
- $\lambda_1 \geq \lambda_2 \geq \ldots \lambda_K$ are the eigenvalues of $S_{11}^{-1/2}S_{10}S_{00}^{-1}S_{01}S_{11}^{-1/2}$.

Asymptotic Properties

The Maximum Likelihood (ML) estimators of the parameters of the Vector Error Correction Model in Equation (2.8) are consistent and have the following central limit theorem [Lütkepohl, 2007],

$$\sqrt{T}\text{vec}\left([\hat{\Pi} : \hat{\Gamma}] - [\Pi : \Gamma]\right) \xrightarrow{d} N(0, \Omega_{co}),$$

where:

$$\Omega_{co} = \left(\begin{bmatrix}\beta & 0\\ 0 & I_{np-n}\end{bmatrix}\Omega^{-1}\begin{bmatrix}\beta' & 0\\ 0 & I_{np-n}\end{bmatrix}\right),$$

and

$$\sqrt{T}\text{vech}\left(\tilde{\Omega}_u - \Omega_u\right) \xrightarrow{d} N(0, 2D_n^{+}\left(\Omega_u \otimes \Omega_u\right)D_n^{+'}),$$

where D_n is the duplication matrix and $D_n^{+} = (D_n^{'}D_n)^{-1}D_n^{'}$, the Moore-Penrose generalised inverse of D_n. Also $\tilde{\Omega}$ is asymptotically independent of $\hat{\Pi}$ and $\hat{\Gamma}$.

2.5.2 Tests for Cointegration Rank

The effectiveness of the Johansen maximum likelihood procedure depends on our ability to correctly estimate the cointegration rank, r. A likelihood ratio test can be used to estimate the rank. To test:

$$H_0 : \text{rank}(\mathbf{\Pi}) = r_0 \qquad \text{against} \qquad H_1 : r_0 < \text{rank}(\mathbf{\Pi}) \leq r_1, \tag{2.12}$$

we compare the maximum likelihood functions for models with cointegration rank r_0 or r_1. The resulting LR test statistic is given by:

$$\begin{aligned}\lambda_{LR}(r_0, r_1) &= 2[\ln\mathcal{L}(r_1) - \ln\mathcal{L}(r_0)]\\ &= T\left[\sum_{i=1}^{r1}\ln(1-\lambda_i) + \sum_{i=1}^{r_0}\ln(1-\lambda_i)\right]\\ &= -T\sum_{i=r_0+1}^{r1}\ln(1-\lambda_i)\end{aligned}$$

The literature has focussed attention on two particular instances of this test:

$$H_0 : \text{rank}(\mathbf{\Pi}) = r_0 \qquad \text{against} \qquad H_1 : r_0 < \text{rank}(\mathbf{\Pi}) \leq K, \tag{2.13}$$

and

$$H_0 : \text{rank}(\mathbf{\Pi}) = r_0 \qquad \text{against} \qquad H_1 : \text{rank}(\mathbf{\Pi}) = r_0 + 1, \tag{2.14}$$

where the first, $\lambda_{LR}(r_0, K)$, is called the *trace statistic*, and the second, $\lambda_{LR}(r_0, r_0+1)$, is called the *maximum eigenvalue statistic*. The distributions of the corresponding likelihood-ratio test

statistics are non-standard and are tabulated, see [Johansen, 1995] or [MacKinnon et al., 1999]. In particular they are not χ^2-distributed, and depend on the number of common trends under H_0 or the alternative hypothesis. Johansen [Johansen, 1995] shows that these are distributed as:

$$\lambda_{LR}(r_0, K) \xrightarrow{d} \mathrm{Tr}(\mathcal{D}),$$

$$\text{and} \qquad \lambda_{LR}(r_0, r_0+1) \xrightarrow{d} \lambda_{\max}(\mathcal{D}),$$

where $\lambda_{\max}(\mathcal{D})$ denotes the maximum eigenvalue of the matrix $\mathcal{D}$, where $\mathcal{D}$ is given by:

$$\mathcal{D} = \left(\int_0^1 \mathbf{W} d\mathbf{W}'\right)' \left(\int_0^1 \mathbf{W}\mathbf{W}' ds\right)^{-1} \left(\int_0^1 \mathbf{W} d\mathbf{W}'\right), \tag{2.15}$$

and $\mathbf{W} := \mathbf{W}_{K-r_0(s)}$ is a standard $(K - r_0)$-dimensional Wiener process [Lütkepohl, 2007].

2.6 *Case Study:* Applying the Johansen Procedure

The Johansen procedure was run on 10,000 separate synthetic datasets generated according to each of the 5 VECM's specified in the paper by [Sugita, 2002], and listed in Table 2.2 below. The maximum eigenvalue test statistics was used to determine the cointegration rank for each synthetic dataset at the 5% significance level. The frequency of the rank as determined by the procedure is then counted and compared to the known rank.

Label	Sugita Model	Rank, r
$\mathbf{S}_0$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_t + \boldsymbol{\epsilon}_t$	0
$\mathbf{S}_1$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_0 + \begin{bmatrix} -0.2 \\ -0.2 \\ -0.2 \\ 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	1
$\mathbf{S}_2$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_0 + \begin{bmatrix} -0.2 & -0.2 \\ 0.2 & -0.2 \\ 0.2 & 0.2 \\ 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	2
$\mathbf{S}_3$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_0 + \begin{bmatrix} -0.2 & -0.2 & -0.2 \\ 0.2 & -0.2 & -0.2 \\ 0.2 & 0.2 & -0.2 \\ 0.2 & 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	3
$\mathbf{S}_4$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_0 + \begin{bmatrix} -0.2 & -0.2 & -0.2 & -0.2 \\ 0.2 & -0.2 & -0.2 & -0.2 \\ 0.2 & 0.2 & -0.2 & -0.2 \\ 0.2 & 0.2 & 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	4

Table 2.2: ECM models from the paper by [Sugita, 2002]

Model Number	Estimated Rank	Frequency
$\mathbf{S}_0$	**0**	**0.9315**
$\mathbf{S}_0$	1	0.0553
$\mathbf{S}_0$	2	0.0025
$\mathbf{S}_0$	3	0.0003
$\mathbf{S}_0$	4	0.0104
$\mathbf{S}_1$	0	0.0000
$\mathbf{S}_1$	**1**	**0.7628**
$\mathbf{S}_1$	2	0.1060
$\mathbf{S}_1$	3	0.0163
$\mathbf{S}_1$	4	0.1149
$\mathbf{S}_2$	0	0.0000
$\mathbf{S}_2$	1	0.0000
$\mathbf{S}_2$	**2**	**0.9232**
$\mathbf{S}_2$	3	0.0509
$\mathbf{S}_2$	4	0.0259
$\mathbf{S}_3$	0	0.0000
$\mathbf{S}_3$	1	0.0000
$\mathbf{S}_3$	2	0.0394
$\mathbf{S}_3$	**3**	**0.8989**
$\mathbf{S}_3$	4	0.0617
$\mathbf{S}_4$	0	0.0000
$\mathbf{S}_4$	1	0.0000
$\mathbf{S}_4$	2	0.0000
$\mathbf{S}_4$	3	0.0000
$\mathbf{S}_4$	**4**	**1.0000**

Table 2.3: Table showing the frequency with which each rank is determined by the maximum eigenvalue test for the Sugita VECM's. Each of the 10,000 datasets consisted of 150 observations from each of the 5 models. The most frequent estimated rank for each model is highlighted in bold-face type.

These results show that the Johansen ML method, and associated likelihood ratio tests, provide an effective methodology for determining the cointegration rank of a VECM system. For example the procedure correctly identified that model $\mathbf{S}_2$ has cointegration rank 2 for 92.3% of the synthetic datasets. It may be important to have a method for determining the cointegration rank; the samplers we develop in chapter 5 for the challenging task of joint estimation of static parameters and latent states in CVAR models operate conditional upon rank. By determining the rank using the ML procedure the appropriate model can be run to find the latent states and parameters.

Remark *In a complete Bayesian analysis however, we could introduce a prior over cointegration rank, r, and then evaluate the model evidence after running each of the alternative models with $r = 0, 1, \ldots, n$.*

Chapter 3

Bayesian Cointegrated Vector Autoregressive Models

3.1 Introduction

In this chapter the Cointegrated VAR (CVAR) models introduced in Chapter 2 are incorporated into a Bayesian modelling framework. The Bayesian analysis of CVAR models has been addressed in several papers, see [Koop et al., 2006] for an overview. The specification of the matrix variate model priors must be carried out with care to ensure that the posterior is not improper, see [Koop et al., 2006]. Here the aim is not to design novel prior specifications so we will adopt the Bayesian model of [Sugita, 2002] and [Geweke, 1996], which has desirable conjugacy properties. The resulting posterior distribution for an $n-$dimensional CVAR model is matrix-variate in dimension up to $3n^2 + n$ for full rank models [Peters et al., 2010]. Significant correlation exists between the blocks of matrix variate random variables, and for these reasons it is extremely difficult to design effective schemes for sampling the posterior distribution.

The focus of this research is the design and implementation of state-of-the-art stochastic sampling techniques which can attack such a high-dimensional and complex problem in the matrix-variate setting. In Chapter 5 suitable techniques are introduced, and for the remainder of this chapter the three main model categories are set out. The first model considered is an ECM CVAR model built on top of a *static* latent system; this has a fixed latent mean level with noise, and will be referred to as **Model $\mathbf{M}_0$** throughout.

The second model considered, **Model $\mathbf{M}_1$**, has a latent system with *dynamic* linear or non-linear trend terms. The final model has a stochastic driving process for the covariance matrix of the CVAR model observation error terms, and is labelled **Model $\mathbf{M}_2$**. The models are introduced in the following sections.

3.2 **Model $\mathbf{M}_0$**: Bayesian Static CVAR Model

Model $\mathbf{M}_0$ Initially consider the case of a CVAR model with an intercept, but no time trend or other dynamics; this amounts to having a stationary mean level $\boldsymbol{\mu}_0$ - with noise $\boldsymbol{w}_t$ - as the latent system process:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{w}_t \tag{3.1}$$

We assume that $\boldsymbol{x}_t$ is an integrated of order 1, $I(1)$, $(n \times 1)$-dimensional vector with r linear cointegrating relationships. The Error Correction Model (ECM) representation for the cointegrated series of vector observations at time t, $\boldsymbol{x}_t$, is given by:

$$(\mathbf{y}_t =) \quad \Delta \boldsymbol{x}_t = \boldsymbol{\mu}_t + \boldsymbol{\alpha}\boldsymbol{\beta}'\boldsymbol{x}_{t-1} + \sum_{i=1}^{p-1} \boldsymbol{\Psi}_i \Delta \boldsymbol{x}_{t-i} + \boldsymbol{\epsilon}_t, \tag{3.2}$$

where $t = p, p+1, \ldots, T$ and p is the number of lags [Sugita, 2002]. The observation errors are assumed to come from $\boldsymbol{\epsilon}_t \sim N(0, \Sigma)$ and are uncorrelated over time, and the system errors are from $\boldsymbol{w}_t \sim N(0, \Sigma_w)$; the matrix dimensions are:

Matrix	Dimension
$\boldsymbol{x}_t, \boldsymbol{\mu}_t, \boldsymbol{\epsilon}_t$	$(n \times 1)$
$\boldsymbol{\Psi}_i, \Sigma, \Sigma_w$	$(n \times n)$
$\boldsymbol{\alpha}, \boldsymbol{\beta}$	$(n \times r)$

3.2.1 The Multivariate Regression Format

We can now re-express the ECM model in a multivariate regression format:

$$Y = X\Gamma + Z\boldsymbol{\beta}\boldsymbol{\alpha}' + E \tag{3.3}$$

$$\overbrace{\begin{bmatrix} \Delta\boldsymbol{x}_p \\ \Delta\boldsymbol{x}_{p+1} \\ \vdots \\ \Delta\boldsymbol{x}_T \end{bmatrix}}^{Y}_{t\times n} = \overbrace{\begin{bmatrix} 1 & \Delta\boldsymbol{x}'_{p-1} & \ldots & \Delta\boldsymbol{x}'_1 \\ 1 & \Delta\boldsymbol{x}'_p & \ldots & \Delta\boldsymbol{x}'_2 \\ \vdots & \vdots & \ldots & \vdots \\ 1 & \Delta\boldsymbol{x}'_{T-1} & \ldots & \Delta\boldsymbol{x}'_{T-p+1} \end{bmatrix}}^{X}_{t\times a} \overbrace{\begin{bmatrix} \boldsymbol{\mu}_0 \\ \boldsymbol{\Psi}_1 \\ \vdots \\ \boldsymbol{\Psi}_{p-1} \end{bmatrix}}^{\Gamma}_{a\times n} + \overbrace{\begin{bmatrix} \boldsymbol{x}_{p-1} \\ \boldsymbol{x}_p \\ \vdots \\ \boldsymbol{x}_{T-1} \end{bmatrix}}^{Z}_{t\times n} \overbrace{\begin{bmatrix} \boldsymbol{\beta}_1 \\ \vdots \\ \boldsymbol{\beta}_r \end{bmatrix}'}^{\boldsymbol{\beta}}_{n\times r} \overbrace{\begin{bmatrix} \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{\alpha}_r \end{bmatrix}}^{\boldsymbol{\alpha}'}_{r\times n} + \overbrace{\begin{bmatrix} \boldsymbol{\epsilon}_p \\ \boldsymbol{\epsilon}_{p+1} \\ \vdots \\ \boldsymbol{\epsilon}_T \end{bmatrix}}^{E}_{t\times n},$$

where t is the number of rows of Y, hence $t = T-p+1$ giving X dimension $t \times (1+n(p-1)) = t \times a$, Γ with dimension $a \times n$. This can be re-expressed in the following very compact form:

$$Y = WB + E \tag{3.4}$$

$$\overbrace{\begin{bmatrix} \Delta\boldsymbol{x}_p \\ \Delta\boldsymbol{x}_{p+1} \\ \vdots \\ \Delta\boldsymbol{x}_T \end{bmatrix}}^{Y}_{t\times n} = \overbrace{\begin{bmatrix} 1 & \Delta\boldsymbol{x}'_{p-1} & \ldots & \Delta\boldsymbol{x}'_1 & \boldsymbol{x}_{p-1}\boldsymbol{\beta}_1 & \ldots & \boldsymbol{x}_{p-1}\boldsymbol{\beta}_r \\ 1 & \Delta\boldsymbol{x}'_p & \ldots & \Delta\boldsymbol{x}'_2 & \boldsymbol{x}_p\boldsymbol{\beta}_1 & \ldots & \boldsymbol{x}_p\boldsymbol{\beta}_r \\ \vdots & \vdots & \ldots & \vdots & \vdots & \ldots & \vdots \\ 1 & \Delta\boldsymbol{x}'_{T-1} & \ldots & \Delta\boldsymbol{x}'_{T-p+1} & \boldsymbol{x}_p\boldsymbol{\beta}_1 & \ldots & \boldsymbol{x}_{T-1}\boldsymbol{\beta}_r \end{bmatrix}}^{W}_{\{t\times(a+r)\}=t\times k} \overbrace{\begin{bmatrix} \boldsymbol{\mu}_0 \\ \boldsymbol{\Psi}_1 \\ \vdots \\ \boldsymbol{\Psi}_{p-1} \\ \boldsymbol{\alpha}_1 \\ \vdots \\ \boldsymbol{\alpha}_r \end{bmatrix}}^{B}_{k\times n} + \overbrace{\begin{bmatrix} \boldsymbol{\epsilon}_p \\ \boldsymbol{\epsilon}_{p+1} \\ \vdots \\ \boldsymbol{\epsilon}_T \end{bmatrix}}^{E}_{t\times n},$$

where I have highlighted in cyan the submatrices of W and B which together specify the long-run multiplier matrix ($\boldsymbol{\Pi} = \boldsymbol{\alpha}\boldsymbol{\beta}'$). The dimensions of W and B are as shown in equation 3.4. It will be useful to also define $\hat{B} = (W'W)^{-1}W'Y$ to be the OLS estimate of the matrix B, and $\hat{S} = (Y - W\hat{B})'(Y - W\hat{B})$ to be the residual sum of squares from the OLS fit.

Priors Now we consider conjugate priors for some of the parameters in equation 3.4, and follow the suggestions given in the paper by Sugita [Sugita, 2002], and used in Peters [Peters et al., 2010]. It is popular to consider hierarchical prior models for matrix-variate parameter Σ (when not evolving dynamically) and B given as:

- The prior for the ECM noise covariance Σ can be given as an Inverse Wishart distribution with h degrees of freedom and where S is an $(n \times n)$ positive definite matrix:

$$\Sigma \sim IW(S, h) \tag{3.5}$$
$$\pi(\Sigma) \sim |S|^{h/2}|\Sigma|^{-(h+n+1)/2} \exp\left[-\frac{1}{2}\operatorname{Tr}\{\Sigma^{-1}S\}\right]$$

- The conjugate prior density for B conditional on the noise covariance Σ follows a matrix-variate Gaussian distribution with covariance matrix $\Sigma \otimes A^{-1}$ of the form:

$$\begin{aligned} B|\Sigma &\sim N_{k,n}(B|\overline{B}, A^{-1}, \Sigma) \\ \pi(B|\Sigma) &\sim |\Sigma|^{-k/2}|A|^{n/2} \exp\left[-\frac{1}{2}\operatorname{Tr}\left\{\Sigma^{-1}(B-\overline{B})'A(B-\overline{B})\right\}\right], \end{aligned} \tag{3.6}$$

  with hyperparameter PSD matrices A $(k \times k)$ and mean PSD matrix $\overline{B}$ $(k \times n)$,

- The prior for the matrix of cointegration vectors $\boldsymbol{\beta}$ can be written as a matrix-variate Gaussian

$$\begin{aligned} \boldsymbol{\beta} &\sim N_{n,r}(\boldsymbol{\beta}|\overline{\boldsymbol{\beta}}, H^{-1}, Q) \\ \pi(\boldsymbol{\beta}) &\sim |Q|^{-n/2}|H|^{r/2} \exp\left[-\frac{1}{2}\operatorname{Tr}\left\{Q^{-1}(\boldsymbol{\beta}-\overline{\boldsymbol{\beta}})'H(\boldsymbol{\beta}-\overline{\boldsymbol{\beta}})\right\}\right], \end{aligned} \tag{3.7}$$

  where $\overline{\boldsymbol{\beta}}$ is the prior mean of $\boldsymbol{\beta}$, Q is an $(r \times r)$ positive definite matrix and H is an $(n \times n)$ PSD matrix.

If we assume as in [Sugita, 2002] that $\boldsymbol{\beta}$, Σ and Σ_w are mutually independent then the joint prior of the parameters of interest is:

$$\begin{aligned} p(\boldsymbol{\beta}, B, \Sigma) &= p(\boldsymbol{\beta})p(B|\Sigma)p(\Sigma) \\ &\propto \pi(\boldsymbol{\beta})|A|^{n/2}|\Sigma|^{-\frac{k+h+n+1}{2}} \exp\left[-\frac{1}{2}\operatorname{Tr}\left\{\Sigma^{-1}[S+(B-\overline{B})'A(B-\overline{B})]\right\}\right] \end{aligned} \tag{3.8}$$

Note: When N is followed with sub-scripts e.g. $Y \sim N_{n,T}(\mu, \Sigma, \Omega)$ it refers to the matrix-variate normal distribution with row dependence captured in the $(n \times n)$ covariance matrix Σ and column dependence captured by a $(T \times T)$ matrix Ω.

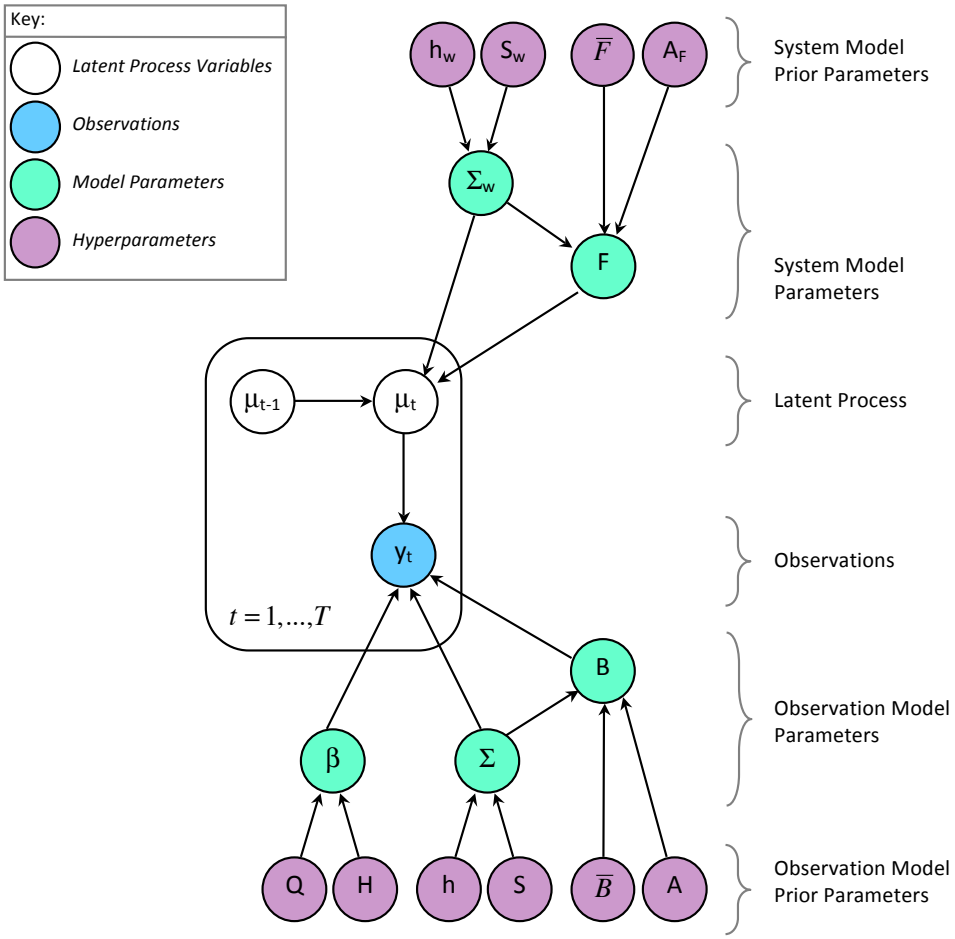


Figure 3.1: Directed Acyclic Graph (DAG) showing the full Bayesian hierarchical ECM considered here. The priors for the latent system are discussed in the next section; the parameters $\overline{F}, A_F, h_w$, and S_w are hyperparameters of the latent system model.

Likelihood According to [Gupta and Nagar, 1999] for an $(n \times T)$ random matrix-variate Gaussian $Y' \sim N_{n,T}(\mu, \Sigma, \Omega)$ then the vectorised form, in which the columns are stacked on top of each other to make a $nT \times 1$ random vector, is multivariate Gaussian and denoted:

$$\text{Vec}(Y) \sim N_{nT}\left(\text{Vec}(\mu), \Sigma \otimes \Omega\right). \tag{3.9}$$

Therefore we can represent the matrix-variate likelihood for this regression, for the parameters of interest B, Σ and $\boldsymbol{\beta}$ by:

$$\mathcal{L}(Y|\boldsymbol{\beta}, B, \Sigma) = (2\pi)^{-\frac{nt}{2}} |\Sigma \otimes I_t|^{-\frac{1}{2}} \exp\left[-\frac{1}{2}\text{Vec}(Y - WB)'(\Sigma^{-1} \otimes \Omega^{-1})\,\text{Vec}(Y - WB)\right] \tag{3.10}$$

Posteriors The conjugate prior distributions allow us to easily write down the posterior distributions for the observation system. We condition upon the observations $Y = [\Delta \boldsymbol{x}_p, \ \ldots, \ \Delta \boldsymbol{x}_T]'$. For derivation details see [Sugita, 2002].

- The conditional posterior of Σ is derived as an Inverted Wishart distribution with parameters $S^\star$ and $h^\star$:

$$\begin{aligned} \Sigma &\sim IW(S^\star, h^\star) \\ p(\Sigma|\beta, Y) &\propto |S^\star|^{(T+h)/2} |\Sigma|^{-(T+h+n+1)/2} \exp\left[-\frac{1}{2}\text{Tr}\{\Sigma^{-1} S^\star\}\right], \end{aligned} \tag{3.11}$$

where we define

$$\begin{aligned} S_\star &= S + \hat{S} + (\overline{B} - \hat{B})'[A^{-1} + (W'W)^{-1}]^{-1}(\overline{B} - \hat{B}) \\ h_\star &= t + h \end{aligned}$$

- The conditional posterior of B is a matrix-variate normal distribution with covariance $\Sigma \otimes A_\star^{-1}$:

$$\begin{aligned} B &\sim N_{k,n}(B_\star, A_\star^{-1}, \Sigma) \\ p(B|\beta, \Sigma, Y) &\propto |A_\star|^{n/2}|\Sigma|^{-k/2} \exp\left[\frac{1}{2}\,\mathrm{Tr}\left\{\Sigma^{-1}(B - B_\star)'A_\star(B - B_\star)\right\}\right], \end{aligned} \tag{3.12}$$

or alternatively if Σ is integrated out B follows a matrix-variate student distribution:

$$\begin{aligned} B &\sim MST_{k,n}(B_\star, A_\star^{-1}, S_\star) \\ p(B|\beta, Y) &\propto |S_\star|^{t/2}|A_\star|^{n/2}\ |S_\star + (B - B_\star)'A_\star(B - B_\star)|^{-(t+k)/2}, \end{aligned} \tag{3.13}$$

where we have defined

$$\begin{aligned} A_\star &= A + W'W, \\ B_\star &= A_\star^{-1}(A\overline{B} + W'W\hat{B}), \end{aligned} \tag{3.14}$$

- By integrating the joint posterior distribution with respect to B the posterior distribution for the cointegrating vector, β, is found to be

$$p(\beta|Y) \propto \pi(\beta)\ |S_\star|^{-(t+h+1)/2}|A_\star|^{-n/2} \tag{3.15}$$

The log-posterior for $\boldsymbol{\beta}$ therefore is:

$$\log p(\boldsymbol{\beta}|Y) \propto -\frac{(t+h+1)}{2}\log|S_\star| - \frac{n}{2}\log|A_\star|, \tag{3.16}$$

and this is plotted in Figure 3.2 for data-sets of varying size.

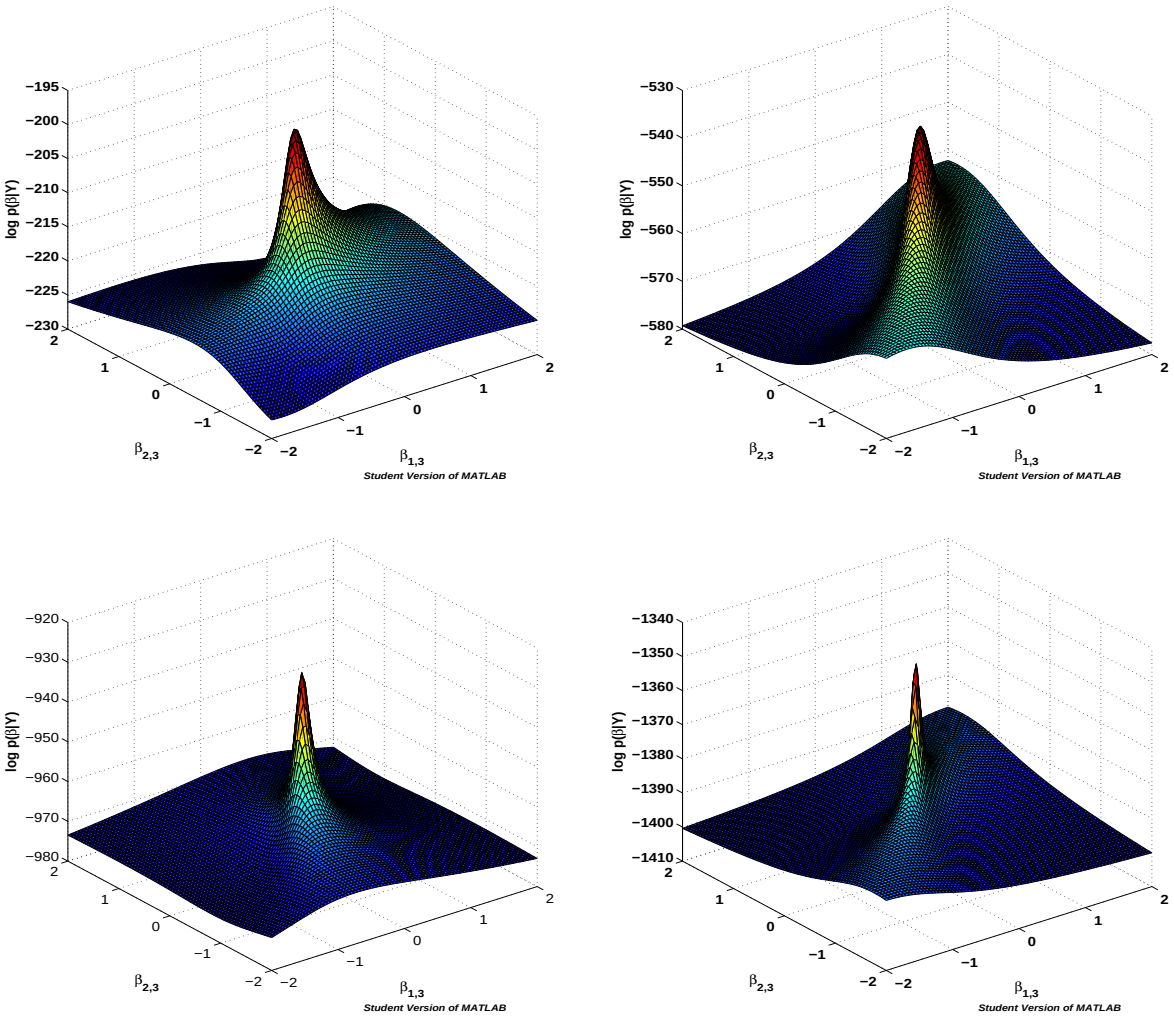


Figure 3.2: Plots showing the effect of altering the data-set size, $T = 50,\ 100,\ 150,\ 200$ on $\log p(\boldsymbol{\beta}|Y)$.

It is trivial to sample from the posteriors for Σ and B given the observations, Y, however sampling from β requires the use of a sophisticated stochastic sampling methodology: we will introduce Adaptive Markov Chain Monte Carlo (AdMCMC) sampling for this purpose in section 5.3.

Remark *Aside from the fact that it is impossible to sample from the posterior distribution of* $\log p(\beta|Y)$ *exactly in closed form the surface is difficult to sample from using standard stochastic techniques. This is both because the posterior surface possesses a very sharp ridge running along certain directions, and also because the shape is very sensitive to small changes in the matrix variate components on which it depends.*

3.3 **Model $\mathbf{M}_1$**: Latent Linear Dynamics CVAR Model

We now increase the flexibility of the model by allowing the latent system to possess a linear latent dynamic in mean. A simple version of this model may be generated via the autoregressive process with transition matrix, F_1 :

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{F}_1\boldsymbol{\mu}_{t-1} + \boldsymbol{w}_t. \tag{3.17}$$

Again we may re-express this in a multivariate regression format by stacking the latent state vectors:

$$M = XF + E \tag{3.18}$$

$$\overbrace{\underbrace{\begin{bmatrix}\boldsymbol{\mu}_1\\ \boldsymbol{\mu}_2\\ \vdots\\ \boldsymbol{\mu}_T\end{bmatrix}}_{T\times n}}^{M} = \overbrace{\underbrace{\begin{bmatrix}1 & \mathbf{0}\\ 1 & \boldsymbol{\mu}_1'\\ \vdots & \vdots\\ 1 & \boldsymbol{\mu}_{T-1}'\end{bmatrix}}_{T\times a}}^{X} \overbrace{\underbrace{\begin{bmatrix}\boldsymbol{\mu}_0\\ \boldsymbol{F}_1\end{bmatrix}}_{a\times n}}^{F} + \overbrace{\underbrace{\begin{bmatrix}\boldsymbol{w}_1\\ \boldsymbol{w}_2\\ \vdots\\ \boldsymbol{w}_T\end{bmatrix}}_{T\times n}}^{E},$$

where T is the number of observations and thus rows of M, giving X dimension $T\times(1+n)) = t\times a$, and F which is defined as $F = [\boldsymbol{\mu}_0 \;\; \mathbf{F}_1]'$, dimension $a \times n$. The OLS estimator of F is given by $\hat{F} = (X'X)^{-1}X'M$, and the residual sum of squares is $\hat{S}_w = (M - X\hat{F})'(M - X\hat{F})$.

System Model Priors Now in a similar way we consider conjugate priors for the latent system $\boldsymbol{\mu}_{1:T}$, due to their attractive properties. It is possible to specify hierarchical conjugate priors [Koop and Korobilis, 2010] for the latent state transition matrix and the latent state noise covariance. Referring to Figure 3.1 the latent system can be seen to have a similar model structure to that of the observation system, particularly the conjugate relationships for $F|\Sigma_w$ and $B|\Sigma$. We specify:

- The prior for the latent state noise covariance may be given as an inverted Wishart distribution, with parameter, S_w, an $(n \times n)$ hyperparameter PSD matrix, and h_w degrees of freedom

$$\begin{aligned}\Sigma_w &\sim IW(S_w, h_w)\\ \pi(\Sigma_w) &\sim |S_w|^{h_w/2}|\Sigma_w|^{-(h_w+n+1)/2}\exp\left[-\frac{1}{2}\operatorname{Tr}\{\Sigma_w^{-1}S_w\}\right]\end{aligned} \tag{3.19}$$

- The conjugate prior density for F conditional on the latent state noise covariance Σ_w follows a matrix-variate Gaussian distribution, with covariance matrix $\Sigma_w \otimes A_F^{-1}$ of the form:

$$\begin{aligned}F|\Sigma_w &\sim N_{a,n}(F|\overline{F}, A_F^{-1}, \Sigma_w)\\ \pi(F|\Sigma_w) &\sim |\Sigma_w|^{-k/2}|A_F|^{n/2}\exp\left[-\frac{1}{2}\operatorname{Tr}\left\{\Sigma_w^{-1}(F-\overline{F})'A_F(F-\overline{F})\right\}\right],\end{aligned} \tag{3.20}$$

  where $\overline{F}$ is a $(a \times n)$ mean matrix, $a = n + 1$ and A_F is an $(a \times a)$ hyperparameter PSD matrix.

Latent states Likelihood We can write down the likelihood of F and Σ_w, where the latent states are denoted $M = [\boldsymbol{\mu}_1 \;\; \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_T]'$:

$$\mathcal{L}(M|F, \Sigma_w) \propto |\Sigma_w|^{-T/2}\exp\left[-\frac{1}{2}\left\{\Sigma_w^{-1}\left[\hat{S}_w + (F-\overline{F})'X'X(F-\overline{F})\right]\right\}\right] \tag{3.21}$$

System Model Posterior Analysis Conditional upon knowledge of the latent system states $M = [\boldsymbol{\mu}_1 \;\; \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_T]'$ we may evaluate the posterior distributions of $p(\Sigma_w|M)$ and $p(F|\Sigma_w, M)$ under the hierarchical framework specified above. The posterior distributions are available exactly since we have chosen conjugate priors.

- The latent system noise covariance, Σ_w, has posterior distributed as an Inverse Wishart with

$$\begin{aligned}\Sigma_w &\sim IW(S_w^\star, h_w^\star)\\ p(\Sigma_w|M) &\propto |S_w^\star|^{(T+h_w)/2}|\Sigma_w|^{-(T+h_w+n+1)/2}\exp\left[-\frac{1}{2}\operatorname{Tr}\{\Sigma_w^{-1}S_w^\star\}\right],\end{aligned} \tag{3.22}$$

where we have defined (analogously to the observations system):

$$S_w^\star = S_w + \hat{S}_w + \hat{F}'X'X\hat{F} + \overline{F}'A_F\overline{F} - F_\star'[A_F^\star]F_\star$$
$$h_w^\star = T + h_w$$

- The latent system transition matrix, F, has a matrix-variate normal distribution, conditional upon the noise matrix Σ_w

$$F \sim N_{a,n}\left(F|\overline{F}, (A_F^\star)^{-1}, \Sigma_w\right)$$
$$p(F|\beta, \Sigma, Y) \propto |A_F^\star|^{n/2}|\Sigma_w|^{-a/2}\exp\left[\frac{1}{2}\,\mathrm{Tr}\left\{\Sigma_w^{-1}(F-F_\star)'A_F^\star(F-F_\star)\right\}\right], \tag{3.23}$$

and, again, by integrating out Σ_w, we can express the conditional posterior for F as a matrix-variate student form:

$$F \sim MST_{a,n}(F_\star, (A_F^\star)^{-1}, S_w^\star)$$
$$p(F|M) \propto |S_w^\star|^{T/2}|A_F^\star|^{n/2}\ |S_w^\star + (F-F_\star)'A_F^\star(F-F_\star)|^{-(T+a)/2}, \tag{3.24}$$

where we have used

$$A_F^\star = A_F + X'X$$
$$F^\star = (A_F^\star)^{-1}(A_F\overline{F} + X'X\hat{F}) \tag{3.25}$$

3.4 **Model $\mathbf{M}_2$**: Latent Covariance Structure CVAR Model

The third model that we consider involves extending the basic static model to incorporate a dynamic latent covariance structure for the ECM observation noise process. The phenomenon of stochastic volatility observed in numerous real-world financial time series is termed *heteroskedasticity*, and we would like to be able to account for its presence in the observed price series error distribution. Various model classes for modelling the multivariate dynamic dependence structure of the errors has been proposed in the literature. One possibility is based around parametric Wishart process models and is the approach we explore briefly here. This approach will focus on incorporating dynamically evolving heteroskedasticity in the innovation process of the CVAR error time series.

The matrix Wishart process as presented in [Bru, 1991] and developed further in [Cuciero et al., 2011] and [Fox and West, 2011], denoted by $W(s, n, \Phi_0)$, of dimension n with index d and initial state Φ_0, defined by the square $\Phi_t = N_t N_t'$ of a $d \times n$ Brownian matrix N_t, has matrix Φ_t that satisfies the s.d.e

$$d\Phi_t = \sqrt{\Phi_t}d\mathbf{W}_t + d\mathbf{W}'_t\sqrt{\Phi_t} + s\mathbb{I}dt, \tag{3.26}$$

where $\mathbf{W}$ is an $n \times n$ Brownian matrix and $\mathbb{I}$ is an identity matrix. The matrix Wishart process is guaranteed to remain on the space of symmetric positive definite matrices as shown in [Cuciero et al., 2011] and [Fox and West, 2011]. Another benefit of utilising the processes of the matrix Wishart form is that it is possible to write an exact representation for the transition probabilities for the matrix states which can be derived in closed form. Conditionally they are defined as:

$$\Phi_t^{-1}\big|\,\nu_t, S_{t-1} \sim \mathcal{W}(S_{t-1}, \nu)$$

$$S_{t-1}\,|A, \Phi_{t-1} = \frac{1}{\nu}\left(A^{1/2}\right)\left(\Phi_{t-1}^{-1}\right)\left(A^{1/2}\right)',$$

where the scattering matrix, S_t, defines the time-variation of the covariance structure, and the parameter matrix A is chosen to be postivie definite. For simplicity here we focus on the simplest member of the family of Wishart diffusion processes: the univariate square Bessel process.

3.4.1 The Square Bessel Process

The univariate square Bessel process is defined by the closed form transition density given by:

$$q_t(x,y) = \frac{1}{2t}\left(\frac{y}{x}\right)^{\nu/2} \exp\left(\frac{x+y}{2t}\right) I_\nu\left(\frac{\sqrt{xy}}{t}\right), \tag{3.27}$$

where I_ν is the modified Bessel function of the first kind with order ν, which for any real ν is given by

$$I_\nu(z) = \left(\frac{1}{2}z\right)^\nu \sum_{k=0}^{\infty} \frac{(\frac{1}{4}z^2)^k}{k!\Gamma(\nu+k+1)}$$

We consider a simple example stochastic volatility CVAR model where the covariance matrix is sparse, $\Sigma_t = \sigma_t \mathbb{I}$, with the stochastic volatility process σ_t given by the square Bessel process as defined by the transition densities in equation (3.27). The innovation error process, $\boldsymbol{w}_t$, is a multivariate Gaussian random vector which evolves according to the square Bessel process:

$$\begin{aligned} \boldsymbol{w}_t &\sim \mathcal{N}(0,\Sigma_t), \quad \text{where} \quad \Sigma_t = \sigma_t \mathbb{I}_n \\ \sigma_t|\sigma_{t-1} &\sim q_t(\sigma_{t-1},\sigma_t) = \frac{1}{2t}\left(\frac{\sigma_t}{\sigma_{t-1}}\right)^{-\nu/2} \exp\left(\frac{\sigma_{t-1}+\sigma_t}{2t}\right) I_\nu\left(\frac{\sqrt{\sigma_{t-1}\sigma_t}}{t}\right) \end{aligned} \tag{3.28}$$

3.4.2 Sampling from the Square Bessel Process

In [Makarov and Glew, 2009] a simple sequential sampling algorithm is given for simulating a path from a square Bessel diffusion process. First a sample Y_n is drawn from a Poisson distribution mean $X_{n-1}/2\Delta t$, then conditional on this random draw X_n is sampled from a gamma distribution. The algorithm returns samples evenly spaced in time.

Algorithm 1: Square Bessel Process Sequential Sampling Algorithm

Input: $X_0 > 0$, $0 = t_0 < t_1 < \cdots < t_N$, $\mu > -1$
for n *from* 1 *to* N **do**

1. Sample $Y_n \sim \mathcal{P}o\left(\frac{X_{n-1}}{2(t_n - t_{n-1})}\right)$
2. Sample $X_n \sim \mathcal{G}\left(Y_n + \mu + 1, \frac{1}{2(t_n - t_{n-1})}\right)$

end
Output: $(X_0, X_1, \ldots, X_N)$

The algorithm is based on sampling from a randomised gamma distribution of the form $G(\alpha+Y,\beta)$ where $\alpha + Y > 0$ and $\beta > 0$ are scale and rate parameters, respectively, and Y is a non-negative integer-valued random variable. This algorithm assumes that $\nu = 2$; in the general situation when $\nu \neq 2$ [Makarov and Glew, 2009] suggest proceeding as follows:

- For given λ_0, ν and X_0 sample a path from the square Bessel process with $\mu = 2\lambda_0/\nu^2 - 1$ that starts at $(2/\nu)^2 X_0$ using the algorithm above.
- After obtaining the path, rescale its values by multiplying them by $(\nu/2)^2$

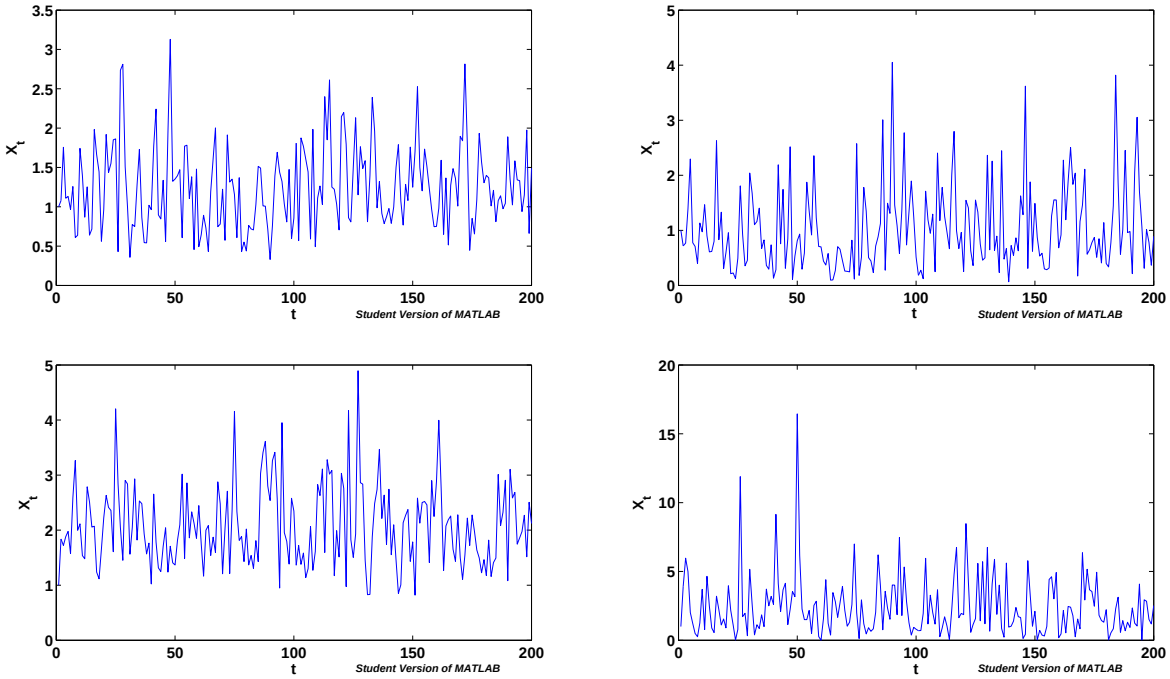


Figure 3.3: Paths drawn from the square Bessel process with parameters (clockwise from top left) $\{\lambda_0 = 1, \nu = 1.25\}$, $\{\lambda_0 = 1, \nu = 2\}$, $\{\lambda_0 = 1, \nu = 3.5\}$ and $\{\lambda_0 = 2, \nu = 2\}$ for 200 time steps starting at $X_0 = 1$ using Algorithm 1.

Remark *Currently we do not attempt the difficult task of attempting to estimate the value of ν, If we did wish to attempt this estimation we would need to use a very strict prior to constrain its value, since otherwise the evaluation of the modified Bessel function in the simulation can cause the square Bessel process to 'blow-up' numerically. Thus, building a hyper-prior structure for ν is a challenge in practice. Instead, for the moment, we assume that ν is known.*

Chapter 4

State-Space Models and Linear Filtering

4.1 Introduction

State-space models have been studied and used extensively within the physical sciences, engineering and systems theory. The general format of these models is that an observed (vector[1]) time series $\mathbf{y}_{1:T} = \mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_T$ depends upon an unobserved or latent underlying state $\mathbf{z}_{1:T} = \mathbf{z}_1, \mathbf{z}_2, \ldots, \mathbf{z}_T$ which is driven by a stochastic process.

The state vector evolves according to

$$\mathbf{z}_t = \mathbf{F}_{t-1}\mathbf{z}_{t-1} + \boldsymbol{w}_{t-1}, \qquad t = 1, 2, \ldots \tag{4.1}$$

where $\mathbf{F}_t$ is a matrix which may, in general, depend upon the time, t, and $\boldsymbol{w}_t$ is an error process. This is often called the *transition equation*, and $\mathbf{F}_t$ the *transition matrix*, because together they describe the transition of the latent state between times t and $t+1$.

The relationship between $\mathbf{y}_t$ and $\mathbf{z}_t$ is governed by the *observation* or *measurement* equation

$$\mathbf{y}_t = \mathbf{H}_t\mathbf{z}_t + \boldsymbol{v}_t, \qquad t = 1, 2, \ldots \tag{4.2}$$

where $\mathbf{H}_t$ is the *measurement matrix*, which may in general depend upon the time period t, and $\boldsymbol{v}_t$ is usually assumed to be a noise process. We may assume:

$\mathbf{z}_t$:	is an $(n \times 1)$ vector of unobserved or latent state variables,
$\mathbf{y}_t$:	is an $(n \times 1)$ vector of observable output variables,
$\boldsymbol{w}_t$:	is an $(n \times 1)$ vector of transition equation errors or noise,
$\boldsymbol{v}_t$:	is an $(n \times 1)$ vector of observation/measurement errors or noise,
$\mathbf{F}_t$:	is an $(n \times n)$ transition matrix,
$\mathbf{H}_t$:	is an $(n \times n)$ measurement matrix.

In general these matrices are allowed to be time varying, but we will often be able to assume that one or more of them are time invariant. We also assume that the process generating the system states $\mathbf{z}_t$, and thus the observed states $\mathbf{y}_t$, starts from an initial state $\mathbf{z}_0$.

The general specification of this state-space model is completed by making assumptions about the noise processes [Lütkepohl, 2007]. The joint process

$$\begin{bmatrix} \boldsymbol{w}_t \\ \boldsymbol{v}_t \end{bmatrix},$$

is a zero mean, serially uncorrelated noise process with possibly time varying covariance matrices:

$$\begin{bmatrix} \Sigma_{\boldsymbol{w}_t} & \Sigma_{\boldsymbol{w}_t\boldsymbol{v}_t} \\ \Sigma_{\boldsymbol{v}_t\boldsymbol{w}_t} & \Sigma_{\boldsymbol{v}_t} \end{bmatrix}.$$

[1] Here for example: $\mathbf{y}_t = \begin{bmatrix} y_{1,t} & y_{2,t} & \ldots & y_{d,t} \end{bmatrix}'$.

The DAG for this state-space model is given below in Figure 4.1; note the latent state process has first order Markov dynamics. An identical DAG also corresponds to the large class of discrete state Hidden Markov Models (HMM's) and continuous state Linear and Non-Linear Dynamical Systems (to which our models belong). More general versions of the SSM are possible (including for example exogenous variables), but the current model will be sufficient for our purposes.

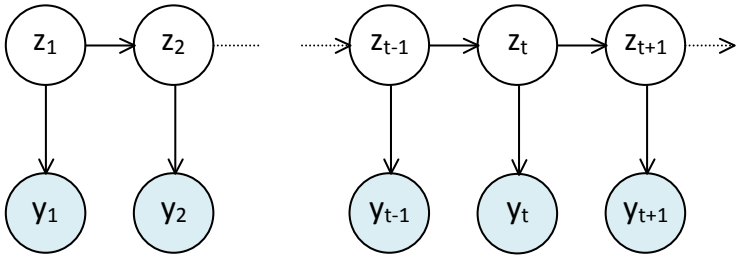


Figure 4.1: DAG for the state-space model with first order Markov latent dynamics.

With the stochastic assumptions mentioned above the means and covariance matrices of the output process can be derived:

$$\mu_{\mathbf{y}_t} = \mathbb{E}(\mathbf{y}_t) = \mathbf{H}_t \mathbb{E}(\mathbf{z}_t) \tag{4.3}$$

$$\mathrm{Cov}(\mathbf{y}_t, \mathbf{y}_{t+h}) = \mathbf{H}_t \mathrm{Cov}(\mathbf{z}_t, \mathbf{z}_{t+h}) \mathbf{H}'_t.$$

In general the means and autocovariances of the $\mathbf{y}_t$'s are not time invariant, therefore in general $\mathbf{y}_t$ is a nonstationary process [Lütkepohl, 2007].

4.2 The Filtering Problem

For a moment let us consider a more general non-linear discrete-time state-space model:

$$\begin{aligned} \mathbf{z}_t &= f_t(\mathbf{z}_{t-1}, \boldsymbol{w}_{t-1}) \\ \mathbf{y}_t &= h_t(\mathbf{z}_t, \boldsymbol{v}_t), \end{aligned}$$

where the variables are as defined previously, except for f_t and h_t which are now permitted to be non-linear, time-variant, deterministic functions.

The general objective of filtering is to estimate the set of system states $\{\mathbf{z}_t, t \in \mathbb{N}\}$ based on the set of observations from the measurement equation $\{\mathbf{y}_i, i = 1, \ldots, t\}$ up to time t. In the Bayesian viewpoint the aim of filtering is to calculate the posterior distribution $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, which naturally captures the degree of belief in the range of values that state $\mathbf{z}_t$ may take, conditional on the observations $\mathbf{y}_{1:t}$. Given the initial, or prior, distribution of the state vector, $p(\mathbf{z}_t)$, the posterior distribution can be obtained recursively in a two stages process: *prediction* and *update*.

Prediction Stage

The result of the prediction stage is a calculation of an *a priori* estimate of the posterior distribution, $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, at time t. This *a priori* estimate is based on the *a posteriori* estimate from the previous time step, and utilizes the identity $p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_{1:t-1}) = p(\mathbf{z}_t|\mathbf{z}_{t-1})$ in the Chapman-Kolmogorov equation, giving:

$$
\begin{aligned}
p(\mathbf{z}_t|\mathbf{y}_{1:t-1}) &= \int p(\mathbf{z}_t, \mathbf{z}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1} \\
&= \int p(\mathbf{z}_t|\mathbf{z}_{t-1}, \mathbf{y}_{1:t-1}) p(\mathbf{z}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1} \\
&= \int p(\mathbf{z}_t|\mathbf{z}_{t-1}) p(\mathbf{z}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{z}_{t-1}. \qquad (4.4)
\end{aligned}
$$

Update Stage

In the update stage an *a posteriori* estimate of the posterior distribution at time t is produced, given a new measurement $\mathbf{y}_t$. The *a posteriori* estimate is calculated using the following equations:

$$
\begin{aligned}
p(\mathbf{z}_t|\mathbf{y}_{1:t}) &= \frac{p(\mathbf{y}_{1:t}|\mathbf{z}_t) p(\mathbf{z}_t)}{p(\mathbf{y}_{1:t})} \\
&= \frac{p(\mathbf{y}_t, \mathbf{y}_{1:t-1}|\mathbf{z}_t) p(\mathbf{z}_t)}{p(\mathbf{y}_t, \mathbf{y}_{1:t-1})} \\
&= \frac{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \mathbf{z}_t) p(\mathbf{y}_{1:t-1}|\mathbf{z}_t) p(\mathbf{z}_t)}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) p(\mathbf{y}_{1:t-1})} \\
&= \frac{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}, \mathbf{z}_t) p(\mathbf{z}_t|\mathbf{y}_{1:t-1}) p(\mathbf{y}_{1:t-1}) p(\mathbf{z}_t)}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) p(\mathbf{y}_{1:t-1}) p(\mathbf{z}_t)} \\
&= \frac{p(\mathbf{y}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})}, \qquad (4.5)
\end{aligned}
$$

and the denominator is given by:

$$
p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{y}_t|\mathbf{z}_t) p(\mathbf{z}_t|\mathbf{y}_{1:t-1}) d\mathbf{z}_t. \qquad (4.6)
$$

During the update stage, the measurement, $\mathbf{y}_t$, is used to modify the prior density to obtain the posterior density of the current state. The recursive structure of the prediction/update scheme (sometimes also called predictor/corrector) is clear from these equations. For general models this optimal solution is intractable due to the potentially high-dimensional, non-analytic integrals in the prediction and update equations.

By imposing linearity restrictions on the model it is possible to perform these integrals exactly, and the continuous variable case is termed the *linear dynamical system*. In conjunction with Gaussian assumptions for the transition and observation error processes the system can be optimally estimated using the *Kalman Filter*, which is described in the next section.

4.3 The Kalman Filter

The *Kalman filter* is a recursive tool to estimate the latent states $\mathbf{z}_t$ given observations of the outputs $\mathbf{y}_{1:t}$, originally developed by Kalman [Kalman, 1960], and Kalman and Bucy. Under normality assumptions the Kalman filter provides the conditional expectation $\mathbb{E}(\mathbf{z}_t|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$ and the conditional covariance matrix $\mathrm{Cov}(\mathbf{z}_t|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$, where the latter may be used as a measure of prediction uncertainty. The computation of the estimators $\mathbb{E}(\mathbf{z}_t|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$ is called *filtering*, to distinguish it from *smoothing* $\mathbb{E}(\mathbf{z}_t|\mathbf{y}_{1:T}, \mathbf{y}_2, \ldots, \mathbf{y}_T)$ for $t = 1, \ldots, T$, and *prediction* $\mathbb{E}(\mathbf{z}_{t+1}|\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_t)$.

The Kalman filter is very widely applied in science and technology: particularly for the guidance, navigation and control of vehicles and weapons systems. It has also found a wide variety of applications in the fields of signal processing and econometrics, where, rather than filtering the likely location of an object based on noisy measurements of position, it is used to filter the likely underlying states based on noisy observations of outputs or econometric variables. The Kalman filter recursions are now presented.

4.3.1 The Kalman Filter Recursions

We assume a state-space model as defined by a slightly modified version of equations (4.1) and (4.2):

$$\begin{aligned} \mathbf{z}_t &= \mathbf{F}\mathbf{z}_{t-1} + \mathbf{J}\mathbf{x}_{t-1} + \boldsymbol{w}_{t-1}, && \textit{Transition Equation} \\ \mathbf{y}_t &= \mathbf{H}_t\mathbf{z}_t + \mathbf{G}\mathbf{x}_t + \boldsymbol{v}_t, && \textit{Observation Equation} \end{aligned}$$

for times $t = 1, 2, \ldots$. The transition matrix, $\mathbf{F}$, is assumed to be time invariant and known. The measurement matrix, $\mathbf{H}_t$, is assumed to be known and non-stochastic at time t, and may include lagged output variables (since these are available at time t). The greyed-out matrices, $\mathbf{J}$ and $\mathbf{G}$, and variables $\mathbf{x}_t$ are *input matrices* and *input variables*; they will not be used in all applications of the Kalman Filter but are included for completeness. The noise processes are assumed to both be Gaussian, with time invariant covariances:

$$\begin{aligned} \boldsymbol{w}_t &\sim \mathcal{N}(0, \Sigma_w), \\ \boldsymbol{v}_t &\sim \mathcal{N}(0, \Sigma_v), \end{aligned}$$

and the initial state is also assumed to be Gaussian, $z_0 \sim \mathcal{N}(\mu_0, \Sigma_0)$.

Some additional notation may help prevent the presentation from becoming cluttered:

$$\begin{aligned} \mathbf{z}_{t|s} &= \mathbb{E}(\mathbf{z}_t|\mathbf{y}_{1:s}), \\ \boldsymbol{\Sigma}^{\mathbf{z}}_{t|s} &= \mathrm{Cov}(\mathbf{z}_t|\mathbf{y}_{1:s}), \\ \mathbf{y}_{t|s} &= \mathbb{E}(\mathbf{y}_t|\mathbf{y}_{1:s}), \\ \boldsymbol{\Sigma}^{\mathbf{y}}_{t|s} &= \mathrm{Cov}(\mathbf{y}_t|\mathbf{y}_{1:s}), \end{aligned}$$

and $(\mathbf{z}|\mathbf{y}) \sim \mathcal{N}(\mu, \Sigma)$ means the conditional distribution of $\mathbf{z}|\mathbf{y}$ is multivariate normal with mean μ and covariance Σ. Under the stated conditions of the model, the normality assumptions imply:

$$\begin{aligned} (\mathbf{z}_t|\mathbf{y}_{1:t-1}) &\sim \mathcal{N}\left(\mathbf{z}_{t|t-1}, \boldsymbol{\Sigma}^{\mathbf{z}}_{t|t-1}\right) \quad \text{for } t = 2, \ldots, T, \\ (\mathbf{z}_t|\mathbf{y}_{1:t}) &\sim \mathcal{N}\left(\mathbf{z}_{t|t}, \boldsymbol{\Sigma}^{\mathbf{z}}_{t|t}\right) \quad \text{for } t = 1, \ldots, T, \\ (\mathbf{y}_t|\mathbf{y}_{1:t-1}) &\sim \mathcal{N}\left(\mathbf{y}_{t|t-1}, \boldsymbol{\Sigma}^{\mathbf{y}}_{t|t-1}\right) \quad \text{for } t = 2, \ldots, T, \end{aligned}$$

and

$$\begin{aligned} (\mathbf{z}_t|\mathbf{y}_{1:T}) &\sim \mathcal{N}\left(\mathbf{z}_{t|T}, \boldsymbol{\Sigma}^{\mathbf{z}}_{t|T}\right) \quad \text{for } t > T, \\ (\mathbf{y}_t|\mathbf{y}_{1:T}) &\sim \mathcal{N}\left(\mathbf{y}_{t|T}, \boldsymbol{\Sigma}^{\mathbf{y}}_{t|T}\right) \quad \text{for } t > T. \end{aligned}$$

[Lütkepohl, 2007] The conditional means and covariance matrices for the data period can be obtained from the *Kalman filter recursions*, which together form a *predictor corrector method* for optimally filtering the underlying states.

The Kalman filter recursions are illustrated in Figure 4.2. The recursions begin at time $t = 0$ with knowledge of the initial state $\mathbf{z}_{0|0} = z_0$ and the initial state covariance $\boldsymbol{\Sigma}^{\mathbf{z}}_{0|0} = \Sigma_0$, which are used in the prediction step, to form *a priori* estimates for the state $\mathbf{z}_{1|0}$ and its covariance $\boldsymbol{\Sigma}^{\mathbf{z}}_{1|0}$ at time $t = 1$. Predictions are also made for the observation $\mathbf{y}_{1|0}$ and its covariance $\boldsymbol{\Sigma}^{\mathbf{y}}_{1|0}$. At time $t = 1$ a new measurement is obtained and the *innovation* $\mathbf{e}_t$ (the difference between the prediction and the observation) is calculated. The correction step updates the predictions in the light of the new information to calculate *a posteriori* estimates of the state $\mathbf{z}_{1|1}$ and its covariance $\boldsymbol{\Sigma}^{\mathbf{z}}_{1|1}$.

The updates are weighted by the *Kalman Filter Gain*, which is optimal in that it ensures the filter produces minimum mean square error (MMSE) estimates. The Kalman gain is proportional to the *a priori* state covariance matrix and inversely proportional to the innovation covariance matrix.

Algorithm 2: The Kalman Filter

Preditions

$$\mathbf{z}_{t|t-1} = \mathbf{F}\mathbf{z}_{t-1|t-1} + \mathbf{J}\mathbf{x}_{t-1} \quad \text{State Prediction}$$
$$\mathbf{\Sigma}^{\mathbf{z}}_{t|t-1} = \mathbf{F}\mathbf{\Sigma}^{\mathbf{z}}_{t-1|t-1}\mathbf{F}' + \Sigma_w \quad \text{State Covariance Prediction} \tag{4.7}$$
$$\mathbf{y}_{t|t-1} = \mathbf{H}_t\mathbf{z}_{t|t-1} + \mathbf{G}\mathbf{x}_t \quad \text{Observation Prediction}$$
$$\mathbf{\Sigma}^{\mathbf{y}}_{t|t-1} = \mathbf{H}_t\mathbf{\Sigma}^{\mathbf{z}}_{t|t-1}\mathbf{H}'_t + \Sigma_v \quad \text{Observation Covariance Prediction}$$

Corrections

$$\mathbf{y}_t \quad \text{Receive new Measurement}$$
$$\mathbf{e}_t = \mathbf{y}_t - \mathbf{y}_{t|t-1} \quad \text{Calculate the Innovation}$$
$$\mathbf{z}_{t|t} = \mathbf{z}_{t|t-1} + \mathbf{K}_t\mathbf{e}_t \quad \text{Update the State Estimate}$$
$$\mathbf{\Sigma}^{\mathbf{z}}_{t|t} = \mathbf{\Sigma}^{\mathbf{z}}_{t|t-1} - \mathbf{K}_t\mathbf{\Sigma}^{\mathbf{y}}_{t|t-1}\mathbf{K}'_t \quad \text{Update the State Covariance}$$

$$\mathbf{K}_t = \mathbf{\Sigma}^{\mathbf{z}}_{t|t-1}\mathbf{H}'_t(\mathbf{\Sigma}^{\mathbf{y}}_{t|t-1})^{-1} \quad \textbf{Kalman Filter Gain} \tag{4.8}$$

For each time step of the Kalman filter the entire history of states is summarized by a set of sufficient statistics; the *a posteriori* state $\mathbf{z}_{t|t}$ and state covariance $\mathbf{\Sigma}^{\mathbf{z}}_{\boldsymbol{t|t}}$. The Kalman filter provides the MMSE estimator of the state if the gain matrix $\mathbf{K}_t$ is chosen to be the Kalman Gain. Under these conditions the *trace* of the *a posteriori* estimated covariance matrix $\mathbf{\Sigma}^{\mathbf{z}}_{t|t}$ is minimized.

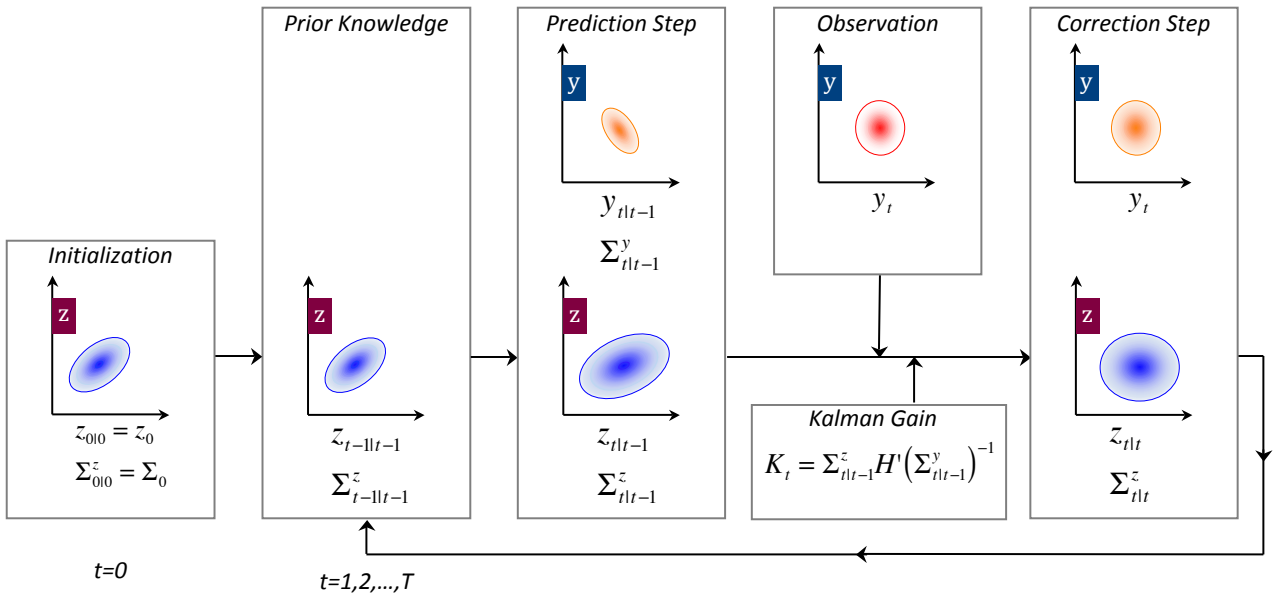


Figure 4.2: Diagram illustrating the Kalman recursions of Algorithm 2 for time steps $t = 1, \ldots, T$. The *a priori* predictions of the latent state and its covariance are updated to become *a posteriori* estimates once an observation is received.

4.3.2 Alternative Covariance Update Forms

The *a priori* predicted state covariance, $\mathbf{\Sigma}^{\mathbf{z}}_{\boldsymbol{t|t-1}}$, (4.7), is, by definition, a positive semi-definite matrix. However when applying the filter in practice, computed values of the matrix can lose this property due to computer numerical errors. If this occurs the Kalman gain may have an incorrect sign, and estimates produced by the filter may diverge (see [Anderson and Moore, 1979] for a discussion of these issues). An approach used to guard against these problems is to employ an alternative form of covariance update. Possible forms are listed in the following table.

Form	Equation
Standard	$\mathbf{\Sigma}^{\mathbf{z}}_{t\|t} = \mathbf{\Sigma}^{\mathbf{z}}_{t\|t-1} - \mathbf{K}_t \mathbf{\Sigma}^{\mathbf{y}}_{t\|t-1} \mathbf{K}'_t$
Joseph	$\mathbf{\Sigma}^{\mathbf{z}}_{t\|t} = (\mathbb{I}_n - \mathbf{K}_t \mathbf{H}_t)\, \mathbf{\Sigma}^{\mathbf{z}}_{t\|t-1}\, (\mathbb{I}_n - \mathbf{K}_t \mathbf{H}_t)' - \mathbf{K}_t \Sigma_v \mathbf{K}'_t$

Table 4.1: Alternative forms of covariance updates for the Kalman Filter

Remark *The standard covariance update can result in a loss of symmetry and positive definiteness due to computer rounding errors. The Joseph form is guaranteed to preserve positive definiteness and symmetry so long as* $\mathbf{\Sigma}^{\mathbf{z}}_{\boldsymbol{t|t-1}} \geq 0$*, however this comes at a greater computational cost that the standard form. The Joseph form holds for any gain,* $\mathbf{K}^{\star}_t$*, and not just the optimal gain* $\mathbf{K}_t$*. There are also further forms of the filter (called information filters) which update the inverse* $\mathbf{\Lambda}^{\mathbf{z}}_{t|t-1} = \left(\mathbf{\Sigma}^{\mathbf{z}}_{t|t-1}\right)^{-1}$.

4.3.3 The Log-Likelihood Function

We will be interested in obtaining the marginal log-likelihood of the data given the static parameters, $\log p(\mathbf{y}_{1:T}|\mathbf{\Theta})$, from the Kalman filter. We assume that the matrices $\mathbf{F}, \mathbf{H}_t, \mathbf{J}, \mathbf{G}, \Sigma_w$ and Σ_v all depend upon (or can be written as) a vector of time-invariant static parameters $\mathbf{\Theta}$ (i.e. $\mathbf{\Theta}$ is time invariant even if $\mathbf{H}_t$ is not [Lütkepohl, 2007]). By Bayes' theorem:

$$\begin{aligned} p(\mathbf{y}_1, \ldots, \mathbf{y}_T|\mathbf{\Theta}) &= p(\mathbf{y}_1, \mathbf{\Theta}) p(\mathbf{y}_2, \ldots, \mathbf{y}_T|\mathbf{y}_1, \mathbf{\Theta}) \\ &= p(\mathbf{y}_1, \mathbf{\Theta}) p(\mathbf{y}_2|\mathbf{y}_1, \mathbf{\Theta}) \ldots p(\mathbf{y}_T|\mathbf{y}_1, \ldots, \mathbf{y}_{T-1}\mathbf{\Theta}) \\ &= p(\mathbf{y}_1, \mathbf{\Theta}) \prod_{t=2}^{T} p(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{\Theta}). \end{aligned}$$

Therefore the log-likelihood can be written (for an observation vector with dimension n):

$$\begin{aligned} \mathcal{L}(\mathbf{y}_1, \ldots, \mathbf{y}_T|\mathbf{\Theta}) &= \ln p(\mathbf{y}_1, \ldots, \mathbf{y}_T|\mathbf{\Theta}) \\ &= \ln p(\mathbf{y}_1|\mathbf{\Theta}) + \sum_{t=2}^{T} \ln p(\mathbf{y}_t|\mathbf{y}_1, \ldots, \mathbf{y}_{t-1}, \mathbf{\Theta}) \\ &= \frac{nT}{2} \ln(2\pi) - \frac{1}{2} \sum_{t=2}^{T} \ln \left|\mathbf{\Sigma}^{\mathbf{y}}_{\boldsymbol{t|t-1}}\right| - \frac{1}{2} \sum_{t=2}^{T} \mathbf{e}'_t \left(\mathbf{\Sigma}^{\mathbf{y}}_{t|t-1}\right)^{-1} \mathbf{e}_t, \end{aligned} \tag{4.9}$$

where we have used $(\mathbf{y}_t|\mathbf{y}_{1:t-1}) \sim \mathcal{N}(\mathbf{y}_{t|t-1}, \mathbf{\Sigma}^{\mathbf{y}}_{t|t-1})$. In other words the log-likelihood is easily evaluated from the innovations and predicted covariances which are produced when running the Kalman filter, without the need for further computation.

4.3.4 Limitations and extensions of the Kalman Filter

The Kalman filter is the optimal *linear* filter and can be successfully applied to linear systems. However very few real systems are completely linear and various extensions to the filter to the case of non-linear systems have been proposed. The *Extended Kalman Filter* (EKF) uses a Taylor series style expansion to linearise about the currently estimated mean and covariance. Unlike the linear version, the EKF is in general not an optimal estimator, and if modelling or initial state estimates are incorrect the filter may quickly diverge. The EKF is in general more difficult to use than the Kalman filter and can be difficult to tune and can give poor results.

The *Unscented Kalman Filter* is a further potential improvement over the other filters, in which probability densities are represented by a deterministic sampling of points which represent the underlying distribution as a Gaussian. The deterministic sampling technique is called the *unscented transform*; the samples define a minimal set of *sigma points* around the mean. The sigma point samples are propagated through the non-linear functions that define the system, after which the mean and covariance can be recovered. The filter can more accurately capture the true mean and covariance than the simpler EKF technique. In some sense this method resembles the particle filter approaches explored in more detail in Chapter 5, however the points are chosen deterministically and for more complex distributions (i.e. very non-linear, bimodal examples, etc.) true particle filters are superior.

Chapter 5

Modern Particle Filters and Non-Linear Filtering

5.1 Introduction

Non linear non-Gaussian state-space models arise in numerous application domains, from control and signal processing, to econometrics, finance, and even ecology. Particle Filters, also known as Sequential Monte Carlo (SMC) methods, provide very good numerical approximations to the latent states estimation problem in these models. However, in many situations the state-space model of interest also depends on unknown static parameters that need to be estimated from the data. In these contexts standard SMC methods fail and it becomes necessary to rely on more sophisticated algorithms.

Specifically, in the case of the non-linear CVAR models introduced in Chapter 3 the target for inference is the *joint* estimation of the static parameters and the hidden latent system states. Not only is this scenario very high dimensional, due to the inclusion of the path space, but it has the added challenge of a complex matrix-variate setting with multiple unknown static parameters. The following subsection introduces the key types of simulation algorithm on which the overall strategy for attacking these complex problems depends, and the remainder of this Chapter explains each component of these algorithms in more depth.

5.1.1 From SMC to PMCMC

Consider a generic state-space model, as introduced in the previous Chapter, with parameters, θ, prior $p(\theta)$, and latent Markov process x_t[1], $p(\mathsf{x}_1|\theta) = \mu_\theta$:

$$p(\mathsf{x}_t|\mathsf{x}_{1:t-1}, \theta) = p(\mathsf{x}_t|\mathsf{x}_{t-1}, \theta) = f_\theta(\mathsf{x}_t|\mathsf{x}_{t-1}), \qquad t \geq 1, \tag{5.1}$$

and observation process

$$p(\mathsf{y}_t|\mathsf{y}_{1:t-1}, \mathsf{x}_{1:t-1}, \theta) = p(\mathsf{y}_t|\mathsf{x}_t, \theta) = g_\theta(\mathsf{y}_t|\mathsf{x}_t), \qquad t \geq 1.$$

We are interested in the recursive exploration of the sequence of posterior distributions:

$$\pi_0(\theta) = p(\theta), \quad \pi_t(\theta, \mathsf{x}_{1:t}|\mathsf{y}_{1:t}),$$

and may also be interested in computing the model evidence $p(\mathsf{y}_{1:t})$. The sequential analysis of state-space models is of interest in a wide variety of settings; even in the batch estimation setting (where fixed observation data $\mathsf{y}_{1:T}$ is available) recursive exploration may be computationally advantageous [Chopin, 2010]. SMC methods for this type of problem are considered state of the art. Their effectiveness stems from their efficient re-use of samples across different times t, which is in contrast to *Markov Chain Monte Carlo* (MCMC) methods - which are usually re-run for each time horizon. MCMC methods will, however, form a key component of the algorithms used in the

[1]Where we have dropped the bold-face on the vector states, for readability, but should retain the idea that these may be vector valued. Also we have switched to using x for the hidden states to conform to the prevailing convention in the literature.

joint estimation problem; they are introduced along with general Monte Carlo sampling methods in section 5.2.

Under mild assumptions SMC methods now have well understood convergence properties (with respect to the number of simulations), see for example [Crisan and Doucet, 2002] or [Chopin, 2004]. The problem of inference in the path space is very effectively addressed using SMC methods, which explore the simpler sequence of posteriors:

$$\pi_t(\mathsf{x}_t|\theta) = p(\mathsf{x}_t|\mathsf{y}_{1:t}, \theta) \tag{5.2}$$

This is a simplification compared to the general case, since the static parameters are treated as known and inferences are made concerning x_t rather than the whole path $\mathsf{x}_{1:t}$; this corresponds to the *filtering* problem as introduced in the previous Chapter. Particle filters evolve a collection of N, weighted, re-sampled particles $\mathsf{x}_t^{1:N}$, so that at each time t they represent a *properly weighted* sample from $\pi(\mathsf{x}_t|\theta)$. In this context *properly weighted* means that the importance weights given to each particle represent the unbiased estimate of the *Radon-Nikodym* derivative between the target and proposal distributions, see [Fearnhead et al., 2010] and also section 5.7.1 which appears later in this Chapter.

Unbiased estimators of the likelihood increments and the marginal likelihood are also available as a by-product of the particle filter output:

$$p(\mathsf{y}_{1:T}|\theta) = p(\mathsf{y}_1|\theta) \prod_{s=2}^{T} p(\mathsf{y}_s|\mathsf{y}_{1:s-1}, \theta), \qquad 1 \leq t \leq T, \tag{5.3}$$

the variance of which increases linearly with time [Chopin, 2010]. SMC methods are described in more detail in 5.7.

However, despite the success of standard SMC methods, the general case of sequential inference for both parameters and latent states for a generic, non-linear non-Gaussian, state-space model is a very challenging problem, which, although extremely important for a wide variety of applications, is still somewhat unresolved; see [Doucet et al., 2009] or [Andrieu et al., 2010] for recent discussions. Even the batch estimation problem of estimating $\pi_T(\mathsf{x}_{1:T}, \theta)$ is a difficult MCMC problem in its own right. The reasons for this are both the high dependence between the parameters and the latent process (which hampers Gibbs strategies, see [Papaspiliopoulos et al., 2007]), and the difficulty in designing efficient schemes for sampling from $\pi_T(\mathsf{x}_{1:T}|\theta)$.

To attempt to overcome these difficulties [Andrieu et al., 2010] developed *Particle Markov Chain Monte Carlo* (PMCMC) algorithms. These are MCMC algorithms which use a particle filter of size, N, to generate their proposal. The algorithm replaces the intractable true value of (5.3), with the unbiased estimator provided by the particle filter. Andrieu et al. ([Andrieu et al., 2010]) show that as N grows the PMCMC algorithm's behaviour approaches that of the theoretical MCMC algorithms which target the intractable $\pi_T(\theta)$, and also that for any fixed value of N the PMCMC algorithm admits $\pi_T(\mathsf{x}_{1:T}, \theta)$ as a stationary distribution [Chopin, 2010]. PMCMC methods are discussed in more depth in section 5.8.

5.2 Monte Carlo Methods

Monte Carlo methods are a class of stochastic computational statistical techniques that employ pseudo-random numbers to solve certain problems. For example Monte Carlo methods include stochastic integration, where a simulation based method is used to evaluate an integral, and Markov-Chain Monte Carlo (MCMC), where a Markov chain is constructed which hopefully converges to a stationary distribution of interest.

Consider attempting to approximate a probability density $\pi(\mathsf{x}_{1:t})$ for fixed t. If we sample N independent random variables, $\mathsf{X}_{1:t}^i$ for $i = 1, \ldots, N$, then the Monte Carlo method approximates

$\pi(\mathsf{x}_{1:t})$ by the empirical measure:

$$\widehat{\pi}(\mathsf{x}_{1:t}) = \frac{1}{N}\sum_{i=1}^{N}\delta_{X^i_{1:t}}(\mathsf{x}_{1:\mathsf{t}}), \tag{5.4}$$

where $\delta_{\mathsf{x}_0}(\mathsf{x})$ indicates the Dirac delta mass located at x_0.

Based on this approximation it is straightforward to approximate any of the marginals, for example:

$$\widehat{\pi}(\mathsf{x}_j) = \frac{1}{N}\sum_{i=1}^{N}\delta_{\mathsf{X}^i_j}(\mathsf{x}_j),$$

and the expectation of a test function $h_t(x_{1:t})$ given by:

$$\mathbb{E}_\pi(h_t) = \int h_t(\mathsf{x}_{1:\mathsf{t}})\pi(\mathsf{x}_{1:t})d\mathsf{x}_{1:t},$$

may be estimated by

$$\mathbb{E}^{MC}_\pi(h_t) = \int h_t(\mathsf{x}_{1:\mathsf{t}})\widehat{\pi}(\mathsf{x}_{1:t})d\mathsf{x}_{1:t} = \frac{1}{N}\sum_{i=1}^{N}h_t(\mathsf{X}^i_{1:t}).$$

Such an estimate may be shown to be unbiased, and to have variance given by

$$\mathrm{Var}\left\{\mathbb{E}^{MC}_\pi(h_t)\right\} = \frac{1}{N}\left(\int h^2_t(\mathsf{x}_{1:t})\widehat{\pi}(\mathsf{x}_{1:t})d\mathsf{x}_{1:t} - \mathbb{E}^2_\pi(h_t)\right).$$

The principal advantage of Monte Carlo methods is that the variance of the approximation error decreases at a rate $O(1/N)$ irrespective of the dimension of the space. However there are also two main problems with the basic method:

- In most practical applications it is impossible (or computationally infeasible) to draw samples directly from the distribution of interest.
- Even in the event that we can sample exactly from $\pi_t(\mathsf{x}_{1:\mathsf{t}})$ the computational complexity will increase at least linearly in t.

To address the first problem certain methods draw samples from a proposal distribution from which it is easy to sample, and then use a weighting or selection scheme to ensure that the samples represent the target distribution properly: these are respectively termed *importance sampling* and *rejection sampling* methods, and the weighting or selection ensures the bias from the proposal distribution does not influence the approximation of the target distribution. The second problem will be discussed in section 5.7.1

5.2.1 Rejection Sampling

Rejection sampling was introduced by von Neuman [von Neuman, 1951]. The fundamental idea is to sample from an easy to sample *proposal distribution*, $q(x)$, and to reject samples which are unlikely under the hard (or impossible) to sample *target distribution*, $p(x)$. The density of $p(x)$ is known up to a proportionality constant C, and the algorithm proceeds as:

Algorithm 3: Rejection Sampling

1. Draw $X \sim q(x)$;
2. Accept X as a sample from $p(x)$ with probability:

$$\frac{p(x)}{C \cdot q(x)} \tag{5.5}$$

otherwise go back to step 1.

The algorithm produces exact samples from the target distribution, and is most efficient when the proposal distribution is a good approximation to the target. The acceptance probability is inversely proportional to the constant C.

Remark *Rejection sampling is of interest since it will be used in the accept/reject stage of the MCMC section of the complete PMCMC algorithms.*

5.2.2 Importance Sampling

In rejection sampling compensation is made for the fact that the samples have come from the proposal distribution $q(x)$, instead of the target $p(x)$, by rejecting some of the values proposed by $q(x)$. Importance sampling is based on the idea of using weights to correct for the fact that we are sampling from the proposal distribution rather than the intended target distribution. The support of $q(x)$ is assumed to cover $p(x)$.

Importance sampling is derived from the identity:

$$\mathbb{E}_p(h(x)) = \int p(x)h(x)dx = \int q(x) \underbrace{\frac{p(x)}{q(x)}}_{=w(x)} h(x)dx = \int q(x)w(x)h(x)dx = \mathbb{E}_q(w(X)h(X)). \quad (5.6)$$

Assuming a sample $X_1, \ldots, X_N \sim q$, and provided $\mathbb{E}_q|w(X) \cdot h(X)|$ exists,

$$\frac{1}{N}\sum_{i=1}^{N} w(X_i)h(X_i) \xrightarrow{\substack{a.s. \\ N\to\infty}} \mathbb{E}_q(w(X) \cdot h(X))$$

and therfore, by (5.6)

$$\frac{1}{N}\sum_{i=1}^{N} w(X_i)h(X_i) \xrightarrow{\substack{a.s. \\ N\to\infty}} \mathbb{E}_q(w(X) \cdot h(X))$$

Therefore we can estimate $\mu = \mathbb{E}_p(h(X))$ using the samples drawn from $q(x)$, by

$$\widehat{\mu} = \frac{1}{N}\sum_{i=1}^{N} w(X_i)h(X_i). \quad (5.7)$$

While the expectation of the weights under $q(x)$, $\mathbb{E}_q\left(\frac{p(x)}{q(x)}\right) = \int \frac{p(x)}{q(x)}q(x)dx = \int p(x)dx = 1$, the weights themselves $w_1(X), \ldots, w_N(X)$ do not necessarily sum up to N, so we might prefer to use the normalized version:

$$\widetilde{w}(X_i) = \frac{w(X_i)}{\sum_{i=1}^{N} w(X_i)}$$

Algorithm 4: Importance Sampling

Choose q such that the support of q covers that of $p \cdot h$;
For i=1,...,N;
1. Draw $X_i \sim q(x)$;
2. Set $w(X_i) = \frac{p(X_i)}{q(X_i)}$;
3. Return either:

$$\widehat{\mu} = \frac{\sum_{i=1}^{N} w(X_i)h(X_i)}{N} \quad \text{(Un-normalized)}$$

$$\text{Or} \qquad \widetilde{\mu} = \sum_{i=1}^{N} \widetilde{w}(X_i)h(X_i) = \frac{\sum_{i=1}^{N} w(X_i)h(X_i)}{\sum_{i=1}^{N} w(X_i)} \quad \text{(Normalized)} \quad (5.8)$$

Remark *Importance sampling is relevant since the SMC methods which perform the filtering stage in PMCMC algorithms implement a type of importance sampling in a sequential fashion, see 5.7.1 for details.*

Bias and Variance of Importance Sampling

The bias and variance of the un-normalized, $\widehat{\mu}$ (5.7), and normalized, $\widetilde{\mu}$ (5.8) estimators are:

$$\mathbb{E}_q(\widehat{\mu}) = \mathbb{E}_q\left(\frac{1}{N}\sum_{i=1}^{N} w(X_i)h(X_i)\right) = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_q(w(X_i)h(X_i)) = \mathbb{E}_p(h(X))$$

$$\mathrm{Var}_q(\widehat{\mu}) = \mathrm{Var}_q\left(\frac{1}{N}\sum_{i=1}^{N} w(X_i)h(X_i)\right) = \frac{1}{N^2}\sum_{i=1}^{N}\mathrm{Var}_q(w(X_i)h(X_i)) = \frac{\mathrm{Var}_q(w(X_i)h(X_i))}{N}$$

$$\mathbb{E}_q(\widetilde{\mu}) = \mu + \frac{\mu\mathrm{Var}_q(w(X)) - \mathrm{Cov}_q(w(X), w(X)h(X))}{N} + O(N^{-2}) \tag{5.9}$$

$$\mathrm{Var}_q(\widetilde{\mu}) = \frac{\mathrm{Var}_q(w(X)h(X)) - 2\mu\mathrm{Cov}_q(w(X), w(X)h(X)) + \mu^2\mathrm{Var}_q(w(X))}{N} + O(N^{-2}) \tag{5.10}$$

see [Johansen and Evans, 2007]. These results imply that unlike the un-normalized version, the normalized estimator $\widetilde{\mu}$ is biased. It is consistent however, with the bias vanishing at a rate bounded above by N. The normalized estimator may also have a lower variance. For practical applications of importance sampling it is crucial that the variances of the importance weights remain finite. When this condition is not met the variance of $\widetilde{\mu}$ will be infinite. Finite importance weights can be obtained through careful selection of the proposal distribution $q(x)$.

Optimal Proposal Distribution

In the case that we already know the function of interest $h(x)$, the proposal distribution that minimizes the variance of $\widehat{\mu}$ is:

$$q^*(x) = \frac{|h(x)|p(x)}{\int |h(s)|p(s)ds}. \tag{5.11}$$

Consider

$$\begin{aligned}
\mathrm{Var}_q(\widehat{\mu}) &= \frac{1}{N}\mathrm{Var}_q\left(h(x)w(x)\right) \\
&= \frac{1}{N}\mathrm{Var}_q\left(\frac{h(x)\cdot p(x)}{q(x)}\right) \\
&= \frac{1}{N}\mathbb{E}_q\left\{\left(\frac{h(x)\cdot p(x)}{q(x)}\right)^2\right\} - \frac{1}{N}\mathbb{E}_q\left\{\left(\frac{(h(x)\cdot p(x))}{q(x)}\right)\right\}^2 \\
&= \frac{1}{N}\int\left\{\frac{(h(x)\cdot p(x))^2}{q(x)}\right\}dx - \frac{\mu^2}{N}
\end{aligned}$$

The variance is minimized by minimizing the first term and the optimal choice for $q(x)$ is therefore $q(x) \propto |h(x)|p(x)$, which obviously cannot be made in practice since it would remove the need for importance sampling. Plugging in q^* gives:

$$\begin{aligned}
\mathbb{E}_{q*}\left(\left(\frac{h(x)\cdot p(x)}{q^*(x)}\right)^2\right) &= \int \frac{h(x)^2\cdot p(x)^2}{q^*(x)}dx \\
&= \left(\int \frac{h(x)^2\cdot p(x)^2}{|h(x)|p(x)}dx\right)\cdot\left(\int |h(s)|p(s)ds\right) \\
&= \left(\int |h(x)|p(x)dx\right)^2
\end{aligned} \tag{5.12}$$

The practical point being that the proposal distribution $q(x)$ should be chosen to have a shape which is close to that of $p \cdot h(x)$. We can also apply Jensen's inequality to the first term in the variance:

$$\mathbb{E}_q\left(\left(\frac{h(x)\cdot p(x)}{q(x)}\right)^2\right) \geq \left(\mathbb{E}_q\left(\frac{|h(x)|\cdot p(x)}{q(x)}\right)\right)^2 = \left(\int |h(x)p(x)|dx\right)^2. \tag{5.13}$$

A key result is that importance sampling using proposal q^* can actually be *super-efficient*; i.e. more efficient than sampling directly from p:

$$\begin{aligned} N\cdot \mathrm{Var}_p\left(\frac{h(X_1)+\ldots h(X_N)}{N}\right) &= \mathbb{E}_p(h(X^2)) - \mu^2 \\ &\geq (\mathbb{E}_p|h(X)|)^2 - \mu^2 \\ &= \left(\int |h(x)p(x)|dx\right)^2 - \mu^2 \\ &= \mathrm{Var}_{q*}(\widehat{\mu}) \end{aligned} \tag{5.14}$$

where the use of q^* concentrates the attention of our sampling on regions of high probability where $|h|$ is large contributing most to the integral, see [Johansen and Evans, 2007] for more details of these points. Finally it is advisable to select $q(x)$ such that the weights $w(x)$ remain bounded by a positive finite constant.

5.2.3 Markov Chain Monte Carlo

Markov Chain Monte Carlo (MCMC) methods are a very general and powerful framework for sampling from a large class of distributions, and they scale well even in sample spaces of very high dimensionality. These techniques originate in physics [Metropolis and Ulam, 1949] and are now very widely researched and applied by statisticians.

Remark *MCMC methods underlie the portion of the PMCMC algorithms used for estimation of the static model parameters for the cointegration models.*

As with rejection and importance sampling they rely on the use of proposal distributions, but now they also retain a record of the states that they visit and the proposal distribution $q(\mathsf{x}|\mathsf{x}^{(t)})$ is allowed to depend upon the current state. The sequence of sampled states $\mathsf{x}^{(t)} = \mathsf{x}^{(1)}, \mathsf{x}^{(2)} \ldots, \mathsf{x}^{(T)}$ forms a Markov chain. Suppose the density of the desired states $p(\mathsf{x})$ is available up to an (unknown) normalizing constant, Z:

$$p(\mathsf{x}) = \frac{\widetilde{p}(\mathsf{x})}{Z}.$$

In other words we are able to readily evaluate $\widetilde{p}(\mathsf{x})$ for any of the states, but not to sample from it directly.

The algorithm uses a proposal distribution $q(\cdot|\mathsf{x}^{(t)})$ that is chosen so that it is straightforward to sample from. At each step of the algorithm we generate a proposed new state x^*, drawn from the proposal distribution, which is then accepted or rejected according to an acceptance criterion.

In the original *Metropolis* version of the algorithm [Metropolis et al., 1953] a symmetric proposal distribution was used (i.e. $q(\mathsf{x}^{(1)}|\mathsf{x}^{(2)}) = q(\mathsf{x}^{(2)}|\mathsf{x}^{(1)})$). The proposed candidate state is accepted with probability:

$$A(\mathsf{x}^*, \mathsf{x}^{(t)}) = \min\left\{1, \frac{\widetilde{p}(\mathsf{x}^*)}{\widetilde{p}(\mathsf{x}^{(t)})}\right\} \tag{5.15}$$

This can be achieved easily by sampling a uniform random number u from the interval $(0, 1)$ and accepting the proposed state if $A(\mathsf{x}^*, \mathsf{x}^{(t)}) > u$. For all proposed new states which increase the

value of $\widetilde{p}(\mathsf{x})$, taking the minimum with 1 in the acceptance criterion ensures that the new state will be accepted. If the candidate state is accepted then $\mathsf{x}^{(t+1)} = \mathsf{x}^*$, otherwise the candidate state, x^*, is discarded, $\mathsf{x}^{(t+1)} = \mathsf{x}^{(t)}$ and the chain remains in the current state.

If the value of $q(\mathsf{x}^{(1)}|\mathsf{x}^{(2)})$ is positive for all states $\mathsf{x}^{(1)}$ and $\mathsf{x}^{(2)}$ then the distribution of $\mathsf{x}^{(t)}$ will tend to $p(\mathsf{x})$ as $t \to \infty$. The samples produced by the Metropolis algorithm are not independent samples from the distribution, since successive samples are correlated. Figure 5.1 shows samples of a bivariate Gaussian density generated using the Metropolis algorithm.

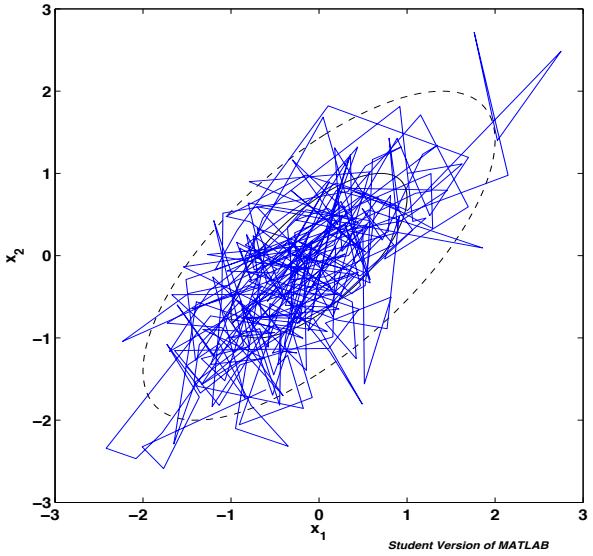


Figure 5.1: This plot shows samples drawn from a bivariate Gaussian density using the Metropolis algorithm. The ellipses are the 1 and 2 standard deviation contours of the target density, where the covariances are $\sigma_1^2 = 1$, $\sigma_2^2 = 1$, and $\rho_{1,2} = 0.7$. The proposal density was chosen to be a symmetric Gaussian centred on the origin. The blue lines show the accepted steps.

Algorithm 5: Metropolis Algorithm

Choose symmetric $q(\cdot|\mathsf{x})$;

1. Draw candidate state $X_i \sim q(\mathsf{x}^*|\mathsf{x}^{(t)})$;
2. Accept candidate state with probability

$$A(\mathsf{x}^*, \mathsf{x}^{(t)}) = \min\left\{1, \frac{\widetilde{p}(\mathsf{x}^*)}{\widetilde{p}(\mathsf{x}^{(t)})}\right\},$$

and set $\mathsf{x}^{(t+1)} = \mathsf{x}^*$, otherwise set $\mathsf{x}^{(t+1)} = \mathsf{x}^{(t)}$.

Markov Chains

It is useful to briefly discuss under what conditions we can expect a Markov chain to converge to its stationary distribution (these basic points follow the presentation in [Bishop, 2006]). A first order Markov chain is defined to be a series of random variables $\mathsf{x}^{(1)}, \mathsf{x}^{(2)}, \ldots, \mathsf{x}^{(T)}$ (which we will also refer to as states) such that:

$$p(\mathsf{x}^{(t+1)}|\mathsf{x}^{(1)}, \ldots, \mathsf{x}^{(T)}) = p(\mathsf{x}^{(t+1)}|\mathsf{x}^{(t)}),$$

in other words the state at time $t+1$ is conditionally independent of all states other than the one at time t. A Markov chain is said to be *homogeneous* if the *transition probabilities* that define the chain $T_t(\mathsf{x}^{(t)}, \mathsf{x}^{(t+1)}) = p(\mathsf{x}^{(t+1)}|\mathsf{x}^{(t)})$ are the same for all t. Marginal probabilities for specific variables in the chain can be expressed in terms of the probabilities of previous variables and the transition probabilities:

$$p(\mathsf{x}^{(t+1)}) = \sum_{\mathsf{x}^{(t)}} T(\mathsf{x}^{(t)}, \mathsf{x}^{(t+1)}) p(\mathsf{x}^{(t)})$$

A distribution is called *invariant* or *stationary*, with respect to a Markov chain, if the steps through the chain leave the distribution invariant. Therefore for a homogeneous Markov chain the distribution $p^*(\mathsf{x})$ is invariant if

$$p^*(\mathsf{x}) = \sum_{\mathsf{x}'} T(\mathsf{x}', \mathsf{x}) p^*(\mathsf{x}').$$

A single Markov chain may have more than one stationary distribution. A sufficient (but not necessary) condition for the states to be invariant is to select the transition probabilities to satisfy the property of *detailed balance*, which is given by:

$$p^*(\mathsf{x}) T(\mathsf{x}, \mathsf{x}') = p^*(\mathsf{x}') T(\mathsf{x}', \mathsf{x}), \tag{5.16}$$

which means that the probability of being in state x and moving to state x' is the same as the probability of starting in state x' and moving in the opposite direction. A transition probability that satisfies the detailed balance condition with respect to a particular distribution leaves that distribution invariant:

$$\begin{aligned} p^*(\mathsf{x}) &= \sum_{\mathsf{x}'} p^*(\mathsf{x}') T(\mathsf{x}', \mathsf{x}) \\ &= \sum_{\mathsf{x}'} p^*(\mathsf{x}) T(\mathsf{x}, \mathsf{x}') = \sum_{\mathsf{x}'} p^*(\mathsf{x}) p(\mathsf{x}'|\mathsf{x}) = p^*(\mathsf{x}). \end{aligned}$$

A Markov chain that respects the detailed balance condition is called *reversible*. A Markov chain that converges to the desired stationary distribution regardless of the choice of initial conditions is called *ergodic*, and such a chain will obviously only have a single stationary distribution - called the *equilibrium distribution*.

5.2.4 The Metropolis-Hastings Algorithm

The *Metropolis-Hastings* algorithm [Hastings, 1970] is a generalization of the earlier Metropolis algorithm to the case of non-symmetric proposal distributions. At time t of the algorithm, in current state $\mathsf{x}^{(t)}$ a sample is proposed from $q_a(\mathsf{x}^*|\mathsf{x}^{(t)})$ (where now $q_a(\mathsf{x}^{(1)}|\mathsf{x}^{(2)}) \neq q_a(\mathsf{x}^{(2)}|\mathsf{x}^{(1)})$). The acceptance probability for the new state x^* is modified to compensate for the asymmetric proposal, and is

$$A(\mathsf{x}^*, \mathsf{x}^{(t)}) = \min\left\{1, \frac{\widetilde{p}(\mathsf{x}^*) q_a(\mathsf{x}^{(t)}|\mathsf{x}^*)}{\widetilde{p}(\mathsf{x}^{(t)}) q_a(\mathsf{x}^*|\mathsf{x}^{(t)})}\right\}.$$

Algorithm 6: Metropolis-Hastings Algorithm

for *times* $t = 1$ *to* T **do**

1. Draw candidate state $X_i \sim q_a(\mathsf{x}^*|\mathsf{x}^{(t)})$;
2. Accept candidate state with probability:

$$A(\mathsf{x}^*, \mathsf{x}^{(t)}) = \min\left\{1, \frac{\widetilde{p}(\mathsf{x}^*) q_a(\mathsf{x}^{(t)}|\mathsf{x}^*)}{\widetilde{p}(\mathsf{x}^{(t)}) q_a(\mathsf{x}^*|\mathsf{x}^{(t)})}\right\}. \tag{5.17}$$

and set $\mathsf{x}^{(t+1)} = \mathsf{x}^*$, otherwise set $\mathsf{x}^{(t+1)} = \mathsf{x}^{(t)}$.

By using the detailed balance condition (5.16), the distribution of $p(\mathsf{x})$ can be shown to be an invariant distribution of the Markov chain defined by the Metropolis-Hastings algorithm:

$$\begin{aligned} p(\mathsf{x}) q_a(\mathsf{x}^*|\mathsf{x}) A(\mathsf{x}^*, \mathsf{x}) &= \min\left\{p(\mathsf{x}) q_a(\mathsf{x}^*|\mathsf{x}), p(\mathsf{x}^*) q_a(\mathsf{x}|\mathsf{x}^*)\right\}, \\ &= \min\left\{p(\mathsf{x}^*) q_a(\mathsf{x}|\mathsf{x}^*), p(\mathsf{x}) q_a(\mathsf{x}^*|\mathsf{x})\right\}, \\ &= p(\mathsf{x}^*) q_a(\mathsf{x}|\mathsf{x}^*) A(\mathsf{x}, \mathsf{x}^*) \end{aligned}$$

5.2.5 Gibbs Samplers

The *Gibbs Sampler* [Geman and Geman, 1984] is a special case of the more general Metropolis-Hastings algorithm which samples sequentially from a series of *full conditional* distributions, which are the distributions of one variable in a space conditioned on the values of all the other variables. Consider the case of sampling from a multidimensional space $p(\mathsf{x}_1^{(t)}, \mathsf{x}_2^{(t)}, \ldots, \mathsf{x}_d^{(t)})$. Each step of the algorithm samples a variable for one of the dimensions conditioned on the current values of the variables in the other dimensions, for example:

$$\text{Sample } \mathsf{x}_1^* \sim p(\mathsf{x}_1 | \mathsf{x}_2^{(t)}, \ldots, \mathsf{x}_d^{(t)}),$$
$$\text{Sample } \mathsf{x}_2^* \sim p(\mathsf{x}_2 | \mathsf{x}_1^{(t)}, \mathsf{x}_3^{(t)}, \ldots, \mathsf{x}_d^{(t)}) \quad \text{etc.}$$

The complete systematic sweep Gibbs sampling algorithm is given in Algorithm 7, and 500 samples from a bivariate Gaussian example are plotted in Figure 5.2.

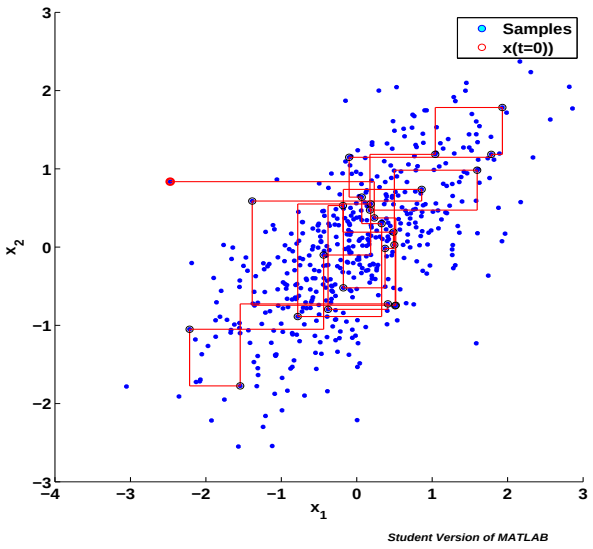


Figure 5.2: This plot shows samples drawn from the same bivariate Gaussian density as the Metropolis example in Figure 5.1 (correlation $\rho = 0.7$) using the sequential sweep Gibbs algorithm. For each step the x_1 variable is sampled conditional on the previous value of the x_2 variable, and then vice versa. The proposal density was chosen to be a symmetric Gaussian centred on the origin. The red lines show the first 25 Gibbs conditional moves, and the larger red spot is the starting position.

Algorithm 7: Systematic Sweep Gibbs Sampling Algorithm

0. Initialize $p(\mathsf{x}_1^{(0)}, \mathsf{x}_2^{(0)}, \ldots, \mathsf{x}_d^{(0)})$

for *times* $t = 1$ *to* T **do**

1. Sample $\mathsf{x}_1^{(t)} \sim p(\mathsf{x}_1 | \mathsf{x}_2^{(t-1)}, \ldots, \mathsf{x}_d^{(t-1)})$
- Sample $\mathsf{x}_2^{(t)} \sim p(\mathsf{x}_2 | \mathsf{x}_1^{(t)}, \mathsf{x}_3^{(t-1)}, \ldots, \mathsf{x}_d^{(t-1)})$
- ⋮
- Sample $\mathsf{x}_3^{(t)} \sim p(\mathsf{x}_3 | \mathsf{x}_1^{(t)}, \mathsf{x}_2^{(t)}, \ldots, \mathsf{x}_d^{(t-1)})$
- ⋮
- Sample $\mathsf{x}_d^{(t)} \sim p(\mathsf{x}_d | \mathsf{x}_1^{(t)}, \mathsf{x}_2^{(t)}, \ldots, \mathsf{x}_{d-1}^{(t)})$

The systematic sweep version of the Gibbs algorithm is not reversible, however there is also a random sweep version which does produce a reversible chain.

The *ergodic theorem for Gibbs Samplers* states that: *If the Markov chain generated by the Gibbs sampler is irreducible and recurrent, then for any integrable function h:*

$$\lim_{n\to\infty} \frac{1}{N} \sum_{t=1}^{N} h(\mathsf{X}^{(t)}) \to \mathbb{E}_f(h(\mathsf{X})) \tag{5.18}$$

for almost every starting value $\mathsf{X}^{(0)}$, [Johansen and Evans, 2007]. This allows us to perform inference using a single Markov chain, and a very similar result also holds for Metropolis-Hastings samplers.

Block Gibbs

The basic versions of the Gibbs Sampler updates a single variable at a time, which creates strong dependencies between successive samples. The other extreme would be to update the full joint distribution simultaneously, which we have assumed is intractable, but this would result in independent samples. A possible intermediate alternative, which we will explore in more detail for the Bayesian CVAR models, is to update groups or *blocks* of variables instead of single variables. See [Jensen et al., 1995]

5.3 Adaptive MCMC

Various classes of *adaptive* MCMC algorithms have been proposed in the literature, for examples see [Roberts and Rosenthal, 2009]. Unlike the MCMC algorithms described so far, Adaptive MCMC (AdMCMC) algorithms generate their Markov chains via a combination of time or state inhomogeneous proposal kernels. The proposal kernels are allowed to depend upon the past history of the Markov chain and most of the literature on adaption describes on-line AdMCMC algorithms. Since the Markov kernels used in AdMCMC are inhomogeneous it is very important to ensure that the chains produced are ergodic with the desired stationary distribution.

Recently, several papers have proposed theoretical conditions which must be satisfied to guarantee ergodicity in AdMCMC algorithms, these include [Haario et al., 2001], [Andrieu and Atachade, 2005], [Andrieu and Moulines, 2006], and [Andrieu and Atachade, 2007]. A proof of ergodicity under conditions, known as *Diminishing Adaptation* and *Bounded Convergence* is provided by the authors in [Roberts and Rosenthal, 2009]. In general it is non-trivial to develop adaption schemes which can be shown to fulfil these conditions. We will use an AdMCMC algorithm to learn the proposal distribution for certain of the static parameters in our Bayesian CVAR models. The particular Adaptive Metropolis algorithm chosen uses a mixture proposal kernel that is known to satisfy the two ergodicity conditions for unbounded state spaces and general classes of target posterior distributions [Peters et al., 2010].

5.3.1 Adaptive Metropolis Proposal Kernel for $\boldsymbol{\beta}$

The proposal kernel is an adaptive Gaussian mixture distribution, one component of which has a covariance structure that is learnt adaptively on-line as the algorithm explores the posterior distribution. Consider the specific case of sampling from the posterior distribution of the cointegrating vector $\boldsymbol{\beta}$, in the Bayesian CVAR models, $p(\boldsymbol{\beta}|Y)$. An adaptive mixture proposal distribution was implemented for parameters $\boldsymbol{\beta}^{\star}$ (size $d = (n-r)\times r$) of the form:

$$Q_j(\boldsymbol{\beta}^{*}|\boldsymbol{\beta}^{(j-1)}) = w\mathcal{N}\left(\boldsymbol{\beta}^{\star}\left|\boldsymbol{\beta}^{(j-1)}, \frac{(2.38)^2}{d}\,\Omega_j\right.\right) + (1-w)\mathcal{N}\left(\boldsymbol{\beta}^{\star}\left|\boldsymbol{\beta}^{(j-1)}, \frac{(0.1)^2}{d}\,I_{d,d}\right.\right), \qquad (5.19)$$

where Ω_j is the current empirical estimate of the covariance matrix for the unrestricted section of the parameters $\boldsymbol{\beta}^{\star}$, and where the theoretical motivations for the scale factors 2.38, 0.1 and dimension d are given in [Roberts and Rosenthal, 2009]. The parameter w of the mixture was set to 0.95, based on the suggestion in the same paper. The empirical estimate for Ω_j is calculated using samples from the Markov chain up to time j via:

$$\begin{aligned}
\mu_{j+1} &= \mu_j + \frac{1}{j+1}\big(\boldsymbol{\beta}^{(j-1)} - \mu_j\big),\\
\Omega_{j+1} &= \Omega_j + \frac{1}{j+1}\left(\big(\boldsymbol{\beta}^{(j-1)} - \mu_j\big)\big(\boldsymbol{\beta}^{(j-1)} - \mu_j\big)' - \Omega_j\right),
\end{aligned} \qquad (5.20)$$

5.3.2 Laplace method for Proposal Covariance

Even with the adaptive scheme just described it is necessary to start the Metropolis algorithm with some initial proposal covariance. We wish to use an appropriate covariance matrix in the multivariate Gaussian proposal distribution $Q(\boldsymbol{\beta}^*|\boldsymbol{\beta}^{(j-1)}) \sim \mathcal{N}(\boldsymbol{\beta}^*|\boldsymbol{\beta}^{(j-1)}, \Omega^{(j)})$. The Laplace method makes an empirical estimate of the mode and nearby curvature of the posterior surface and uses these to inform the choice of covariance matrix; this amounts to performing a second order Taylor expansion in the vicinity of the mode (for a Gaussian distribution only the moments up to second order exist). At the mode of the posterior distribution the log posterior can be approximated as:

$$\log p(\boldsymbol{\beta}|Y) \approx \log p(\boldsymbol{\beta_0}|Y) - \frac{1}{2}(\boldsymbol{\beta} - \boldsymbol{\beta_0})' A (\boldsymbol{\beta} - \boldsymbol{\beta_0}) + \ldots,$$

where

$$A_{ij} = -\frac{\partial^2}{\partial\beta_i \partial\beta_j} \log p(\boldsymbol{\beta}|Y)\Big|_{\boldsymbol{\beta}=\boldsymbol{\beta}_0}$$

The appropriate value to use for the Gaussian proposal covariance matrix is therefore:

$$\Omega_{\text{Laplace}} = [-\nabla\nabla \log p(\boldsymbol{\beta}|Y)]^{-1},$$

the inverse of the negative of the Hessian matrix.

5.4 *Case Study:* Adaptive Metropolis scheme for $\boldsymbol{\beta}$: effect of prior choice.

A random walk Metropolis sampler (Algorithm 8) was coded up for parameter $\boldsymbol{\beta}$, in order to draw samples from the posterior density:

$$p(\boldsymbol{\beta}|Y) \propto p(\boldsymbol{\beta})|S_\star|^{-(t+h+1)/2}|A_\star|^{-n/2}$$

The proposal covariance matrix ($\Omega^{(j)}$) can be specified either through a Laplace approximation to the posterior surface local curvature, or via the adaptive scheme described previously.

Algorithm 8: Random Walk Metropolis Algorithm

for *iterations j=1 to J* **do**

1. Generate a proposal for $\boldsymbol{\beta}^*$ from $Q(\boldsymbol{\beta}^*|\boldsymbol{\beta}^{(j-1)}) \sim \mathcal{N}(\boldsymbol{\beta}^*|\boldsymbol{\beta}^{(\boldsymbol{j-1})}, \Omega^{(j)})$
2. Accept the proposed move with probability $A(\boldsymbol{\beta}^{(j-1)}, \boldsymbol{\beta}^*) = \min\left\{1, \frac{p(\boldsymbol{\beta}^*|Y)}{p(\boldsymbol{\beta}^{(\boldsymbol{j-1})}|Y)}\right\}$, and set $\boldsymbol{\beta}^{(j)} = \boldsymbol{\beta}^*$

Otherwise set $\boldsymbol{\beta}^{(j)} = \boldsymbol{\beta}^{(j-1)}$.

The priors for $\boldsymbol{\beta}$ were set to those specified in [Peters et al., 2010], and are listed in Table 5.1 below:

$\boldsymbol{\beta}$	Σ	$B\|\Sigma$
$\bar{\boldsymbol{\beta}} = \mathbb{E}[\boldsymbol{\beta}] = [I_r\ \mathbf{0}]'$	$S = \tau Y'Y$	$P = (\widehat{W}'\widehat{W})^{-1}\widehat{W}Y$
$Q = I_n$	$h = n + 1$	$A = \lambda(\widehat{W}'\widehat{W})/T$
$H = \tau Z'Z$	$\tau = 1/T$	$\widehat{W} = [X\ Z\widehat{\boldsymbol{\beta}}]$

Table 5.1: Priors for $\boldsymbol{\beta}$

The rank 2 model from [Sugita, 2002] was chosen to test the performance of the Metropolis sampling algorithm running with both the adaptive scheme, and a non-adaptive scheme using the Laplace method proposal covariance.

Sugita Rank 2 Model:

$$\Delta x_t = \boldsymbol{\mu} + \underbrace{\begin{bmatrix} -0.2 & -0.2 \\ 0.2 & -0.2 \\ 0.2 & 0.2 \\ -0.2 & 0.2 \end{bmatrix}}_{\boldsymbol{\alpha}} \underbrace{\begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \end{bmatrix}}_{\boldsymbol{\beta}'} x_{t-1} + \boldsymbol{\epsilon}_t \tag{5.21}$$

This is a cointegrated VAR of rank 2, with dimension $n = 4$. Sugita sets $\boldsymbol{\mu} = [\ 0.1 \quad 0.1 \quad 0.1 \quad 0.1\ ]'$. We note that in this model there are no lag matrices ($\boldsymbol{\Psi}_i$). The observation errors are assumed independent and come from $\boldsymbol{\epsilon}_t \sim N(\mathbf{0}, \Sigma) = N(\mathbf{0}, 0.1 \times I_n)$.

Posterior Surface: The posterior surface of $\log[p(\boldsymbol{\beta}^\star|Y)]$ as a function of $\boldsymbol{\beta}_{1,3}, \boldsymbol{\beta}_{2,3}$ is plotted in Figure 5.3 below[2]. 100 data points were generated from the Sugita rank 2 model (equation 5.21) and then used with the priors specified in Table 5.1 to calculate the log-posterior $\log[p(\boldsymbol{\beta}^\star|Y)]$. Two different values for the prior mean $\bar{\boldsymbol{\beta}}$ were tested, either

$$\bar{\boldsymbol{\beta}} = \begin{bmatrix} I_r \\ \mathbf{0} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}'$$

as suggested in Peters [Peters et al., 2010] or, subsequently, the true values setting $\bar{\boldsymbol{\beta}} = \boldsymbol{\beta}$. As is clear from the upper surface and contour plots in Figure 5.3, with a default setting for the prior mean $\bar{\boldsymbol{\beta}} = [I_r\ \mathbf{0}]'$ the shape of the posterior is highly elongated. On setting $\bar{\boldsymbol{\beta}} = \boldsymbol{\beta}_{\text{Truth}}$ the mode now more obviously corresponds to the correct parameter values. Changing the prior parameters has affected the posterior surface shape, and this also alters the convergence of the Metropolis algorithm. The sample plots in Figure 5.3 show that running the Metropolis algorithm on the more elongated posterior surface for the *default* prior settings results in poorer convergence of the chain; only around 5% of the proposed moves were accepted. Despite this the true parameter values have been recovered (the initial state was set away from the true values). Also the mixing isn't as good as is required to treat the samples as approximately independent since the autocorrelation function still has a value of around 0.7 after lag $l = 20$.

Running the Metropolis algorithm on the posterior with the prior set to the *true* parameters for $\bar{\boldsymbol{\beta}}$ results in a chain with more efficient convergence. The autocorrelation function has decayed to around 0.2 after lag $l = 20$, and the true parameters have been recovered. Around 13% of proposed moves were accepted.

These plots also demonstrate the effect of switching on the adaptive MCMC proposal covariance; after step 10,000 in each algorithm the adaptive scheme was started. The chain can be seen to almost immediately find the true parameter values in the default prior case, and settle into more robust convergence on these values for the case of the truth prior.

[2]Where $\boldsymbol{\beta} = \begin{bmatrix} \boldsymbol{\beta}_{1,1} & \boldsymbol{\beta}_{2,1} \\ \boldsymbol{\beta}_{1,2} & \boldsymbol{\beta}_{2,2} \\ \boldsymbol{\beta}_{1,3} & \boldsymbol{\beta}_{2,3} \\ \boldsymbol{\beta}_{1,4} & \boldsymbol{\beta}_{2,4} \end{bmatrix}$, and I have highlighted the 'unrestricted' portion of the matrix (which we label $\boldsymbol{\beta}^\star$) in cyan (the other part being determined by the r^2 restriction).

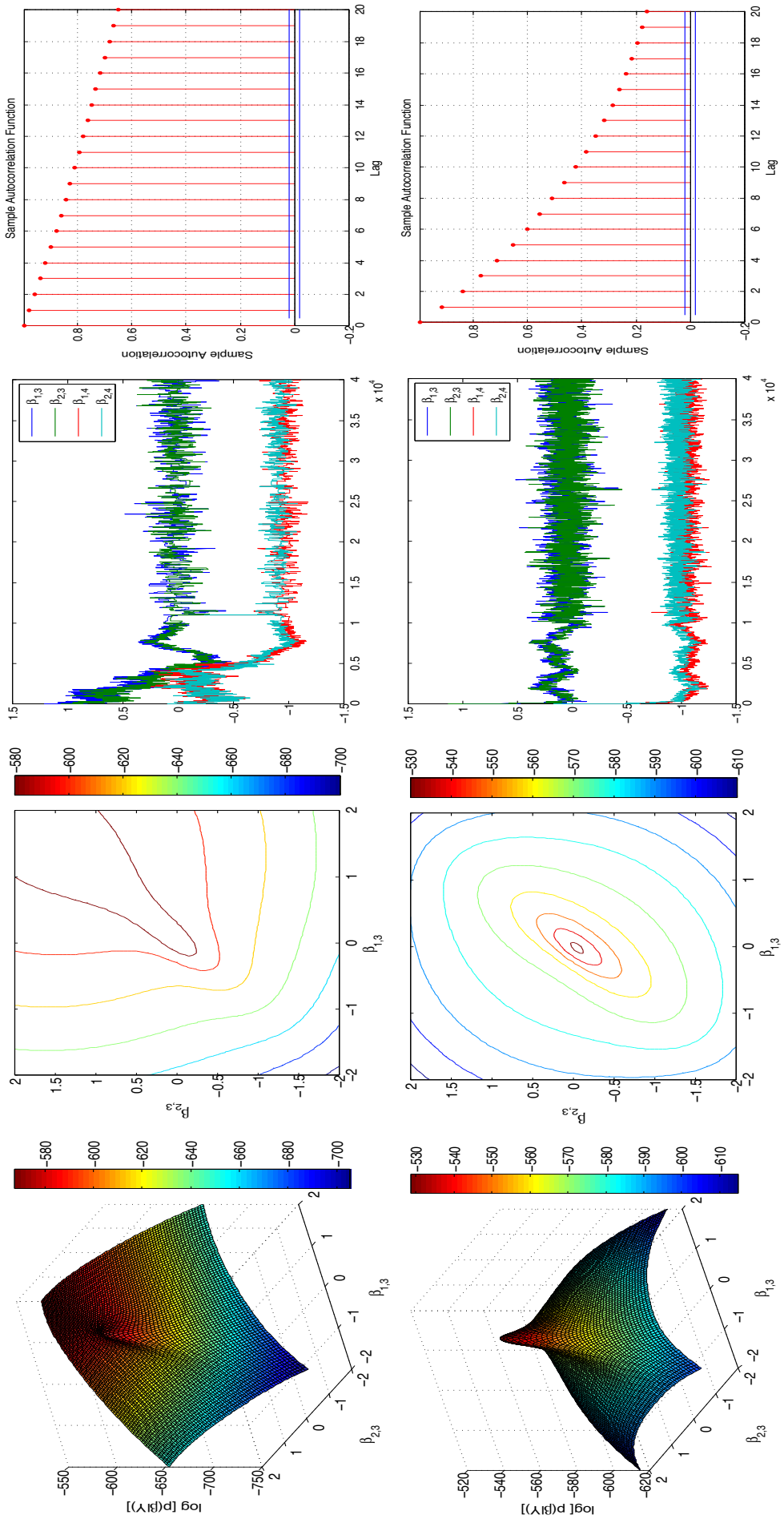


Figure 5.3: Surface and contour plot of $\log[p(\boldsymbol{\beta}^{\star}|\mathbf{Y})]$ as a function of $\boldsymbol{\beta}_{1,3}, \boldsymbol{\beta}_{2,3}$; the prior choice for $\bar{\boldsymbol{\beta}}$ was $[I_r\ \mathbf{0}]'$ (upper row of plots) or $\bar{\boldsymbol{\beta}} = \boldsymbol{\beta}_{\text{Truth}}$ (lower row). The other elements of the $\boldsymbol{\beta}$ vector were set to their known values from equation (5.21) Also plotted are the sample chains produced by the Metropolis algorithm with the adaptive scheme switched on at the 10000^{th} sample, and the Laplace method covariance before that. The rightmost plots are the empirical ACFs of samples (20000:40000) for parameter $\boldsymbol{\beta}_{1,3}$. The acceptance probability was around 5% in the default prior case and 13% in the $\bar{\boldsymbol{\beta}} = \boldsymbol{\beta}_{\text{True}}$ case.

5.5 *Case Study:* Adaptive Metropolis scheme for $\boldsymbol{\beta}$; effect of hyperparameter τ.

The adaptive Metropolis scheme was also tested on the posterior generated after adjusting the prior parameter τ. Small values of τ effectively down-weight the prior, and give increased weight to the data likelihood. The plots in Figure 5.4 show the posterior surface shape for the cases $\tau = 0.1,\ 0.01$, and 0.001 (top to bottom rows).

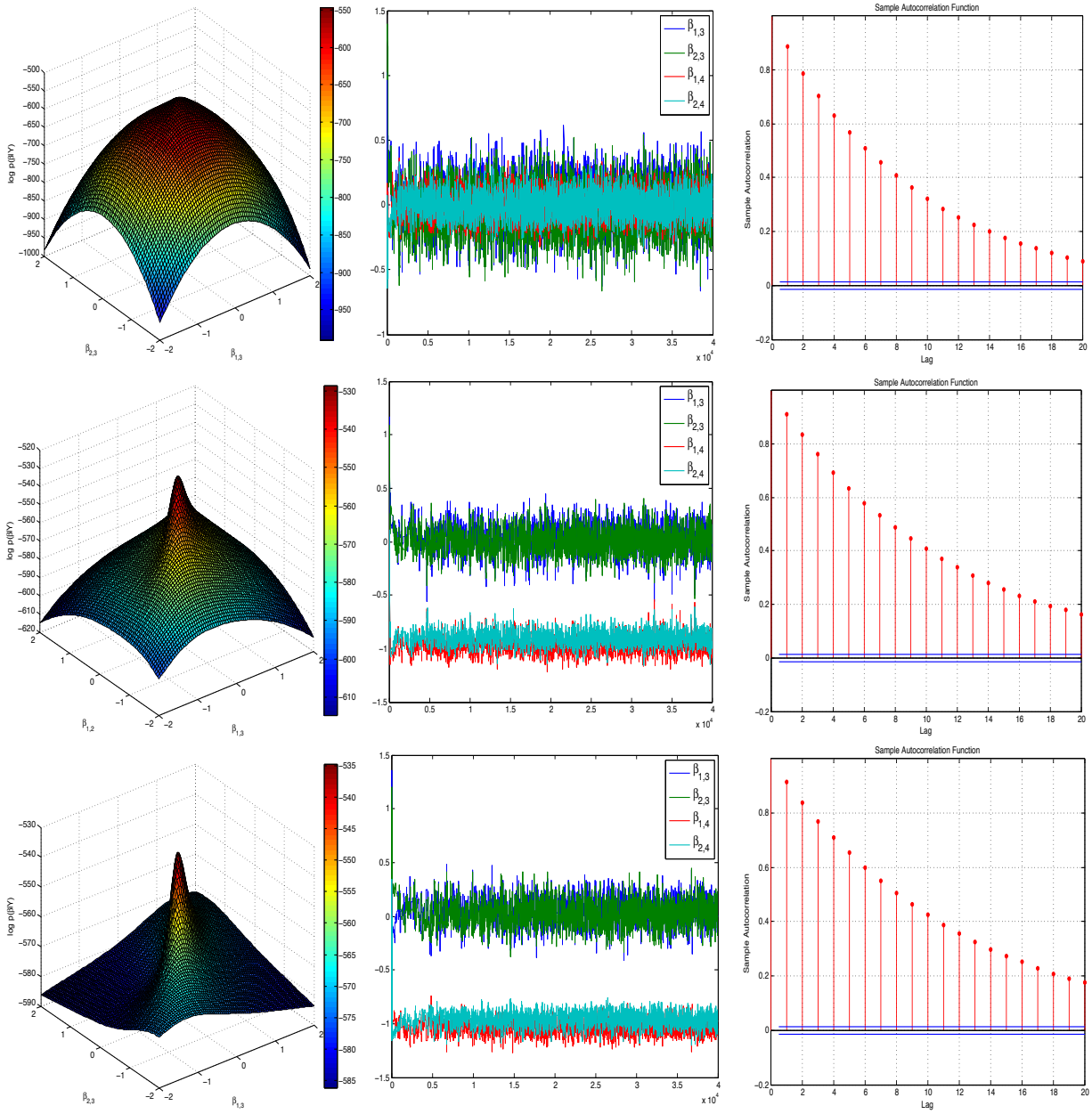


Figure 5.4: Surface and contour plots showing the effect of altering τ on the log posterior $\log p(\boldsymbol{\beta}|Y)$. From top to bottom row: $\tau = 0.1,\ 0.01,\ 0.001$. Dataset size = 100, the initial parameter state was $\boldsymbol{\beta}^{\star} = [1\ \ 1;\ 0\ \ 0]$, and the prior mean was set to $\bar{\boldsymbol{\beta}} = [I_r\ \ \mathbf{0}]'$ throughout. Also shown are the results of running the Adaptive Metropolis algorithm on each surface to produce 40,000 samples, and the corresponding autocorrelation functions (calculated from samples 20,000:40,000). These show that despite the elongated posterior surface shape the AdMCMC algorithm can achieve good convergence.

Weakening the effect of the $\boldsymbol{\beta}$ prior by using smaller values for τ produces a more peaked

posterior surface; this is easier for the the Metropolis algorithm to sample from than the more diffuse surfaces. All three examples have an acceptable decay to their autocorrelation functions, however the first example has failed to converge to the true parameter values for $\boldsymbol{\beta}$.

5.6 Convergence in Practice

The theory of Markov chains guarantees that a Markov chain that is irreducible and has an invariant distribution, f, converges to that distribution. The *ergodic theorems*, for example 5.18 allow us to approximate the desired expectations using the appropriate means:

$$\frac{1}{T}\sum_{t=1}^{T} h(\mathsf{X}^t) \rightarrow \mathbf{E}_f(h(\mathsf{X})) \tag{5.22}$$

using the entire chain; in practice however often a reduced subset of the chain is used. A number of alternatives methods to define that subset are popular:

- **Burn-in**: Depending on the initial starting point for the chain i.e. how $\mathsf{X}^{(0)}$ is chosen, the distribution of $\mathsf{X}^{(1:t)}$ for small t might still be far from the stationary distribution f. Therefore it may be advantageous to discard the first t_0 samples, i.e. $\mathsf{X}^{(0)}, \mathsf{X}^{(1)}, \ldots, \mathsf{X}^{(t_0)}$. The early stage of the sampling process is often termed the *burn-in period*, and the size of this period depends on the speed of mixing in the chain. A burn-in period is illustrated in Figure 5.5 below:

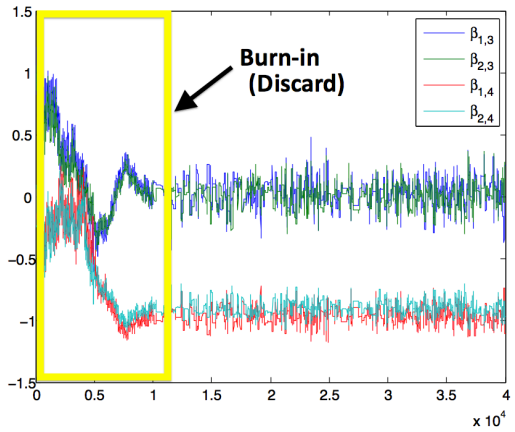


Figure 5.5: The samples in this chain have an obvious burn-in period.

- **Sub-sampling** MCMC methods usually result in chains with positive autocorrelations $\rho(\mathsf{X}^{(t)}, \mathsf{X}^{(t+l)})$ for small l. If these are considered too large *sub-sampling* the chain can reduce them.

5.6.1 Monitoring convergence

The most basic method to monitor the convergence of the chain is to plot the samples produced in order, as in Figure 5.5, where we note that the convergence is *in distribution*, i.e. we do not expect the path itself to converge to a single value.

A key tool for monitoring the convergence of MCMC methods is a plot of the autocorrelation function ρ out to around lag $l = 30$. Ideally we would like to observe a rapid decay in the empirical ACF to around the level of 0.2 by lag $l = 30$; this will indicate that the samples only have a small autocorrelation at this lag. If the ACF is higher than this we may consider sub-sampling.

Various formal tests also exist to monitor and evaluate the convergence of Markov chains, for example non-parametric test such as the *Kolmogorov-Smirnov* test can determine whether an existing chain has already converged (see [Johansen and Evans, 2007] for details). Simple averages can be compared to known values in the case of a synthetic study, and multiple separate chains can also be compared to ensure that they have converged to the same distribution.

5.7 Sequential Monte Carlo

Many scientific problems involve making inferences for the unknown quantities of models which have a sequential, if not explicitly temporal, basis. Since we are in a Bayesian setting a prior distribution is specified over the unknown quantities capturing the prior knowledge of the system, and all the available information from the prior and the data becomes summarized in the posterior. As new observations become available the information that they carry is incorporated into the posterior distribution on-line. In Chapter 4 it was shown that for the case of linear dynamical Gaussian state-space models it is possible to calculate an exact analytical expression for the evolving sequence of posterior distributions via the Kalman filter recursions. A similar recursive method exists for the class of discrete-state Hidden Markov Models (HMMs) which is called the Baum-Welch algorithm [Baum, 1972]. These filters rely on several restricting assumptions to ensure tractability; real data can be much more complex, non-linear, high dimensional and non-Gaussian, and thus preclude an analytic solution.

Sequential Monte Carlo methods are a collection of simulation-based techniques for computing a recursive series of posterior distributions over such complex models. SMC methods are very flexible, relatively easy to implement, parallelisable and applicable in a very wide variety of settings. Since computing power has become so readily available, and due to certain recent advances in applied statistics these methods have recently become a mainstay of advanced research methods in this field.

SMC methods are applicable to continuous-space hidden Markov models of the type that are described by the graphical model in Figure 4.1. An empirical approximation of the distributions of interest is propagated forwards in time using a technique derived from importance sampling. We can view these algorithms as approximating the filtering distribution $p(\mathsf{x}_t|\mathsf{y}_{1:t})$ directly; at each step a weighted empirical distribution is evolved forwards in time according to a specified sampling scheme. In the SMC framework the posterior distribution is empirically represented by a weighted sum of N samples, termed *particles* as follows:

$$p_t(d\mathsf{X}_{1:t}|\mathsf{Y}_{1:t}) \approx \frac{1}{N}\sum_{i=1} w_t^i \delta_{\mathsf{x}_{1:t}^i}(d\mathsf{X}_{1:t}),$$

where

N	is the number of independent samples,
w_t^i	is the weight of sample i at time t,
X_t^i	are iid samples drawn from $p_t(\mathsf{X}_{1:t}\|\mathsf{Y}_{1:t})$ at time t,
$\delta_{\mathsf{x}_0}(\mathsf{x})$	indicates the dirac delta mass located at x_0

Note: in this section when we discuss particles and write X^i, this indicates the set of particles $\mathsf{X}^{i=1:N}$.

SMC methods sample sequentially from the sequence of probability densities $\{p_t(\mathsf{x}_{1:t})\}$ of increasing dimension, where

$$p_t(\mathsf{x}_{1:t}) = \frac{\widetilde{p}_t(\mathsf{x}_{1:t})}{Z_t}, \tag{5.23}$$

and $\widetilde{p}$ is known pointwise. SMC methods provide an approximation of $p_1(\mathsf{x}_1)$ and an estimate of Z_1 at time 1, then an approximation of $p_{1:2}(\mathsf{x}_{1:2})$ and an estimate of Z_2 at time 2 etc. The basic categories of SMC filters are described in the proceeding sections.

5.7.1 Sequential Importance Sampling

The simple form of importance sampling already discussed is not applicable to the problem of recursive estimation. It is necessary to have all the observations, $\mathsf{x}_{1:t}$, before estimating $p(\mathsf{x}_{1:t}|\mathsf{y}_{1:t})$.

In general, as each new data point y_{t+1} becomes available it is necessary to recompute the importance weights over the entire state sequence; the computational complexity of this operation therefore increases with time. *Sequential Importance Sampling* (SIS) is a technique designed to overcome this problem and to offer a solution for recursive estimation with fixed computational cost.

The solution depends upon choosing an importance function with the following structure

$$\begin{aligned} q_k(\mathsf{x}_{1:t}) =& q_{t-1}(\mathsf{x}_{1:t-1}) q_t(\mathsf{x}_t|\mathsf{x}_{1:t-1}) \\ =& q_1 \prod_{k=2}^{t} q_k(\mathsf{x}_k|\mathsf{x}_{1:k-1}). \end{aligned} \tag{5.24}$$

To obtain the particles $\mathsf{X}^i_{1:t} \sim q_t(\mathsf{x}_{1:t})$ at time t, we sample $\mathsf{X}^i_2 \sim q_1(\mathsf{x}_1)$ at time $t = 1$, then $\mathsf{X}^i_k \sim q_k(\mathsf{x}_k|\mathsf{X}^i_{1:k-1})$ at times $k = 2, \ldots, t$. The corresponding unnormalized weights can be computed using

$$\begin{aligned} w_t(\mathsf{x}_{1:t}) &= \frac{\widetilde{p}_t(\mathsf{x}_{1:t})}{q_t(\mathsf{x}_{1:t})} \\ &= \frac{\widetilde{p}_{t-1}(\mathsf{x}_{1:t-1})}{q_{t-1}(\mathsf{x}_{1:t-1})} \frac{\widetilde{p}_t(\mathsf{x}_{1:t})}{\widetilde{p}_{t-1}(\mathsf{x}_{1:t-1}) q_t(\mathsf{x}_t)} \end{aligned}$$

which can be written as

$$\begin{aligned} w_t(\mathsf{x}_{1:t}) &= w_{t-1}(\mathsf{x}_{1:t-1}) \cdot \alpha_t(\mathsf{x}_{1:t}) \\ &= w_1(\mathsf{x}_1) \prod_{k=2}^{t} \alpha_k(\mathsf{x}_{1:k}), \end{aligned}$$

where the *incremental weight function* $\alpha_t(\mathsf{x}_{1:t})$ is defined as

$$\alpha_t(\mathsf{x}_{1:t}) = \frac{\widetilde{p}_t(\mathsf{x}_{1:t})}{\widetilde{p}_{t-1}(\mathsf{x}_{1:t-1}) q_t(\mathsf{x}_t)}.$$

[Doucet and Johansen, 2011].

Algorithm 9: Sequential Importance Sampling Algorithm

begin

At time $t = 1$:

1. Sample $\mathsf{X}^i_1 \sim q_1(\mathsf{x}_1)$
2. Compute the weights $w_1(\mathsf{X}^i_1)$ and the normalised weights $\widetilde{w}_1(\mathsf{X}^i_1) = \frac{w_1(\mathsf{X}^i_1)}{\sum_{i=1}^N w_1(\mathsf{X}^i_1)}$

for *times* $t = 2$ *to* T **do**

3. Sample $\mathsf{X}^i_t \sim q_t(\mathsf{x}_t|\mathsf{X}^i_{1:t-1})$
4. Compute the weights $w_t(\mathsf{X}^i_{1:t}) = w_{t-1}(\mathsf{X}^i_{1:t-1}) \cdot \alpha_t(\mathsf{X}^i_{1:t})$ and the normalised weights $\widetilde{w}_t(\mathsf{X}^i_t) = \frac{w_t(\mathsf{X}^i_{1:t})}{\sum_{i=1}^N w_1(\mathsf{X}^i_{1:t})}$

At any time t we can obtain the approximations to $\widehat{p}(\mathsf{x}_{1:t})$ and Z_t from

$$\widehat{p}_t(\mathsf{x}_{1:t}) = \sum_{i=1}^{N} \widetilde{w}^i_t \delta_{\mathsf{X}^i_{1:t}}(\mathsf{x}_{1:t}), \qquad \widehat{Z_t} = \frac{1}{N} \sum_{i=1}^{N} w_t(\mathsf{X}^i_{1:t}). \tag{5.25}$$

Despite the attractive recursive structure of the SIS algorithm it is well known to suffer from some serious problems. The *degeneracy problem* occurs, when, as t increases the distribution of the importance weights becomes more and more severely skewed; in practice after a few time steps only one of the particles will possess a non-zero importance weight. Also, the variance of the estimate can be shown to increase exponentially with t, see [Doucet and Johansen, 2011].

5.7.2 Resampling

Resampling is a technique for removing particles with a low weight with high probability; this is important in the sequential framework since we would like to concentrate computational effort on areas of the distribution with high mass. Resampling also allows us (in some scenarios) to solve the problem of the exponential increase in variance mentioned above. Consider approximating the distribution of interest, $p(\mathsf{x}_{1:t})$, using the importance sampling approximation, $\widehat{p}(\mathsf{x}_{1:t})$. This is based on weighted samples from $q_t(\mathsf{x}_{1:t})$, and does not provide samples from $p(\mathsf{x}_{1:t})$. However, to obtain samples from $p(\mathsf{x}_{1:\mathrm{t}})$ samples can be drawn from the importance sampling approximation itself $\widehat{p}(\mathsf{x}_{1:t})$; i.e. we select $\mathsf{X}^i_{i:t}$ with probability $\widetilde{w}^i_t$. This operation is termed resampling; if we require N samples we simply sample N times from $\widehat{p}(\mathsf{x}_{1:t})$. Equivalently this can be viewed as associating a number N^i_t offspring with each particle $\mathsf{X}^i_{1:t}$ such that $N^{1:N}_t = \left\{N^1_t, N^2_t, \ldots, N^N_t\right\}$ has a multinomial distribution $N^{1:N}_t \sim \text{Multinomial}(N, \widetilde{w}^{1:N}_t)$, and a weight for each offspring particle $1/N$.

Now, a second approximation to $p(\mathsf{x}^i_{1:t})$ is available

$$\overline{p}(\mathsf{x}_{1:t}) = \sum_{i=1}^{N} \frac{N^i_t}{N} \delta_{\mathsf{X}^i_{1:t}}(\mathsf{x}_{1:t}),$$

and the resampling scheme that we choose should satisfy the unbiasedness property:

$$\mathbb{E}[\overline{p}(\mathsf{x}_{1:t}|\mathsf{y}_{1:t})] = \widehat{p}(\mathsf{x}_{1:t}|\mathsf{y}_{1:t}). \tag{5.26}$$

The multinomial scheme mentioned above satisfies this property since $\mathbb{E}[N^i_t|\widetilde{w}^{1:N}_t] = N\widetilde{w}^i_t$. Resampling alleviates degeneracy, but can introduce extra noise and also the problem of *sample impoverishment*. Sample impoverishment occurs when samples with high importance weights are repeatedly selected, which leads to loss of diversity amongst the samples. Several improved resampling schemes have been suggested; these must select N^i_t such that the unbiasedness property (5.26) is preserved, and such that $\text{Var}[N^i_t|\widetilde{w}^{I:N}_t]$ is smaller than that achieved by the multinomial scheme [Doucet and Johansen, 2011].

Residual Resampling

The *residual resampling* scheme is closely related to splitting up the empirical CDF up into N components and ensuring a sample is drawn once from each (this is called the stratified approach, and was introduced in [Carpenter et al., 1999]).

Algorithm 10: Residual Resampling Scheme

1. Set $\widetilde{N}^i_t = \lfloor N\widetilde{w}^i_t \rfloor$
2. Sample $\overline{N}^i_t \sim \text{Multinomial}(N, \overline{w}^{1:N}_t)$, where $\overline{w}^i_t \propto \widetilde{w}^i_t - \frac{\widetilde{N}^i_t}{N}$
3. Set $N^i_t = \widetilde{N}^i_t + \overline{N}^i_t$.

Systematic Resampling

The *systematic resampling* scheme has been shown to have the lowest variance of the aforementioned schemes [Douc and Cappe, 2005] and extends the stratified type approach. In the systematic scheme the random variables drawn from each 'strata' are deterministically linked. The scheme proceeds as:

Algorithm 11: Systematic Resampling Scheme

1. Sample $U_1 \sim U[0, \frac{1}{N}]$
2. Define $U_i = U_1 = \frac{i-1}{N}$ for $i = 2, \ldots, N$
3. Set $N^i_t = \left|\left\{U_j : \sum_{k=1}^{i-1} \widetilde{w}^k_t \leq U_j \leq \sum_{k=1}^{i} \widetilde{w}^k_t\right\}\right|$

The samples produced can be shown to be unbiased, but not independent [Künsch, 2005].

5.7.3 Sequential Importance Sampling Resampling (SISR)

In general, SMC methods combine sequential importance sampling and resampling. The *sequential importance sampling resampling* algorithm is the most basic full SMC method. At time $t = 1$ we calculate the importance sampling approximation $\widehat{p}(\mathsf{x}_1)$, which consists of a weighted collection of particles $\left\{\widetilde{w}_1^i, \mathsf{X}_1^i\right\}$. The resampling step is used to eliminate (with high probability) those particles having low weights. After resampling the distribution is represented by an equally weighted collection of particles $\left\{\frac{1}{N}, \overline{\mathsf{X}}_1^i\right\}$. At time $t = 2$ we sample $\mathsf{X}_2^i \sim q_2(\mathsf{x}_2|\overline{\mathsf{X}}_1^i)$, and thus $(\overline{\mathsf{X}}_1^i, \mathsf{X}_2^i)$ is now approximately distributed according to $p_1(\mathsf{x}_1)q_2(\mathsf{x}_2|\mathsf{x}_1)$ (therefore the corresponding importance weights are simply equal to the incremental weights $\alpha_2(\mathsf{x}_{1:2})$). The algorithm continues by resampling these particles with respect to those normalized weights etc. and is listed as Algorithm 12 below, and depicted graphically in Figure 5.6.

Algorithm 12: Sequential Importance Sampling Resampling Algorithm

begin

At time $t = 1$:

1. Sample $\mathsf{X}_1^i \sim q_1(\mathsf{x}_1)$
2. Compute the weights $w_1(\mathsf{X}_1^i)$ and the normalised weights $\widetilde{w}_1(\mathsf{X}_1^i) = \frac{w_1(\mathsf{X}_1^i)}{\sum_{i=1}^N w_1(\mathsf{X}_1^i)}$
3. Resample $\left\{\widetilde{w}_1^i, \mathsf{X}_1^i\right\}$ to obtain N equally weighted particles $\left\{\frac{1}{N}, \overline{\mathsf{X}}_1^i\right\}$

for *times* $t = 2$ *to* T **do**

4. Sample $\mathsf{X}_t^i \sim q_t(\mathsf{x}_t|\overline{\mathsf{X}}_{1:t-1}^i)$, and set $\mathsf{X}_{1:t}^i \leftarrow \left(\overline{\mathsf{X}}_{1:t-1}^i, \mathsf{X}_t^i\right)$
5. Compute the weights $\alpha_t(\mathsf{X}_{1:t}^i)$ and the normalised weights $\widetilde{w}_t(\mathsf{X}_t^i) = \frac{\alpha_t(\mathsf{X}_{1:t}^i)}{\sum_{i=1}^N \alpha_t(\mathsf{X}_{1:t}^i)}$
6. Resample $\left\{\widetilde{w}_t^i, \mathsf{X}_{1:t}^i\right\}$ to obtain N equally weighted particles $\left\{\frac{1}{N}, \overline{\mathsf{X}}_{1:t}^i\right\}$

At each time, t, the algorithm provides two approximations to $p_t(\mathsf{x}_{1:t})$, either $\widehat{p}_t(\mathsf{x}_{1:t})$ after the sampling step, or $\overline{p}_t(\mathsf{x}_{1:t})$ after the resampling step:

$$\widehat{p}_t(\mathsf{x}_{1:t}) = \sum_{i=1}^N \widetilde{w}_t^i \delta_{\mathsf{X}_{1:t}^i}(\mathsf{x}_{1:t}), \qquad \overline{p}_t(\mathsf{x}_{1:t}) = \frac{1}{N}\sum_{i=1}^N \delta_{\overline{\mathsf{X}}_{1:t}^i}(\mathsf{x}_{1:t}), \tag{5.27}$$

the first of which is to be preferred [Doucet and Johansen, 2011].

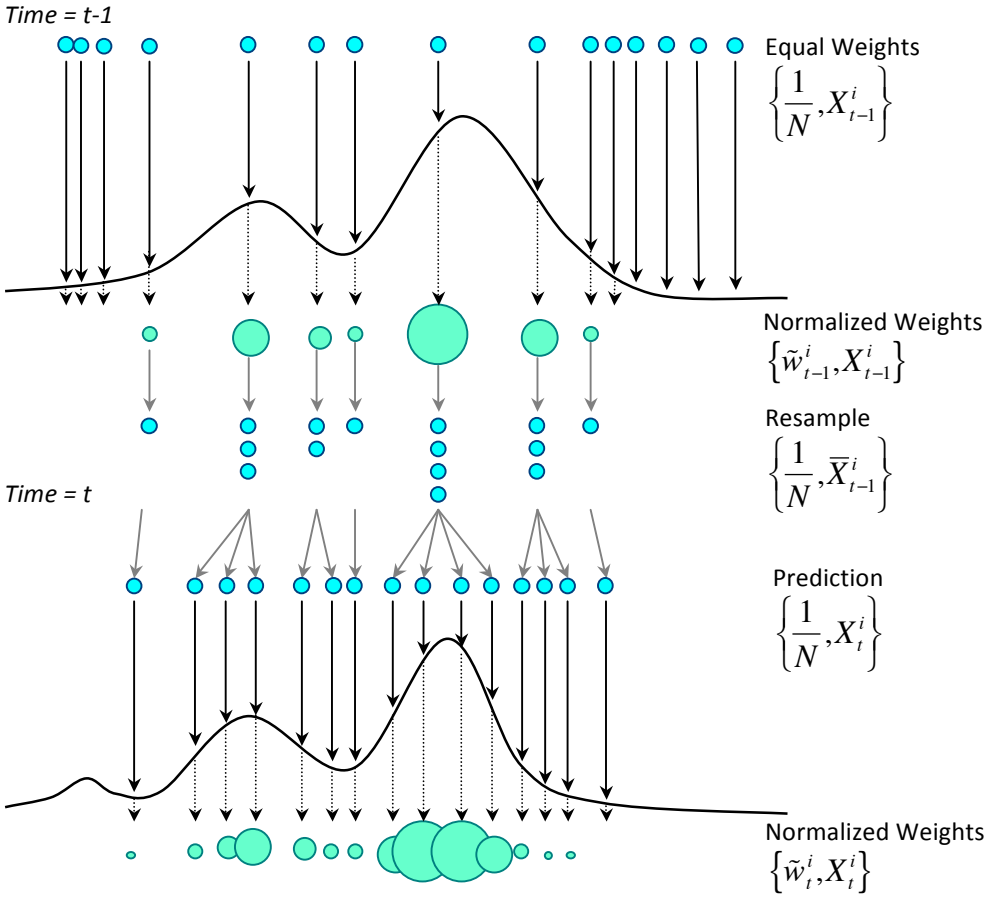


Figure 5.6: The steps of the SISR algorithm: at time $t-1$ an equally weighted set of particles approximates $p_{t-1}(\mathsf{x}_{t-1})$, importance weights are calculated and normalized based on the information at $t-1$ resulting in the weighted samples $\left\{\widetilde{w}^i_{t-1}, \mathsf{X}^i_{t-1}\right\}$. Resampling selects the most likely particles to give the equally weighted approximation $\left\{\frac{1}{N}, \overline{\mathsf{X}}^i_{t-1}\right\}$. The sampling/prediction step at time t evolves these, introducing variety, resulting in $\left\{\widetilde{w}^i_t, \mathsf{X}^i_{1:t}\right\}$ which is an approximation to $p_t(\mathsf{x}_t)$.

As mentioned above resampling also has the effect of introducing some extra variance, therefore it is prudent to only perform the resampling step in practice if the variance of the unnormalized weights is above some pre-define threshold N_R. The *effective sample size* is a tool to measure the variability of the weights, and an estimator for it (which takes values between 1 and N) is given by:

$$\text{ESS} = \left(\sum_{i=1}^{N} (\widetilde{w}^i_t)^2\right)^{-1},$$

5.7.4 SMC for Filtering

Before discussing PMCMC algorithms for jointly estimating latent states and static parameters this section explains the use of SMC methods in the context of sequential filtering for the models defined in (5.1). Here we are interested in approximating the series of distributions $\{p(\mathsf{x}_{1:t}|\mathsf{y}_{1:t})\}$ for $t = 1, \ldots T$. Assume for the moment that the static parameters θ are known. Given observations $\mathsf{y}_{1:t}$, inference about the state $\mathsf{x}_{1:t}$ is given by the posterior:

$$p_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t}) = \frac{p_\theta(\mathsf{x}_{1:t}, \mathsf{y}_{1:t})}{p_\theta(\mathsf{y}_{1:t})},$$

where the joint distribution of latent and observed states is given by

$$p_\theta(\mathsf{x}_{1:t}, \mathsf{y}_{1:t}) = \mu_\theta(\mathsf{x}_1) \prod_{k=2}^{t} f_\theta(\mathsf{x}_k|\mathsf{x}_{k-1}) \prod_{k=2}^{t} g_\theta(\mathsf{y}_k|\mathsf{x}_k),$$

the marginal likelihood is given by $p_\theta(\mathsf{y}_{1:\mathsf{t}}) = \int p_\theta(\mathsf{x}_{1:t}, \mathsf{y}_{1:t}) d\mathsf{x}_{1:t}$. Also

$$p_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t}) = p_\theta(\mathsf{x}_{1:t-1}|\mathsf{y}_{1:t-1}) \frac{f_\theta(\mathsf{x}_t|\mathsf{x}_{t-1}) g_\theta(\mathsf{y}_t|\mathsf{x}_t)}{p_\theta(\mathsf{y}_t|\mathsf{y}_{1:t-1})},$$

and $p_\theta(\mathsf{y}_\mathsf{t}|\mathsf{y}_{1:\mathsf{t}-1}) = \int f_\theta(\mathsf{x}_t|\mathsf{x}_{t-1}) g_\theta(\mathsf{y}_t|\mathsf{x}_t) p_\theta(\mathsf{x}_{t-1}|\mathsf{y}_{1:t-1}) d\mathsf{x}_{t-1:t}$. The SMC filtering algorithm relies on the introduction of the importance density $q_\theta(\mathsf{x}_1|\mathsf{y}_1)$ at time 1, and $q_\theta(\mathsf{x}_t|\mathsf{y}_t, \mathsf{x}_{t-1})$ at times $t \geq 1$. The default choice is made by using $q_\theta(\mathsf{x}_1) = \mu_\theta(\mathsf{x}_1)$ and $q_\theta(\mathsf{x}_t|\mathsf{y}_t, \mathsf{x}_{t-1}) = f_\theta(\mathsf{x}_t|\mathsf{x}_{t-1})$. Optimal choices are given by $p_\theta(\mathsf{x}_1|\mathsf{y}_1)$ and $q_\theta(\mathsf{x}_t|\mathsf{y}_t, \mathsf{x}_{t-1})$ [Doucet et al., 2009]. The SMC Filtering algorithm is presented as Algorithm 13.

Algorithm 13: Sequential Monte Carlo for Filtering

Assuming we know the parameter vector θ:

begin

At time $t = 1$:

1. Sample $\mathsf{X}_1^i \sim q_1(\mathsf{x}_1)$
2. Compute the weights $w_1(\mathsf{X}_1^i) = \frac{\mu_\theta(\mathsf{X}_1^i) g_\theta(\mathsf{Y}_1^i|\mathsf{X}_1^i)}{q_\theta(\mathsf{X}_1^i|\mathsf{Y}_1^i)}$ and normalised weights $\widetilde{w}_t(\mathsf{X}_t^i)$
3. Resample $\left\{\widetilde{w}_1^i, \mathsf{X}_1^i\right\}$ to obtain N equally weighted particles $\left\{\frac{1}{N}, \overline{\mathsf{X}}_1^i\right\}$

for *times* $t = 2$ *to* T **do**

4. Sample $\mathsf{X}_t^i \sim q_t(\mathsf{x}_t|\overline{\mathsf{X}}_{1:t-1}^i)$, and set $\mathsf{X}_{1:t}^i \leftarrow \left(\overline{\mathsf{X}}_{1:t-1}^i, \mathsf{X}_t^i\right)$
5. Compute the weights $w_t(\mathsf{X}_{t-1:t}^i) = \frac{f_\theta(\mathsf{X}_t^i|\mathsf{X}_{t-1}^i) g_\theta(\mathsf{Y}_t^i|\mathsf{X}_t^i)}{q_\theta(\mathsf{X}_t^i|Y_t^i, \mathsf{X}_{t-1}^i)}$, and normalised weights $\widetilde{w}_t(\mathsf{X}_t^i)$
6. Resample $\left\{\widetilde{w}_t^i, \mathsf{X}_{1:t}^i\right\}$ to obtain N equally weighted particles $\left\{\frac{1}{N}, \overline{\mathsf{X}}_{1:t}^i\right\}$

At time t we can obtain approximations to $\widehat{p}_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t})$ (before resampling) or $\overline{p}_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t})$ (after resampling) from

$$\widehat{p}_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t}) = \sum_{i=1}^{N} \widetilde{w}_t^i \delta_{\mathsf{X}_{1:t}^i}(\mathsf{x}_{1:t}), \qquad \overline{p}_\theta(\mathsf{x}_{1:t}|\mathsf{y}_{1:t}) = \frac{1}{N} \sum_{i=1}^{N} \delta_{\overline{\mathsf{X}}_{1:t}^i}(\mathsf{x}_{1:t}). \tag{5.28}$$

Finally the marginal likelihood may be estimated via $\widehat{p}_\theta(\mathsf{y}_{1:t}) = \widehat{p}_\theta(\mathsf{y}_1) \prod_{k=2}^{T} \widehat{p}_\theta(\mathsf{y}_k|\mathsf{y}_{1:k-1})$.

5.8 PMCMC Algorithms

As discussed the standard approach in statistics to sample from distributions for which analytic forms are unavailable is MCMC, for example we could use MCMC to approximate $p(\mathsf{x}_{1:t}, \theta|\mathsf{y}_{1:t})$. Unfortunately it is very difficult to design effective proposal distributions for non-linear non-Gaussian state space models. *Particle Markov Chain Monte Carlo* (PMCMC) methods are a recently developed class of MCMC techniques which use embedded SMC algorithms to create efficient high-dimensional proposal distributions. A popular example of such PMCMC algorithms is the *Particle Marginal Metropolis Hastings* algorithm, which is described in the following section.

5.8.1 The PMMH Algorithm

The *Particle Marginal Metropolis Hastings* (PMMH) algorithm is an approximation of an ideal Marginal Metropolis-Hastings sampler for sampling from $p(\mathsf{x}_{1:T}, \theta|\mathsf{y}_{1:T})$ (static parameter vector $\theta = \mathbf{\Theta}_{\mathsf{M}_0}$ for model M_0 for example). The standard marginal Metropolis-Hastings algorithm has acceptance probability given by:

$$\begin{aligned}
&\min\left\{1, \frac{p\big(\mathsf{x}^*_{1:T}, \theta^*|\mathsf{y}_{1:T}\big)q\big((\mathsf{x}_{1:T}, \theta)|(\mathsf{x}^*_{1:T}, \theta^*)\big)}{p\big(\mathsf{x}_{1:T}, \theta|\mathsf{y}_{1:T}\big)q\big((\mathsf{x}^*_{1:T}, \theta^*)|(\mathsf{x}_{1:T}, \theta)\big)}\right\} \\
=&\min\left\{1, \frac{p\big(\mathsf{x}^*_{1:T}, \theta^*|\mathsf{y}_{1:T}\big)q(\theta|\theta^*)p_\theta(\mathsf{x}_{1:T}|\mathsf{y}_{1:T})}{p\big(\mathsf{x}_{1:T}, \theta|\mathsf{y}_{1:T}\big)q(\theta^*|\theta)p_{\theta^*}(\mathsf{x}^*_{1:T}|\mathsf{y}_{1:T})}\right\} \\
=&\min\left\{1, \frac{p_{\theta^*}(\mathsf{y}_{1:T})p(\theta^*)q(\theta|\theta^*)}{p_\theta(\mathsf{y}_{1:T})p(\theta)q(\theta^*|\theta)}\right\}
\end{aligned}$$

Ideally the samper would use the following proposal density:

$$q\big((\mathsf{x}^*_{1:T}, \theta^*)|(\mathsf{x}_{1:T}, \theta)\big) = q(\theta^*|\theta)p_{\theta^*}(\mathsf{x}^*_{1:T}|\mathsf{y}_{1:T}), \tag{5.29}$$

to propose moves in the joint parameter/latent space, where $q(\theta^*|\theta)$ is some proposal density for a move in the static parameter space to candidate θ^* when we are at location θ, and $p_{\theta^*}(\mathsf{x}^*_{1:T}|\mathsf{y}_{1:T})$ is the conditional density of the latent states $\mathsf{x}^*_{1:T}$ given the observations and the proposed parameters θ^*. Unfortunately, in practice, this ideal algorithm cannot be implemented since we can neither sample exactly from $p_{\theta^*}(\mathsf{x}^*_{1:T}|\mathsf{y}_{1:T})$, nor can we compute the terms $p_{\theta^*}(\mathsf{y}_{1:T})$ and $p_\theta(\mathsf{y}_{1:T})$ which appear in the acceptance ratio. The PMMH sampler approximates this ideal sampler by using SMC approximations of the unknown terms; Algorithm 14 is the general version, and a specific version for joint estimation of parameters $\{\boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{B}\}$ and latent states $\mathbf{z}_{1:T}$ in Bayesian CVAR model $\mathbf{M}_0$ is given as Algorithm 15.

Remark *More complex versions of PMCMC algorithms exist; for example there is a Particle Gibbs version of the algorithms first described in [Andrieu et al., 2010]. This samples iteratively from* $p(\theta|\mathsf{y}_{1:T}, \mathsf{x}_{1:T})$ *and* $p_\theta(\mathsf{x}_{1:T}|\mathsf{y}_{1:T})$*. To create a valid particle approximation to the Gibbs sampler requires a special type of* conditional *SMC update which ensures that a prespecified path* $\mathsf{X}_{1:T}$*, with known ancestral lineage, is guaranteed to survive all the resampling steps. This increases the algorithmic complexity, particularly in the matrix variate setting.*

Algorithm 14: PMMH sampler for static parameters $\boldsymbol{\theta}$ and latent states $\mathbf{z}_{1:T}$

---

if *iteration* $k = 0$ **then**

1. Set statatic parameter vector $\boldsymbol{\theta}(0)$ arbitrarily.
2. Run an SMC algorithm targeting $p_{\boldsymbol{\theta}(0)}(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$, sample $\mathbf{x}_{1:T}(0) \sim \hat{p}_{\boldsymbol{\theta}(0)}(d\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$

2a. Compute the marginal likelihood estimate $\hat{p}_{\boldsymbol{\theta}(0)}(\mathbf{y}_{1:T})$

end

for *iterations* $k \geq 1$ **do**

3. Sample a proposal $\boldsymbol{\theta}^* \sim q(\boldsymbol{\theta}^*|\boldsymbol{\theta})$
4. Run an SMC algorithm targeting $p_{\boldsymbol{\theta}^*}(\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$, sample $\mathbf{x}^*{}_{1:T} \sim \hat{p}_{\boldsymbol{\theta}^*}(d\mathbf{x}_{1:T}|\mathbf{y}_{1:T})$

4b. Compute the marginal likelihood estimate $\hat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})$

5. Calculate the Metropolis Hastings Acceptance Probability:

$$A(\boldsymbol{\theta}^{(k-1)}, \boldsymbol{\theta}^*) = \min\left\{1, \frac{p_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})p(\boldsymbol{\theta}^*)q\big(\boldsymbol{\theta}(k-1)|\boldsymbol{\theta}^*\big)}{p_{\boldsymbol{\theta}(k-1)}(\mathbf{y}_{1:T})p\big(\boldsymbol{\theta}(k-1)\big)q(\boldsymbol{\theta}^*|\boldsymbol{\theta}(k-1))}\right\} \tag{5.30}$$

Set $\boldsymbol{\theta}(k) = \boldsymbol{\theta}^*$, $\mathbf{x}_{1:T}(k) = \mathbf{x}^*{}_{1:T}$ and $\hat{p}_{\boldsymbol{\theta}}(k)(\mathbf{y}_{1:\mathbf{T}}) = \hat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})$. Otherwise set $\boldsymbol{\theta}(k) = \boldsymbol{\theta}(k-1)$, $\mathbf{x}_{1:T}(k) = \mathbf{x}_{1:T}(k-1)$ and $\hat{p}_{\boldsymbol{\theta}(k)}(\mathbf{y}_{1:T}) = \hat{p}_{\boldsymbol{\theta}(k-1)}(\mathbf{y}_{1:T})$.

end

5.8.2 The PMMH algorithm in the CVAR context

We conclude this section by listing the key components of the complete PMMH algorithm for the joint estimation of $\boldsymbol{\beta}, \Sigma, B$ and $\mathsf{z}_{1:T}$, as described over the earlier parts of this Chapter:

- An adaptive method to learn, online, an appropriate covariance matrix for use in a multivariate normal distribution to generate proposed states for the cointegrating vector $\boldsymbol{\beta}$ on each iteration of the PMCMC algorithm.
- A SISR filter which sequentially evaluates the filtering distribution of the latent states $p_\beta(\mathsf{z}_{1:T}|\mathsf{y}_{1:T})$, and is also capable of providing an estimate of the marginal likelihood $p_\beta(\mathsf{y}_{1:T})$. The filter incorporates sequential importance sampling and one of a number of re-sampling algorithms as described earlier.
- A suitable acceptance ratio to use with a rejection sampling technique to accept/reject moves proposed by the algorithm; such a ratio is given in equation (5.30).
- For estimation in a Bayesian setting we also require components to calculate the prior densities based on the conjugate analysis presented in Chapter 3; these need to be included in the acceptance ratio of equation (5.30).
- If we are interested in also estimating Σ and B we will also require a component to sample 'exactly' from the known posterior distributions $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$.

The complete specific version for joint estimation of parameters $\{\boldsymbol{\beta}, \boldsymbol{\Sigma}, \boldsymbol{B}\}$ and latent states $\mathbf{z}_{1:T}$ in Bayesian CVAR model $\mathbf{M}_0$ is given as Algorithm 15.

Remark *To avoid confusion with the earlier notation for the observations ($\Delta\mathbf{x}_t$) in Bayesian CVAR models we will use $\mathbf{z}_{1:T}$ to represent the latent states when discussing concrete applications of the algorithms.*

Algorithm 15: PMMH sampler for parameters $\boldsymbol{\beta}$, $\boldsymbol{\Sigma}$, $\boldsymbol{B}$ and latent states $\mathbf{z}_{1:T}$ (Model $\mathbf{M}_0$)

Input: Initial parameter states $\boldsymbol{\theta}^{(0)} = \left\{\boldsymbol{\beta}^{(0)}, \boldsymbol{\Sigma}^{(0)}, \boldsymbol{B}^{(0)}\right\}$, observations $\mathbf{y}_{1:T}$, priors $\pi(\boldsymbol{\beta}), \pi(\boldsymbol{\Sigma})$, $\pi(\boldsymbol{B})$.

Output: Parameter estimates $\left\{\boldsymbol{\theta}^{(t)}\right\} = \left\{\boldsymbol{\beta}^{(t)}, \boldsymbol{\Sigma}^{(t)}, \boldsymbol{B}^{(t)}\right\}, t = 1, \ldots, T$, and latent states $\mathbf{z}_{1:T}$.

begin

if *iteration* $k = 0$ **then**

1. Set $\boldsymbol{\theta}(0) = \boldsymbol{\theta}^{(0)}$

2. Run an SMC algorithm targeting $p_{\boldsymbol{\theta}(0)}(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$
   Using:
   - Prior: $p(Z_0)=\delta(Z) = Z_0$
   - Transition Proposal: $p(Z_t|Z_{t-1}) = \text{MVN}(Z_t|b_{\boldsymbol{\theta}(0)}(Z_{t-1}), \Sigma_w)$
   - Observation Likelihood: $p(Y_t|Z_t) = \text{MVN}(Y_t|c_{\boldsymbol{\theta}(0)}(Z_t), \Sigma)$

   where $b_{\boldsymbol{\theta}}$ and $c_{\boldsymbol{\theta}}$ are functions dependent upon the parameters of the model.

2a. Sample $\mathbf{z}_{1:T}(0) \sim \hat{p}_{\boldsymbol{\theta}(0)}(d\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$

2b. Compute the marginal likelihood estimate: $\widehat{p}_{\boldsymbol{\theta}(0)}(\mathbf{y}_{1:T}) = \widehat{p}_{\boldsymbol{\theta}}(\mathbf{y}_1)\prod_{t=2}^{T}\widehat{p}_{\boldsymbol{\theta}}(\mathbf{y}_t|\mathbf{y}_{t-1})$.

end

for *iteration* $k = 2$ *to* T **do**

3. Sample candidate parameter $\boldsymbol{\beta}^*$ from proposal: $\boldsymbol{\beta}^* \sim q(\boldsymbol{\beta}^*|\boldsymbol{\beta}) = \mathcal{N}(\boldsymbol{\beta}^*|\boldsymbol{\beta}, \Omega)$

3a. Sample $\boldsymbol{\Sigma}^* \sim (\cdot|\boldsymbol{\beta}^*, Y)$ and $\boldsymbol{B}^* \sim (\cdot|\boldsymbol{\beta}^*, Y)$ exactly from conjugate posteriors.

3b. Form the proposed state: $\boldsymbol{\theta}^* = \{\boldsymbol{\beta}^*, \boldsymbol{\Sigma}^*, \boldsymbol{B}^*\}$

4. Run an SMC algorithm targeting $p_{\boldsymbol{\theta}^*}(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$
   Using:
   - Prior: $p(Z_0)=\delta(Z) = Z_0$
   - Transition Proposal: $p(Z_t|Z_{t-1}) = \text{MVN}(Z_t|b_{\boldsymbol{\theta}^*}(Z_{t-1}), \Sigma_w)$
   - Observation Likelihood: $p(Y_t|Z_t) = \text{MVN}(Y_t|c_{\boldsymbol{\theta}^*}(Z_t), \Sigma)$

4a. Sample $\mathbf{z}^*{}_{1:T} \sim \hat{p}_{\boldsymbol{\theta}^*}(d\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$

4b. Compute the marginal likelihood estimate: $\widehat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T}) = \widehat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_1)\prod_{t=2}^{T}\widehat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_t|\mathbf{y}_{t-1})$.

5. Calculate the Metropolis Hastings Acceptance Probability:

$$A(\boldsymbol{\theta}^{(k-1)}, \boldsymbol{\theta}^*) = \min\left\{1, \frac{p_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})p(\boldsymbol{\theta}^*)q\big(\boldsymbol{\theta}(k-1)|\boldsymbol{\theta}^*\big)}{p_{\boldsymbol{\theta}(k-1)}(\mathbf{y}_{1:T})p\big(\boldsymbol{\theta}(k-1)\big)q(\boldsymbol{\theta}^*|\boldsymbol{\theta}(k-1))}\right\}$$
$$= \min\left\{1, \frac{p_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})p(\boldsymbol{\beta}^*, \boldsymbol{\Sigma}^*, \boldsymbol{B})q\big(\boldsymbol{\theta}(k-1)|\boldsymbol{\theta}^*\big)}{p_{\boldsymbol{\theta}(k-1)}(\mathbf{y}_{1:T})p(\boldsymbol{\beta}^{(k-1)}, \boldsymbol{\Sigma}^{(k-1)}, \boldsymbol{B}^{(k-1)})q(\boldsymbol{\theta}^*|\boldsymbol{\theta}(k-1))}\right\}$$

5a. By rejection, with probability A:

Set $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^* = \{\boldsymbol{\beta}^*, \boldsymbol{\Sigma}^*, \boldsymbol{B}^*\}$, $\mathbf{z}_{1:T}^{(k)} = \mathbf{z}^*{}_{1:T}$ and $\widehat{p}_{\boldsymbol{\theta}(k)}(\mathbf{y}_{1:T}) = \widehat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T})$,

Otherwise set $\boldsymbol{\theta}^{(k)} = \boldsymbol{\theta}^{(k-1)}$, $\mathbf{z}_{1:T}^{(k)} = \mathbf{z}_{1:T}^{(k-1)}$and $\widehat{p}_{\boldsymbol{\theta}(k)}(\mathbf{y}_{1:T}) = \widehat{p}_{\boldsymbol{\theta}(k-1)}(\mathbf{y}_{1:T})$.

end

end

5.8.3 Rao-Blackwellised PMMH

For models $\mathbf{M}_0$ and $\mathbf{M}_1$ which are totally linear we are in the happy position of being able to apply a *Rao-Blackwellised* version of the PMMH algorithm. The term *Rao-Blackwellisaztion* derives from the Rao-Blackwell theorem of statistics, which states that if $\widehat{\theta}$ is any estimator of a parameter θ then the conditional expectation of $\widehat{\theta}$ given T, where T is a sufficient statistic, is at least as good (in the mean square error sense) as $\widehat{\theta}$. Writing $\widehat{\theta}^{\star}(X) = \mathbb{E}[\widehat{\theta}(X)|T(X)]$:

$$\mathrm{MSE}(\widehat{\theta}^{\star}) \leq \mathrm{MSE}(\widehat{\theta}) \quad \forall\theta. \tag{5.31}$$

In the case of an ECM observation system with linear latent system we can use the optimal Kalman filter solution to finding $p(\mathbf{z}_{1:T}|\mathbf{y}_{1:T}, \Theta)$. The filter is optimal since when we choose the matrix $\mathbf{K}$ to be equal to the Kalman gain the estimators of $\mathbf{z}_{1:T}$ are the MMSE estimates. The Rao-Blackwellised PMMH algorithm (RB-PMMH) is a modified version of Algorithm 15 of section 5.8.1 where now the latent state estimation, and the calculation of the marginal likelihood, are performed by an optimal Kalman filter rather than a sub-optimal particle filter. Effectively this is the same as evolving a single particle to perfectly represent the Gaussian sufficient statistics of the linear system, and is thus computationally much less costly than using the full particle filter with its population of particles. However use of the RB-PMMH algorithm is limited to systems with linear latent dynamics.

Algorithm 16: RB-PMMH sampler for parameters $\boldsymbol{\beta}$, $\boldsymbol{\Sigma}$, $\boldsymbol{B}$ and latent states $\mathbf{z}_{1:T}$.

Note: *Only those parts which differ from Algorithm 15, the PMMH algorithm, are listed here.*

if *iteration* $k = 0$ **then**

⋮

2. Run an optimal *Kalman Filter* algorithm to calculate $\mathbf{z}_{1:T}(0) \sim p_{\boldsymbol{\theta}(0)}(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$

2b. Compute the marginal likelihood exactly: $p_{\boldsymbol{\theta}(0)}(\mathbf{y}_{1:T}) = p_{\boldsymbol{\theta}}(\mathbf{y}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}}(\mathbf{y}_t|\mathbf{y}_{t-1})$ using the outputs from the Kalman filter.

end

for *iteration* $k = 2$ *to* T **do**

⋮

4. Run optimal *Kalman Filter* algorithm to calculate $\mathbf{z}^*{}_{1:T} \sim p_{\boldsymbol{\theta}^*}(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$

4b. Compute the marginal likelihood exactly: $p_{\boldsymbol{\theta}^*}(\mathbf{y}_{1:T}) = \widehat{p}_{\boldsymbol{\theta}^*}(\mathbf{y}_1)\prod_{t=2}^{T} p_{\boldsymbol{\theta}^*}(\mathbf{y}_t|\mathbf{y}_{t-1})$ using the outputs from the Kalman filter.

⋮

Accept/reject using the Metropolis Acceptance probability as for PMMH Algorithm.

end

5.8.4 *Case Study:* Rao-Blackwellised PMMH and PMMH Marginal Likelihoods

The Rao-Blackwellised PMMH sampler embeds a Kalman filter to perform the latent state estimation and the calculation of the marginal likelihood $p_{\boldsymbol{\theta}}(\mathbf{y}_{1:T})$. The performance of both algorithms depends on the estimates of the marginal likelihoods produced with the Kalman or SMC filters. To check that the SMC filter is working effectively we can compare the marginal likelihood with that produced by the Kalman Filter for any of the linear models. The top left plot in Figure 5.7 shows

the surface of $\log p(\mathbf{y}_{1:T}|\beta)$ as estimated using the Kalman filter for **model $\mathbf{M}_0$**. We can compare this to the esimate calculated using the particle filter; for clarity we will look at the values from both filters for a slice through the surface at $\beta_{1,3} = 0$.

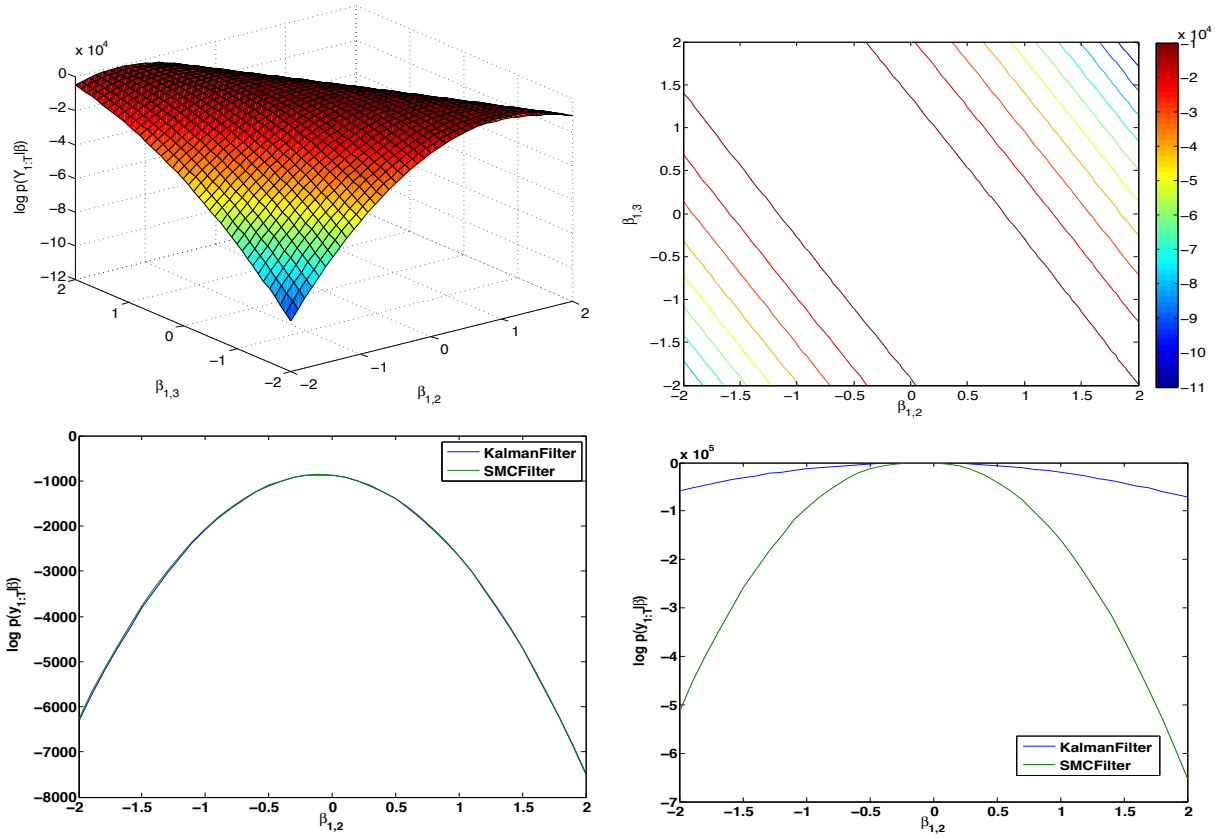


Figure 5.7: The surface and corresponding contour plots show the marginal likelihood as evaluated from the output of the Kalman Filter; the lower plots show a 'Slice' through the marginal likelihood surface at $\beta_{1,3} = 0$ for SNRs of -10dB (left), +10dB (right)

The example for $+10dB$ shows a divergence between the two filters - the reason for this is a too low noise level in the SMC filter. Figure 5.8 shows the latent states recovered by the Kalman Filter and the SMC filter for the same settings of the signal-to-noise ratio (SNR).

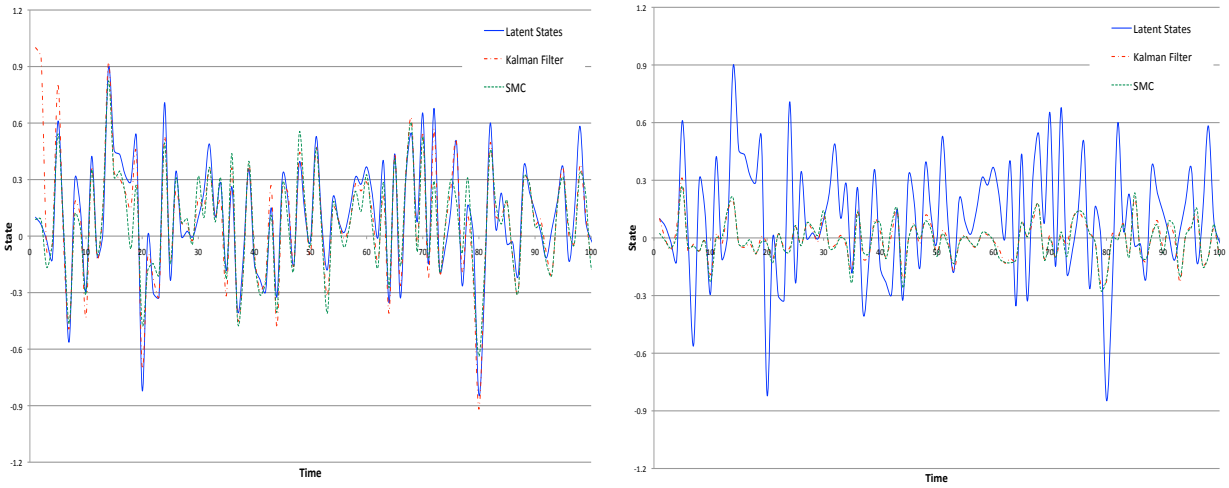


Figure 5.8: The latent states recovered using either the Kalman Filter or the SMC filter: the left plot is for SNR= $+10dB$, and the right plot is for SNR= $-10dB$.

5.9 Challenges in Practice

Despite the great benefits brought by the adaptive MCMC scheme the algorithms can still (in their present form) require a certain degree of manual tuning. Several additional parameters were encoded to give greater control over the properties of the algorithms (particularly the proposal covariances):

- Writing the initial Laplace covariance for the $\boldsymbol{\beta}$ proposals as Ω_{Laplace}, we introduce a scaling factor γ_{L} which multiplies the initial covariance $\Omega^{(0)} = \gamma_{\text{L}} \times \Omega_{\text{Laplace}}$. A value close to 1, and certainly in the range $0.1 \leq \gamma_{\text{L}} \leq 5$ was chosen.
- Writing the adaptive covariance for $\boldsymbol{\beta}$ proposals as $\Omega_{\text{Ad}}^{(j)}$ we introduce a scaling factor γ_{A} which multiplies the adaptice covariance: $\Omega^{(j)} = \gamma_{\text{A}} \times \Omega_{\text{Ad}}^{(j)}$. Values close to 1, and certainly in the range $0.1 \leq \gamma_{\text{Ad}} \leq 1$ were chosen.

These setting were made manually by observing a short run (≈ 2000 samples) of the chain for a small set of values of γ_{Ad} or γ_{L} and noting the acceptance ratio. The settings which gave the highest acceptance rate were chosen (it was rarely the case that *too many* moves were accepted as can sometimes occur for MCMC methods).

Another variable that in practice must be considered is the number of iterations, m, for which the initial proposal covariance Ω_{Laplace} (or its scaled version if γ_{L} is not set to zero) is used. This can be set using a small-scale empirical run of the algorithm for a few different values, and defining some criterion (e.g. high acceptance ratio) to target.

A key aim in future work would be to remove these manual tuning components altogether, and to find a more robust way to increase the likelihood of good mixing performance and good initial parameter settings. Possible alternative methods for the initialisation include raising the system to a temperature, T, and gradually reducing the temperature in an optimisation approach derived from simulated annealing. Alternatively more sophisticated adaption schemes are possible, and these may also lead to improved mixing performance.

Chapter 6

Synthetic Studies

6.1 Introduction

Various synthetic studies were performed to better understand the performance of the proposed samplers in the context of Bayesian CVAR models. In the case of model types $\mathbf{M}_0$ and $\mathbf{M}_1$ the observation model is chosen to be one of the following 2 models, ($\mathbf{S}_1$, and $\mathbf{S}_3$), where the digit indicates the cointegration rank, based on those of [Sugita, 2002]. We will restrict attention to the case of $n = 4$-dimensional observations, $\mathbf{y}_t$ ($= \Delta\mathbf{x}_t$), and latent states $\boldsymbol{\mu}_t$.

Label	Sugita Model	Rank, r
$\mathbf{S}_1$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_t + \begin{bmatrix} -0.2 \\ -0.2 \\ -0.2 \\ 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	1
$\mathbf{S}_3$	$\Delta\mathbf{x}_t = \boldsymbol{\mu}_t + \begin{bmatrix} -0.2 & -0.2 & -0.2 \\ 0.2 & -0.2 & -0.2 \\ 0.2 & 0.2 & -0.2 \\ 0.2 & 0.2 & 0.2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & -1 \\ 0 & 0 & 1 & -1 \end{bmatrix} \mathbf{x}_{t-1} + \boldsymbol{\epsilon}_t$	3

Table 6.1: ECM observation models considered: these govern the cointegrated part of the system.

6.2 Model $\mathbf{M}_0$: Static Latent Process (ECM rank = 1)

A series of synthetic studies were run to establish and tune the performance characteristics of the PMMH algorithm in the case of the static CVAR model $\mathbf{M}_0$. The static latent system is of the form:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \mathbf{w}_t \qquad \mathbf{w}_t \sim \mathcal{N}(0, \Sigma_w),$$

where $\boldsymbol{\mu}_0 = [0.1\ 0.1\ 0.1\ 0.1]'$, and the noise level in the latent system was set to $\Sigma_w = 0.2 \times \mathbb{I}_n$. The noise level in the observation system was set to $\Sigma = 1 \times \mathbb{I}_n$ giving a signal to noise ratio of $\sim -7dB$.

In each case 150 synthetic states were generated from the static latent system, which were used to simulate 150 corresponding observations from Sugita model $\mathbf{S}_1$. In the following sections the PMMH algorithm is used initially to estimate just the cointegrating matrix $\boldsymbol{\beta}$; the conjugate analysis of chapter 3, and prior knowledge, are used to sample exactly from parameters B and Σ. After establishing that the MCMC mixes adeqately for this limited version the other static parameters are gradually re-introduced to the problem and used to drive the SMC filter embedded within the PMMH algorithm. In effect we explore several alternative sampling schemes:

- **Scheme Ia**: Assume prior knowledge (ML estimation for example) enables us to take parameters B and Σ as known and use them to drive the filter. Adaptive proposals are made for $\boldsymbol{\beta}$ and the PMMH algorithm targets $p(\mathbf{z}_{1:t}, \boldsymbol{\beta}|\mathbf{y}_{1:t})$. Conditional on the accepted values of $\boldsymbol{\beta}$

we sample Σ and B exactly from the posteriors developed in chapter 3. On each iteration of the PMMH algorithm the proposed value of $\boldsymbol{\beta}$ is used within the SMC filter to estimate the latent states $p(\mathbf{z}_{1:t}|\mathbf{y}_{1:t}, \boldsymbol{\beta})$ and the marginal likelihood $p(\mathbf{y}_{1:t})$; i.e. Algorithm 15 was run but only the sampled values for $\boldsymbol{\beta}$ enter function $\mathbf{c}_{\boldsymbol{\theta}}$ (the others are set to the true known values).

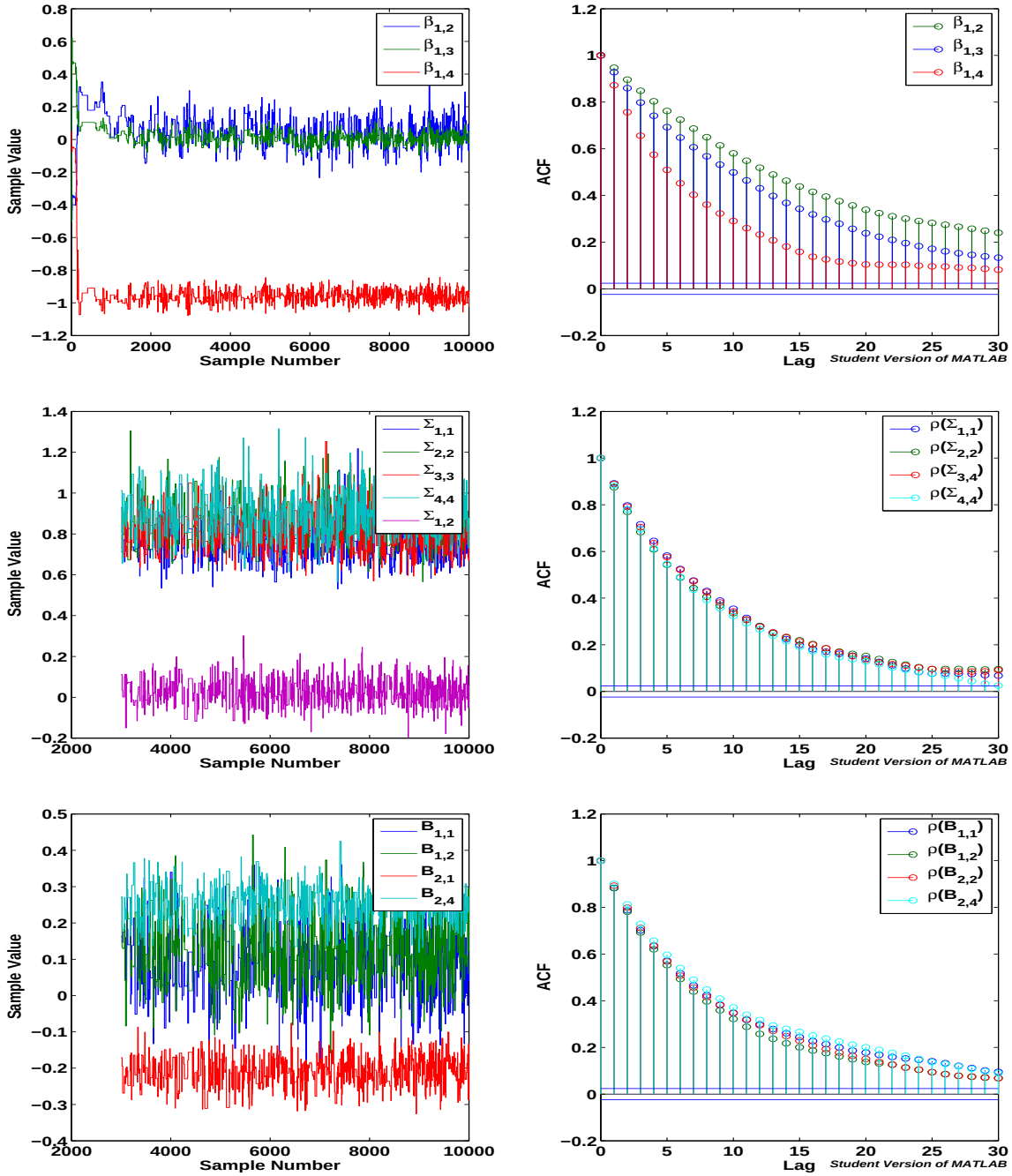
- **Scheme Ib**: As well as targetting $\boldsymbol{\beta}$ we now assume that B is also unknown. Conditional on the accepted values of $\boldsymbol{\beta}$ we sample exactly (from the posterior form) for $B|\boldsymbol{\beta}, Y$ and use this sampled value to drive the SMC filter. We continue to assume that Σ has been set with prior knowlegde.
- **Scheme Ic**: Now we target $p(\mathbf{z}_{1:t}, \boldsymbol{\beta}, \Sigma, B|\mathbf{y}_{1:t})$. Values for both B and Σ are sampled exactly from $p(B|\boldsymbol{\beta}, Y)$ and $p(\Sigma|\boldsymbol{\beta}, Y)$ and these are both used, along with the current proposed value of $\boldsymbol{\beta}$, to drive the SMC filter and to estimate the latent states $\mathbf{z}_{1:T}$.

In the examples that follow the PMMH algorithm was started with random initialisation for $\boldsymbol{\beta}^{(0)}$ and the other parameters. The recovered parameters and plots are presented for each variant of the algorithm having each been run for a chain length of 10,000 samples. The averages are made with samples 3,000:10,000 to allow for a 'burn-in' period.

For the full PMMH, and RBPMMH algorithms 5 seperate chains were generated and the averages made by first calculating the mean and standard deviation for each chain and then combining; in this way we can also calculate the standard errors of these quantities between the different chains.

6.2.1 PMMH for $\{\boldsymbol{\beta}, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$ (Exact Sampling for Σ, B, Scheme Ia)

A first example consists of running the PMMH algorithm solely for parameter $\boldsymbol{\beta}$, and assuming that prior knowledge has allowed us to take parameters B and Σ as known, this corresponds to **Scheme Ia** mentioned above. We assume for example that the values for B and Σ have been estimated via maximum likelihood procedures on a subset of our observation data. Diagnostic plots are presented in Figure 6.1 below. The embedded SMC algorithm was run with $2^7 = 128$ particles. This initial example shows good mixing for all the parameters of interest, and the algorithm is clearly converging towards the correct value for $\boldsymbol{\beta}$.

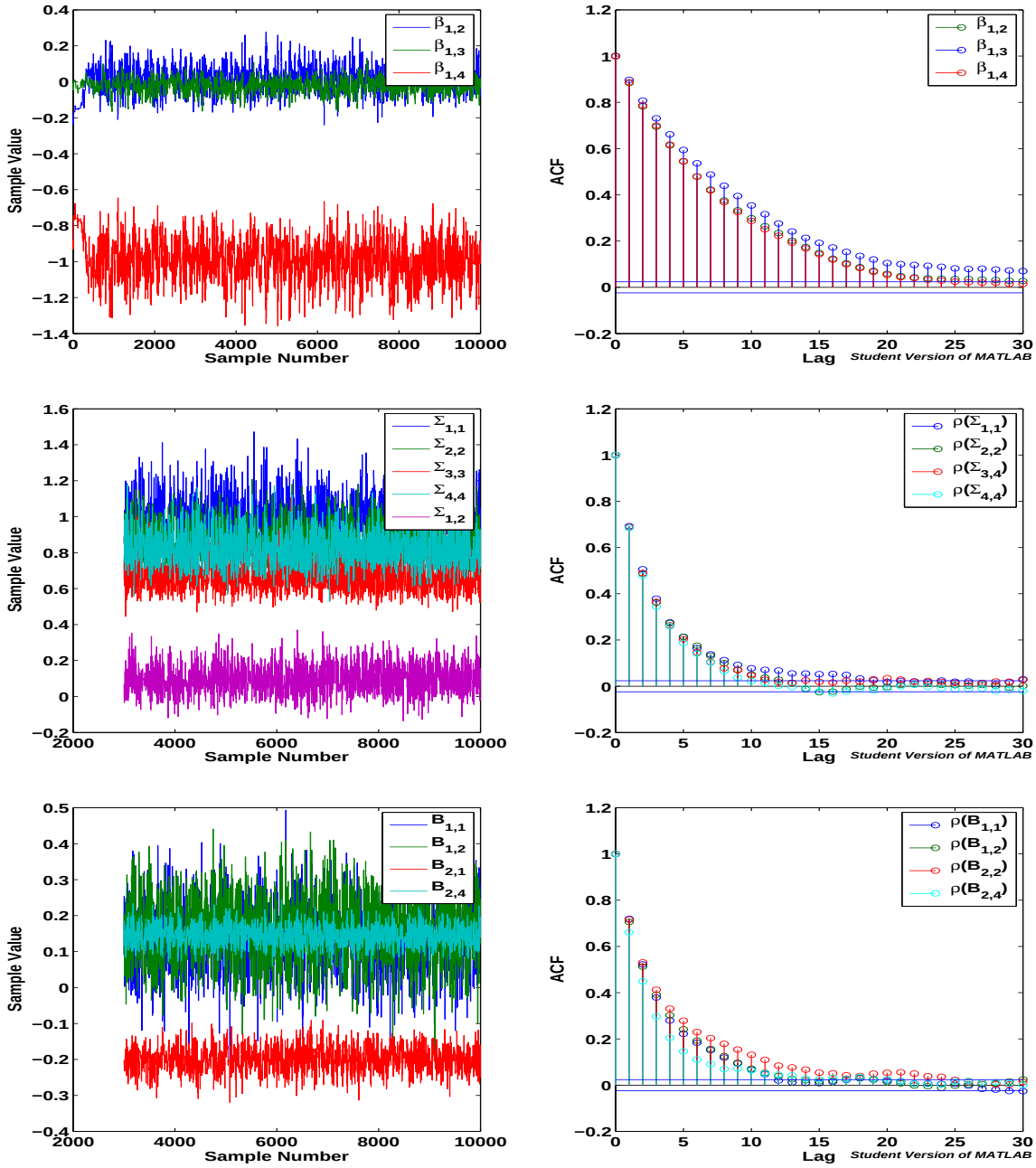


Parameter (θ)	Truth	θ	σ_θ
$\beta_{1,2}$	0.0	0.039	0.083
$\beta_{1,3}$	0.0	0.008	0.035
$\beta_{1,4}$	-1.0	-0.959	0.037
$\mathrm{Tr}(\Sigma)$	4.0	3.335	0.214
$\Sigma_{1,2}$	0.0	0.027	0.067
$\alpha_{1,1}$	-0.2	-0.211	0.042
$\alpha_{1,2}$	-0.2	-0.217	0.043
$\alpha_{1,3}$	-0.2	-0.232	0.041
$\alpha_{1,4}$	0.2	0.253	0.049

Figure 6.1: **Model** : $\mathbf{M}_0, \mathbf{S}_1$, Samples from $p(\boldsymbol{\beta}|Y)$, $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$. The PMMH algorithm was run to target $\{\boldsymbol{\beta}, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps, with adaptive proposal covariance from the 5th iteration. The table gives the mean parameter values for $\boldsymbol{\beta}$, Σ and B from samples 3,000:10,000 of the algorithm; the values for Σ and B are sampled conditional on the $\boldsymbol{\beta}$ samples.

6.2.2 PMMH for $\{\boldsymbol{\beta}, B, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$ (Exact Sampling for Σ, Scheme Ib)

The PMMH algorithm was re-run and the values of B which are exactly sampled (conditional on the accepted values of $\boldsymbol{\beta}$) were used as parameters driving the SMC filter.

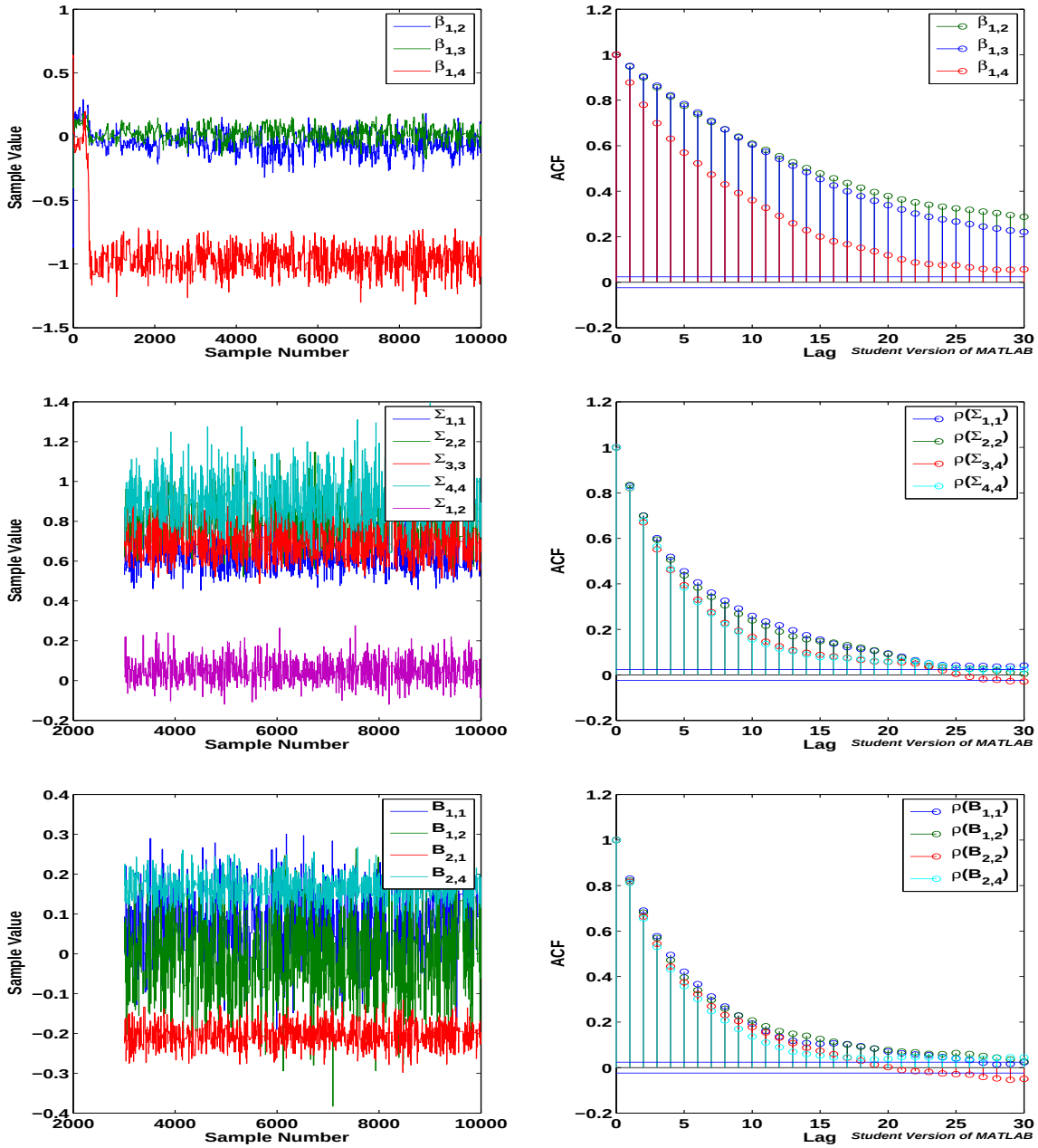


Parameter (θ)	Truth	θ	σ_θ
$\beta_{1,2}$	0.0	0.015	0.071
$\beta_{1,3}$	0.0	-0.024	0.040
$\beta_{1,4}$	-1.0	-0.998	0.111
$\mathrm{Tr}(\Sigma)$	4.0	3.335	0.195
$\Sigma_{1,2}$	0.0	0.099	0.077
$\alpha_{1,1}$	-0.2	-0.199	0.031
$\alpha_{1,2}$	-0.2	-0.219	0.027
$\alpha_{1,3}$	-0.2	-0.188	0.027
$\alpha_{1,4}$	0.2	0.145	0.025

Figure 6.2: **Model** : $\mathbf{M}_0, \mathbf{S}_1$, Samples from $p(\boldsymbol{\beta}|Y)$, $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$. The PMMH algorithm was run to target $\{\boldsymbol{\beta}, B, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps, with adaptive proposal covariance from the 5th iteration. The table gives the mean parameter values for $\boldsymbol{\beta}$, Σ and B from samples 3,000:10,000 of the algorithm; the values for Σ are sampled conditional on the $\boldsymbol{\beta}$ samples.

6.2.3 PMMH for $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:T}|\mathbf{Y}\}$ (Scheme Ic)

The full PMMH algorithm was run for 10,000 samples targeting static parameters $\boldsymbol{\beta}, \Sigma, B$ and the latent states $\mathbf{z}_{1:T}$ with 5 separate data-sets each having 150 observations of the rank 1 model.

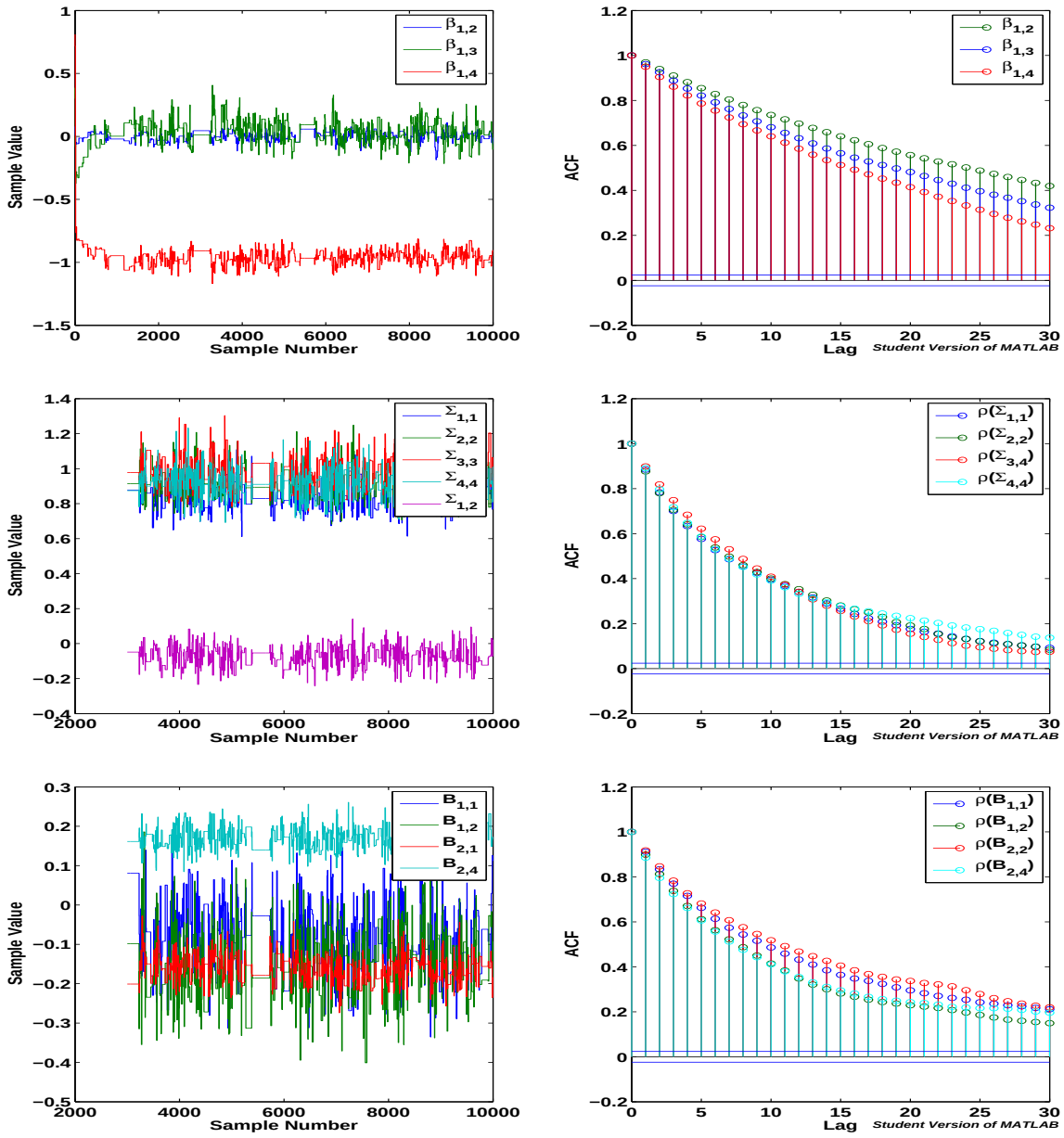


Par. (θ)	Truth	$\theta(SE(\theta))$	$\sigma_\theta(SE(\sigma_\theta))$
$\beta_{1,2}$	0.0	0.014 (.021)	0.057 (.010)
$\beta_{1,3}$	0.0	0.018 (.027)	0.048 (.008)
$\beta_{1,4}$	-1.0	-0.953 (.051)	0.086 (.012)
Tr(Σ)	4.0	3.321 (.151)	0.196 (.008)
$\Sigma_{1,2}$	0.0	0.023 (.023)	0.069 (.004)
$\alpha_{1,1}$	-0.2	-0.190 (.021)	0.035 (.002)
$\alpha_{1,2}$	-0.2	-0.214 (.016)	0.034 (.001)
$\alpha_{1,3}$	-0.2	-0.176 (.022)	0.032 (.001)
$\alpha_{1,4}$	0.2	0.229 (.027)	0.036 (.003)

Figure 6.3: **Model** : $\mathbf{M}_0, \mathbf{S}_1$; Samples from $p(\boldsymbol{\beta}|Y)$, $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$ for the PMMH algorithm run to target $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps, adaptive proposal covariance from the 5th iteration. The table shows parameters found from averaging samples 3000:10000 of the output, and numbers in brackets are standard errors between the 5 data sets.

6.2.4 RBPMMH for $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$, (Scheme Ic)

The Rao-Blackwellised version is considerably faster than the PMMH algorithm ($\approx 2070s$ for RBPMMH, with $5 \times 10,000$ samples, vs. $\approx 9000s$ PMMH: running on a 2.53GHz Intel Core 2 Duo Processor with 4GB RAM).



Par. (θ)	Truth	$\theta(SE(\theta))$	$\sigma_\theta(SE(\sigma_\theta))$
$\beta_{1,2}$	0.0	0.045 (.026)	0.050 (.011)
$\beta_{1,3}$	0.0	-0.014 (.018)	0.045 (.011)
$\beta_{1,4}$	-1.0	-0.879 (.058)	0.080 (.012)
$\mathrm{Tr}(\Sigma)$	4.0	3.483 (.050)	0.152 (.004)
$\Sigma_{1,2}$	0.0	0.015 (.028)	0.056 (.000)
$\alpha_{1,1}$	-0.2	-0.193 (.018)	0.038 (.002)
$\alpha_{1,2}$	-0.2	-0.203 (.019)	0.036 (.003)
$\alpha_{1,3}$	-0.2	-0.158 (.022)	0.034 (.003)
$\alpha_{1,4}$	0.2	0.182 (.013)	0.030 (.001)

Figure 6.4: **Model** : $\mathbf{M}_0, \mathbf{S}_1$, {Noise: $[\Sigma_w, \Sigma] = [\mathbf{0.2}I_n, \mathbf{1}I_n]$}, Samples from $p(\beta|Y)$, $p(\Sigma|\beta, Y)$ and $p(B|\beta, Y)$ from the RB-PMMH algorithm run for 10,000 steps, with adaptive proposal covariance from the 5th iteration; and Table showing the parameters found from averaging samples 3000:10000 of the output. The numbers in brackets are standard errors between the 5 data sets.

6.2.5 Discussion Model $\mathbf{M}_0$, ECM Rank = 1

In this section results have been presented for three variants of Algorithm 15 - the Particle Marginal Metropolis-Hastings algorithm, and the corresponding Rao-Blackwellised version - for estimation of a static latent system with rank 1 observation model.

PMMH: Scheme Ia

This example achieves good mixing with the required decay of the autocorrelation function by lag $l = 30$ (to ≈ 0.2) for two of the components of $\boldsymbol{\beta}$. Since the other parameters, B and Σ, are 'exactly' sampled we would expect there to be good mixing for these; this is confirmed by the ACF plots. The value of $\mathrm{Tr}(\Sigma)$, 3.335, is below its true value (4), and lies outside the limit of 1 standard deviation (0.214). The reason for this is probably related to the performance of the filter under varying noise conditions, more discussion of this is postponed to section 6.4. The values for $\boldsymbol{\beta}_{1,2}$ and $\boldsymbol{\beta}_{1,3}$ are recovered within one standard deviation of their true values, but $\boldsymbol{\beta}_{1,4}$ is slightly outside this; but clearly converging towards the true value.

PMMH Scheme Ib

Including the parameter B has, in this example, actually improved the performance of the sampler; the decay of the ACF for $\boldsymbol{\beta}$ is in line with the desired performance. Again the value for $\mathrm{Tr}(\Sigma)$ is slightly low as in the previous example. The values for α_1 have been recovered within 1 standard deviation of their true values (apart from $\alpha_{1,4}$, which is also slightly too low).

PMMH: Scheme Ic

This is an example of the full PMMH scheme, and targets $\boldsymbol{\beta}, \Sigma$ and B as well as the latent states. For this scheme standard errors between 5 separate datasets were calculated to give an indication of the variability in the algorithm's performance. The mixing is adequate, and could perhaps be slightly improved for two of the $\boldsymbol{\beta}$ components (sub-sampling could be used to reduce the autocorrelation between the generated samples as discussed in section 5.6).

All the parameters for $\boldsymbol{\beta}$ are recovered within standard error. The signal-to-noise characteristics for this run were the same as for the previous examples ($+7dB$) and the recovered value for $\mathrm{Tr}(\Sigma)$ is, again, slightly lower than the true value. Overall the algorithm has shown encouraging performance, and has enabled the most difficult parameter $\boldsymbol{\beta}$ to be recovered satisfactorily, and the α parameter (which is part of B) is also recovered well (very nearly within standard error for all components).

RBPMMH

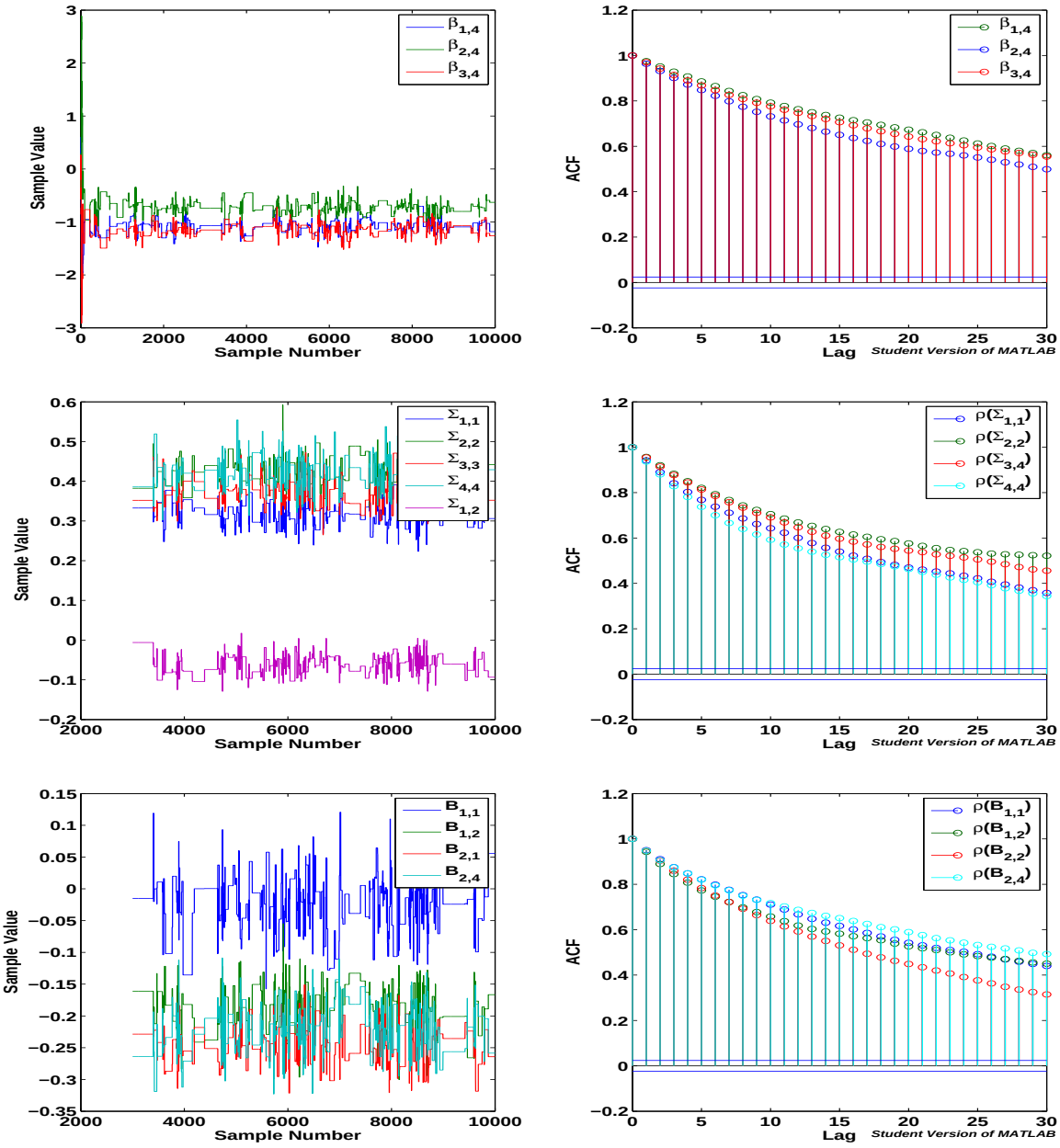
The RBPMMH algorithm is - in practice - much faster to run than the particle version, and also generally gives good results. The decay of the ACF for $\boldsymbol{\beta}$ in this example is slightly slow for two of the components, and indicates that sub-sampling may be necessary to reduce the autocorrelation amongst the generated samples. In the example plots it is interesting to note the point somewhere around sample 5500 where the algorithm 'gets stuck'; after around 300 samples the algorithm begins to make progress again. The decays of the ACF's for Σ and B are both acceptable.

The standard errors have been calculated between 5 separate chains, and the average recovered value for Σ is closer to its true value than for earlier versions (but still outside the S.E.). The recovered value for $\boldsymbol{\beta}$ is not in agreement with the true value; this may be improved by adjusting the tuning of the algorithm or implementing sub-sampling to reduce the ACF amongst the $\boldsymbol{\beta}$ samples.

Remark *The examples in the proceeding section are from the same static latent model, but for the rank 3 observation system.*

6.3 Model $\mathbf{M}_0$: Static Latent Process, ECM rank =3

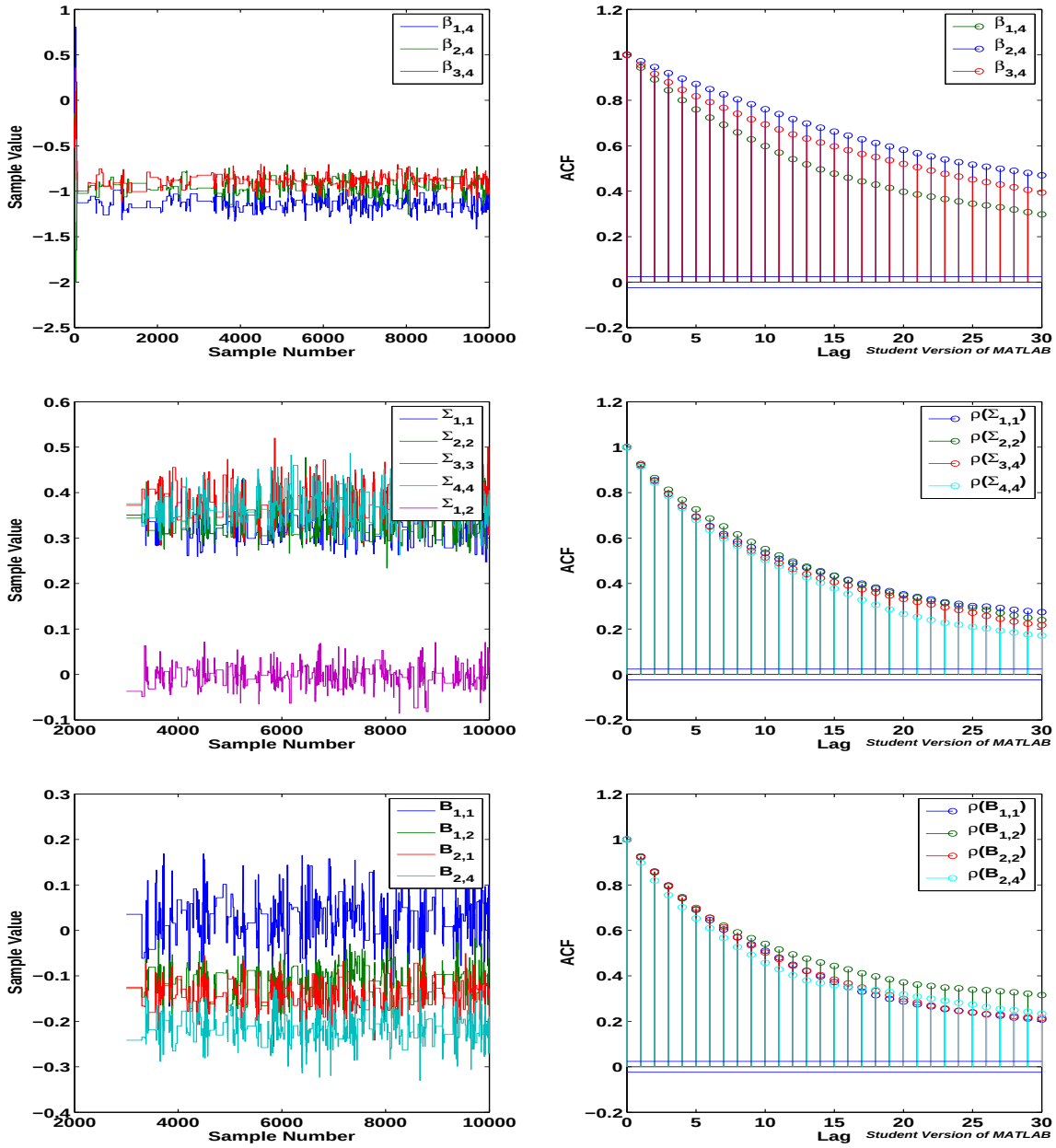
6.3.1 PMMH for $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$



Par. (θ)	Truth	$\theta(SE(\theta))$	$\sigma_\theta(SE(\sigma_\theta))$
$\beta_{1,4}$	-1.0	-0.897 (.070)	0.081 (.015)
$\beta_{2,4}$	-1.0	-0.973 (.076)	0.084 (.011)
$\beta_{3,4}$	-1.0	-1.022 (.039)	0.068 (.014)
Tr(Σ)	2.0	1.503 (.056)	0.069 (.005)
$\Sigma_{1,2}$	0.0	0.020 (.023)	0.026 (.001)
$\alpha_{1,1}$	-0.2	-0.235 (.015)	0.024 (.002)
$\alpha_{1,2}$	0.2	0.174 (.010)	0.029 (.002)
$\alpha_{1,3}$	0.2	0.191 (.021)	0.025 (.001)
$\alpha_{1,4}$	0.2	0.162 (.023)	0.025 (.002)
$\alpha_{2,1}$	-0.2	-0.228 (.013)	0.028 (.004)
$\alpha_{2,2}$	-0.2	-0.208 (.033)	0.031 (.003)
$\alpha_{2,3}$	0.2	0.186 (.013)	0.028 (.002)
$\alpha_{2,4}$	0.2	0.191 (.015)	0.028 (.003)
$\alpha_{3,1}$	-0.2	-0.160 (.028)	0.034 (.001)
$\alpha_{3,2}$	-0.2	-0.187 (.015)	0.031 (.004)
$\alpha_{3,3}$	-0.2	-0.214 (.025)	0.032 (.002)
$\alpha_{3,4}$	0.2	0.260 (.015)	0.031 (.002)

Figure 6.5: **Model** : $\mathbf{M}_0, \mathbf{S}_3$: Samples from $p(\boldsymbol{\beta}|Y)$, $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$ for the PMMH algorithm run to target $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps repeated over 5 separate datasets, with adaptive proposal covariance from the 5th iteration. The table shows parameters found from averaging samples 3000:10000 of the output. The numbers in brackets are standard errors between the 5 data sets.

6.3.2 RBPMMH for $\{\boldsymbol{\beta}, \Sigma, B\}, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}$



Par (θ)	Truth	$\theta(SE(\theta))$	$\sigma_\theta(SE(\sigma_\theta))$
$\beta_{1,4}$	-1.0	-0.993 (.040)	0.134 (.033)
$\beta_{2,4}$	-1.0	-0.980 (.076)	0.133 (.024)
$\beta_{3,4}$	-1.0	-0.925 (.029)	0.115 (.022)
Tr(Σ)	2.0	1.532 (.076)	0.071 (.003)
$\Sigma_{1,2}$	0.0	0.002 (.007)	0.024 (.002)
$\alpha_{1,1}$	-0.2	-0.192 (.014)	0.030 (.002)
$\alpha_{1,2}$	0.2	0.185 (.026)	0.031 (.002)
$\alpha_{1,3}$	0.2	0.186 (.025)	0.030 (.002)
$\alpha_{1,4}$	0.2	0.169 (.031)	0.028 (.002)
$\alpha_{2,1}$	-0.2	-0.171 (.018)	0.033 (.001)
$\alpha_{2,2}$	-0.2	-0.232 (.016)	0.032 (.001)
$\alpha_{2,3}$	0.2	0.177 (.025)	0.033 (.001)
$\alpha_{2,4}$	0.2	0.212 (.028)	0.029 (.002)
$\alpha_{3,1}$	-0.2	-0.189 (.019)	0.033 (.002)
$\alpha_{3,2}$	-0.2	-0.203 (.006)	0.034 (.003)
$\alpha_{3,3}$	-0.2	-0.218 (.025)	0.032 (.001)
$\alpha_{3,4}$	0.2	0.204 (.015)	0.030 (.001)

Figure 6.6: **Model** : $\mathbf{M}_0, \mathbf{S}_3$: Samples from $p(\boldsymbol{\beta}|Y)$, $p(\Sigma|\boldsymbol{\beta}, Y)$ and $p(B|\boldsymbol{\beta}, Y)$ for the RBPMMH algorithm run to target $\{\boldsymbol{\beta}, \Sigma, B, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps repeated over 5 separate datasets, with adaptive proposal covariance from the 5th iteration. The table shows parameters found from averaging samples 3000:10000 of the output. The numbers in brackets are standard errors between the 5 data sets.

6.3.3 Discussion: Model $\mathbf{M}_0$, ECM Rank = 3

The examples in this section were from the PMMH and RBPMMH algorithms running on a rank 3 model.

PMMH

The algorithm has succesfully converged towards the true parameter values, finding $\boldsymbol{\beta}_{2,4}$ and $\boldsymbol{\beta}_{3,4}$ within standard error. Parameter $\boldsymbol{\beta}_{1,4}$ is slightly out. Unfortunately the mixing properties of the algorithm applied to this example are not as good as for the rank 1 system. The ACF's are still too high at lag $l = 30$ having a value of around ≈ 0.5 for $\boldsymbol{\beta}$. The rank 3 system is more difficult to tune (in the sense of choosing the appropriate covariance scaling) than the rank 1 system.

The SNR in this example is $+0dB$, both Σ_w and Σ were set to $0.5 \times \mathbb{I}_n$. The recovered covariance of the observation noise, Σ has $\mathrm{Tr}(\Sigma) = 1.503$, with standard error 0.076; the true value is 2. Around half the elements of α were recovered within standard error.

RBPMMH

This version has slightly better performance than the PMMH algorithm in the rank 3 system for the examples given here. The mixing, although not perfect, is better than for the PMMH algorithm; the ACF of $\boldsymbol{\beta}$ has decayed to ≈ 0.4 at lag $l = 30$, and the ACF's for the other parameters Σ and B has dropped to ≈ 0.3 or better by the same point.

$\boldsymbol{\beta}_{1,4}$ and $\boldsymbol{\beta}_{2,4}$ are recovered within standard error, and a clear majority of the elements of α are recovered within error too. The value of Σ, however, is too low and takes a similar value to the previous example: the reasons for which will be discussed in section 6.4

6.4 Model $\mathbf{M}_1$: Dynamic Linear Latent Process & Sampling Schemes

A VAR(1) system is introduced as the latent linear dynamic process, which is of the form mentioned in Chapter 3:

$$\boldsymbol{\mu}_t = F\boldsymbol{\mu}_{t-1} + \boldsymbol{\mu}_0 + \mathbf{w}_t.$$

Since this model is linear the Rao-Blackwellised version of the PMMH sampler can be used. The marginal likelihood surface is the same for both filters as shown, for a linear model, in Figure 5.8. In this model we may also be interested in attempting to perform inference for the latent transition matrix F, and the latent state noise covariance Σ_w. To aid in this task we can use the posterior analysis given for this system in Chapter 3. The sampler components are gradually built up in stages, in what amount to several possible alternative sampling schemes, as follows:

- Scheme I: The adaptive proposal mechanism is used to explore the cointegrating vector parameter $\boldsymbol{\beta}$. Known parameter values are used to drive the Kalman Filter or SMC filter, and then, conditional on $\boldsymbol{\beta}$ and Y the conjugate posteriors allow 'exact' sampling from $\sim p(\Sigma|\boldsymbol{\beta}, Y)$ and $\sim p(B|\boldsymbol{\beta}, Y)$. The sampled values of Σ, B are not used as part of the acceptance ratio, exploration is driven purely by $\boldsymbol{\beta}$. This scheme allows us to test the conjugate sampling components. Figure 6.1 was generated by scheme I.

- Scheme II: The adaptive proposal mechanism is used to explore the $\boldsymbol{\beta}$ posterior, prior knowledge is used to set the parameter values which drive the Kalman Filter or SMC filter and $\Sigma \sim p(\Sigma|\boldsymbol{\beta}, Y)$ and $B \sim p(B|\boldsymbol{\beta}, Y)$ are sampled exactly. At each iteration the SMC or Kalman filters return a draw from the latent path space $\mathbf{z}_{1:T} \sim p_\theta(\mathbf{z}_{1:T}|\mathbf{y}_{1:T})$ (where θ is the vector of static parameters). *Conditional* on these sampled latent states the posterior analysis of Chapter 3 can be used to sample in a block Gibbs scheme from $F \sim p(F|\boldsymbol{z}_{1:T})$ and $\Sigma_w \sim p(\Sigma_w|\mathbf{z}_{1:T})$.

- Scheme III: As scheme II, but now all the sampled parameters from the Gibbs blocks are used to drive the SMC or Kalman filters.

6.4.1 RBPMMH for $\{\boldsymbol{\beta}, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$, with Gibbs blocks for $\{B, \Sigma, F, \Sigma_w\}$ (Scheme II)

The Rao-Blackwellised PMMH sampler was run for a chain length of 10,000 samples on system model $\mathbf{M}_1(\mathbf{VAR}(1))$. The observation noise level was set to $\boldsymbol{\epsilon} \sim N(0, 0.1 \times I_n)$, and the system noise level was set to $\boldsymbol{w}_t \sim N(0, 1 \times I_n)$; giving a signal-to-noise ratio of $-10dB$. The samples drawn for one run of the algorithm are plotted across two figures in Figures 6.7 and 6.8

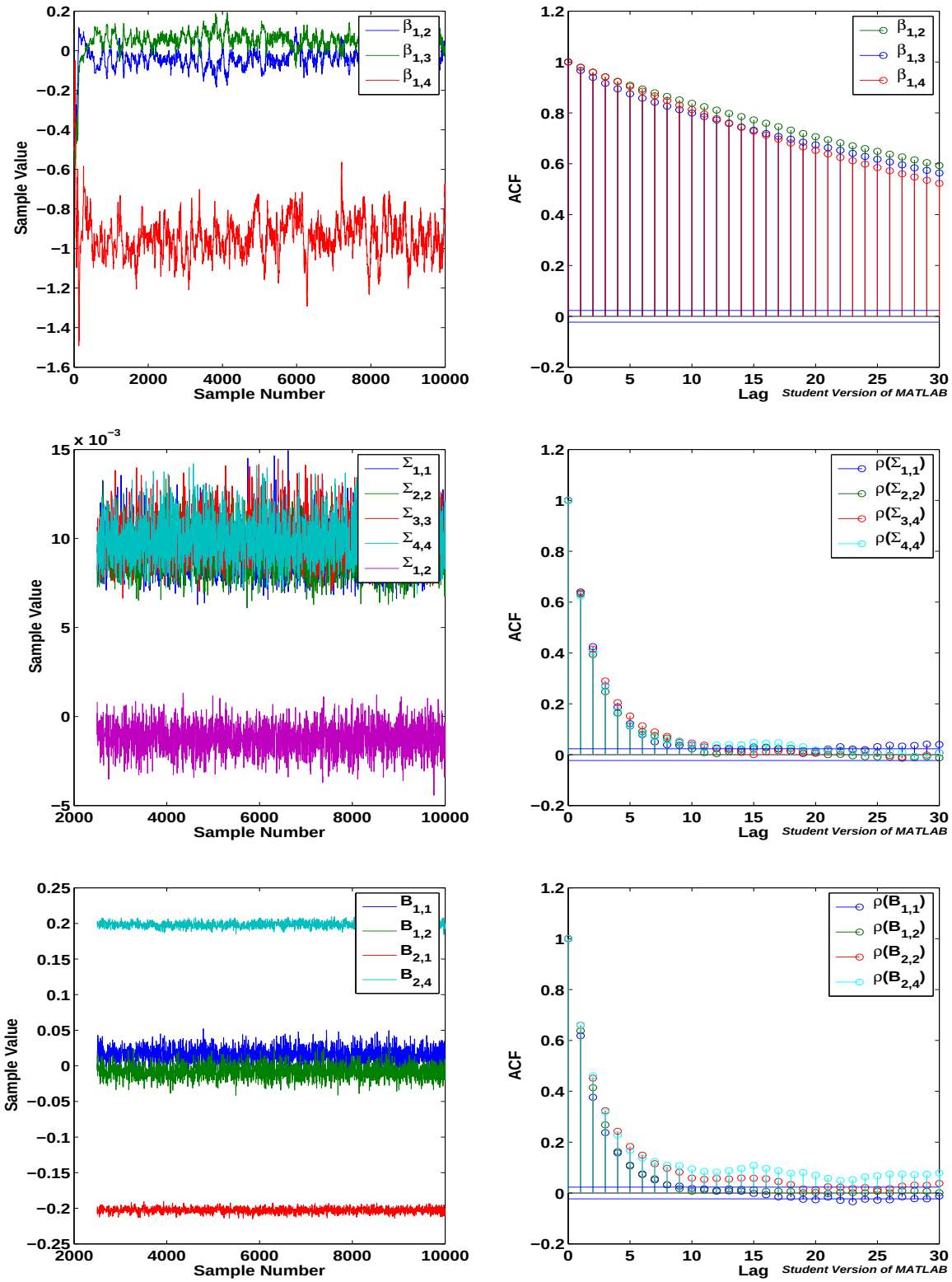


Figure 6.7: **Model** : $\mathbf{M}_1(\mathbf{VAR}(1)), \mathbf{S}_1, \{\Sigma_w = \mathbf{1} \times I_n, \Sigma = \mathbf{0.1} \times I_n, F = 0.3 \times I_n, \boldsymbol{\mu}_0 = [0.1\ 0.1\ 0.1\ 0.1]'\}$ Samples from $\log p(\beta|Y)$, $\log p(\Sigma|\beta, Y)$ and $\log p(B|\beta, Y)$ for the RB-PMMH algorithm run for 10,000 steps, with adaptive proposal covariance from the 100th iteration.

The autocorrelation functions for the exactly sampled variables Σ and B show a very rapid decay; this is expected since these are random samples (there is some slight autocorrelation introduced in the Σ and B samples through the value of β, if we sampled exactly from $p(\Sigma|Y, \beta_0)$ and $p(B|Y, \beta_0)$ for a fixed value of β_0 the Σ and B samples would display zero autocorrelation).

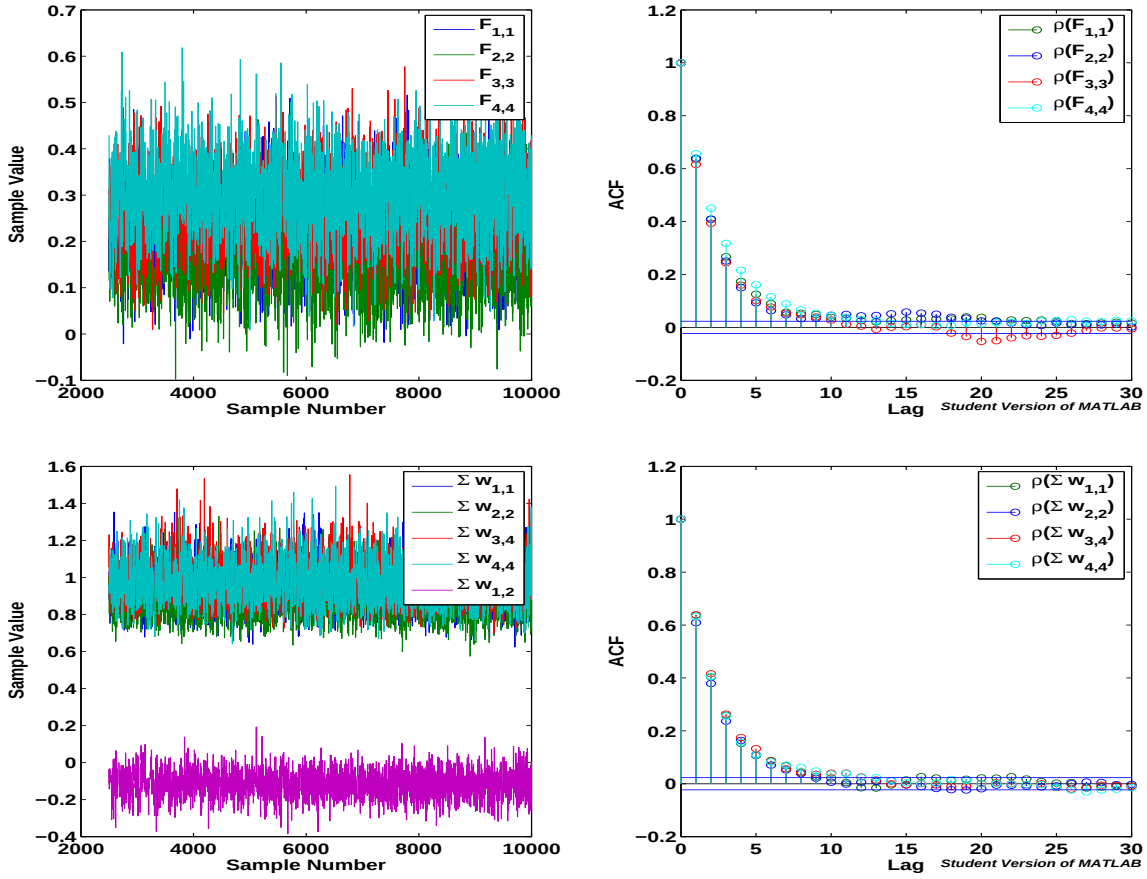


Figure 6.8: **Model** : $\mathbf{M}_1, (\mathbf{VAR}(1)), \mathbf{S}_1, \{\Sigma_w = \mathbf{1} \times I_n, \Sigma = \mathbf{0.1} \times I_n, F = 0.3 \times I_n, \boldsymbol{\mu}_0 = [0.1\ 0.1\ 0.1\ 0.1]'\}$ Samples from $\log p(F|M)$, $\log p(\Sigma_w|M)$ for the RB-PMMH algorithm run for 10,000 steps, with adaptive proposal covariance from the 100th iteration.

Parameter (θ)	Truth	Mean ($\bar{\theta}$)	$\sigma(\theta)$
$\beta_{1,4}$	0.0	-0.041	0.043
$\beta_{2,4}$	0.0	0.056	0.037
$\beta_{3,4}$	-1.0	-0.941	0.095
$\mathrm{Tr}(\Sigma)$	0.4	0.038	0.002
$\Sigma_{1,2}$	0.0	-0.001	0.001
$\alpha_{1,1}$	-0.2	-0.203	0.004
$\alpha_{1,2}$	-0.2	-0.202	0.004
$\alpha_{1,3}$	-0.2	-0.203	0.004
$\alpha_{1,4}$	0.2	0.198	0.004
$\mathrm{Tr}(\Sigma_w)$	4.0	3.793	0.224
$\Sigma_{1,2}$	0.0	-0.112	0.075
$(\mu_0)_1$	0.1	0.144	0.090
$(\mu_0)_2$	0.1	-0.117	0.093
$(\mu_0)_3$	0.1	-0.040	0.091
$(\mu_0)_4$	0.1	-0.082	0.092
$\mathrm{Tr}(F)$	1.2	0.981	0.162
$F_{1,2}$	0.0	0.031	0.079

Table 6.2: Average parameter values calculated from samples 3,000:10,000 for **Model** : $\mathbf{M}_1(\mathbf{VAR}(1)), \mathbf{S}_1$.

6.4.2 RBPMMH for $\{\boldsymbol{\beta}, B, F, \Sigma_w, \mathbf{z}_{1:T}|\mathbf{Y}\}$ (Scheme III)

The RBPMMH algorithm was run for a chain length of 10,000 samples targeting static parameters $\boldsymbol{\beta}, B, F, \Sigma_w$ and the latent states $\mathbf{z}_{1:T}$ with 5 separate data-sets each having 150 observations of the rank 1 model. This algorithm follows sampling scheme III as set out at the start of the section: block Gibbs updates are used to sample F and Σ_w conditional on the sampled latent path, and then these parameters are used to drive the filter on the subsequent iteration. The acceptance ratio includes $\boldsymbol{\beta}, B, \Sigma_w$ and F.

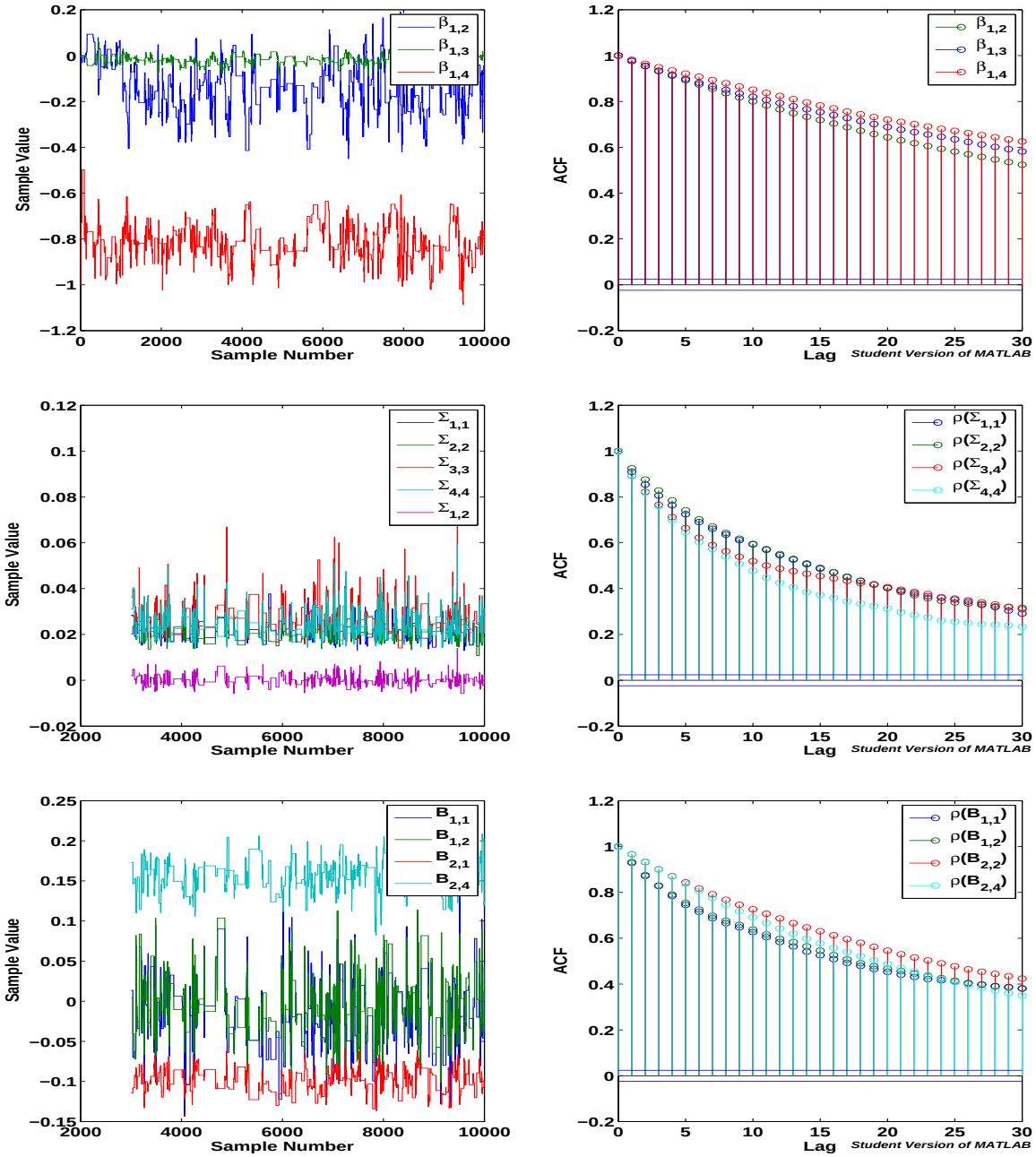


Figure 6.9: **Model** : $\mathbf{M}_1(\mathbf{VAR}(1)), \mathbf{S}_1$: Samples from $p(\beta|Y)$, $p(\Sigma|\beta, Y)$ and $p(B|\beta, Y)$ for the PMMH algorithm run to target $\{\beta, \Sigma, B, F, \Sigma_w, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps.

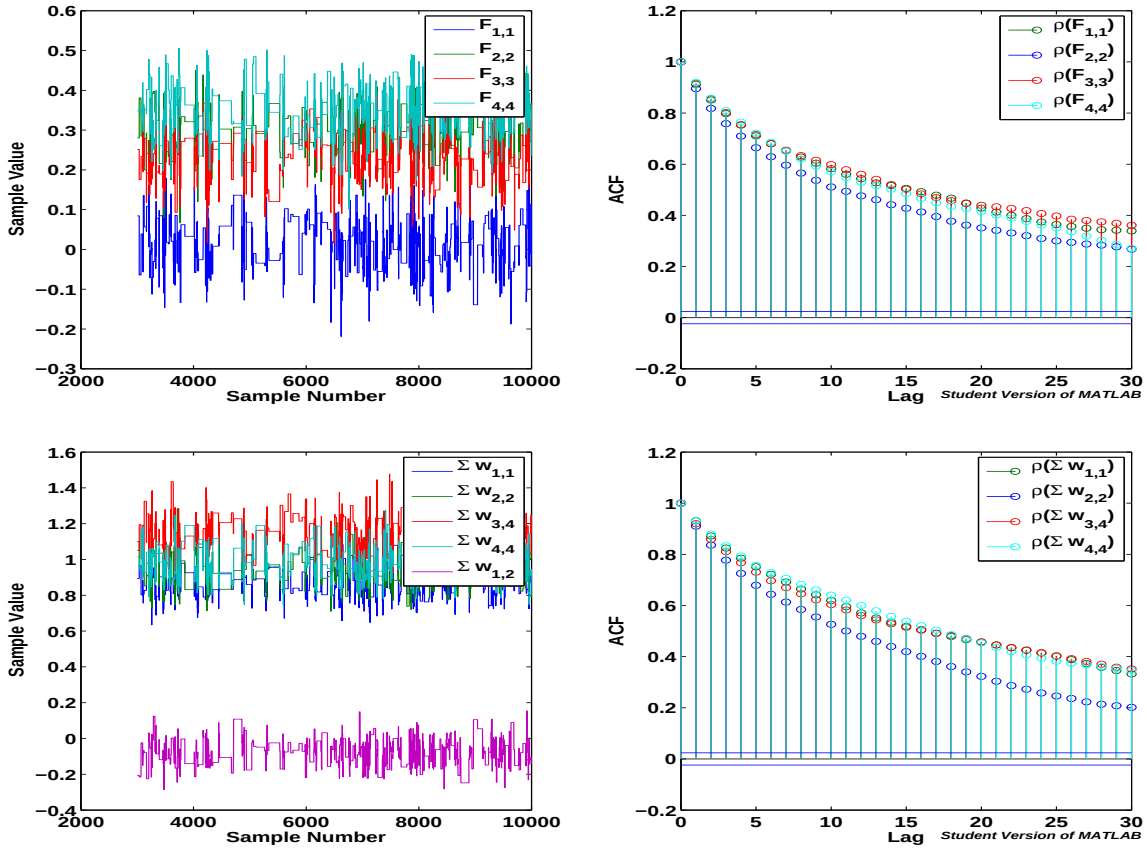


Figure 6.10: **Model** : $\mathbf{M}_1(\mathbf{VAR}(1)), \mathbf{S}_1$: Samples from $p(F|Y)$, $p(\Sigma_w|\beta, Y)$ for the RBPMMH algorithm run to target $\{\beta, \Sigma, B, F, \Sigma_w, \mathbf{z}_{1:T}|\mathbf{y}_{1:T}\}$ for 10,000 steps.

Parameter (θ)	Truth	$\bar{\theta}(SE(\theta))$	$\sigma_\theta(SE(\sigma_\theta))$
$\beta_{1,2}$	0.0	0.022 (0.071)	0.066 (0.018)
$\beta_{1,3}$	0.0	-0.040 (0.008)	0.036 (0.007)
$\beta_{1,4}$	-1.0	-0.938 (0.083)	0.060 (0.011)
$\text{Tr}(\Sigma)$	0.4	0.097 (0.005)	0.016 (0.002)
$\Sigma_{1,2}$	0.0	0.003 (0.001)	0.003 (0.000)
$\alpha_{1,1}$	-0.2	-0.158 (0.024)	0.018 (0.001)
$\alpha_{1,2}$	-0.2	-0.171 (0.029)	0.019 (0.001)
$\alpha_{1,3}$	-0.2	-0.220 (0.030)	0.020 (0.002)
$\alpha_{1,4}$	0.2	0.161 (0.012)	0.020 (0.002)
$\text{Tr}(\Sigma_w)$	4.0	3.740 (0.104)	0.166 (0.007)
$\Sigma_{1,2}$	0.0	-0.021 (0.021)	0.079 (0.006)
$(\mu_0)_1$	0.1	-0.029 (0.040)	0.097 (0.005)
$(\mu_0)_2$	0.1	0.080 (0.066)	0.099 (0.004)
$(\mu_0)_3$	0.1	0.143 (0.036)	0.108 (0.008)
$(\mu_0)_4$	0.1	0.077 (0.014)	0.094 (0.005)
$\text{Tr}(F)$	1.2	0.961 (0.113)	0.116 (0.002)
$F_{1,2}$	0.0	0.080 (0.066)	0.099 (0.004)

Table 6.3: Average parameters from 5 separate runs of the RB-PMMH algorithm with dynamic latent VAR system.

6.4.3 Disussion: Model $\mathbf{M}_1$

In this section the RBPMMH algorithm was used to target one of two schemes:

- $\{\boldsymbol{\beta}, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$ with Gibbs blocks for $\{B, \Sigma, F, \Sigma_w\}$ (Scheme II)
- $\{\boldsymbol{\beta}, B, F, \Sigma_w, \mathbf{z}_{1:\mathbf{T}}|\mathbf{Y}\}$ (Scheme III)

Scheme II

The ACF plots for $\boldsymbol{\beta}$ for this dynamic example shows that the samples are still too correlated, and sub-sampling may be required to overcome this. The samples for Σ, B have very low autocorrelation since they are being sampled exactly from the known posteriors; the values of the components of B in particular have small variance. This scheme really serves to test the exact sampling portions of the algorithm, and in the setting of this model also the recovery of the noise in the latent states, Σ_w.

The samples for F and Σ_w also show a very sharp decay in their autocorrelation functions; this is as expected since the samples, conditioned on the recovered latent paths, are being exactly sampled (and in this scheme are not being used to drive the filtering). The value for $\mathrm{Tr}(\Sigma)$ recovered was 3.79, which is within one standard deviation of the true value, 4. The value of $\mathrm{Tr}(F)$ has been recovered as 0.98, where the true value is 1.2.

Scheme III

This scheme brings together all the elements of the sampler into a single RBPMMH algorithm (and these schemes can be applied to similar effect in the PMMH version). The recovered latent states are used to condition the exact sampling of the parameters from the posteriors and these, in turn, drive the filter. The autocorrelation of the $\boldsymbol{\beta}$ samples does not decay as rapidly as desired; again either improvements in tuning or sub-sampling are required here. The algorithm is more complex, and there are now more dependencies between the sampled parameters and the filter; this appears to make tuning more difficult. However the algorithm is successfully recovering many of the parameters within standard error, despite the inadequate mixing.

We may begin to understand the poorer mixing performance for the cases where we attempt the estimation of a greater number of parameters by considering the effect this has on the shape of the posterior likelihood surface. In the sampling schemes we have described, parameters such as Σ and B that are used to drive filtering are sampled exactly from their known posterior distributions. The stochasticity generated in this sampling causes the likelihood surface (which, of course the MCMC portion of the algorithm is exploring) to become distorted. A simple example is given in Figure 6.11 below, which shows the effect on the marginal likelihood $p_\theta(\mathsf{y}_{1:T})$ of either taking Σ as fixed throughout or sampling it before evaluating the likelihood at each point on the surface; the marginal likelihood surfaces are generated using the SMC filter. Also the same test was tried where we sampled the parameter B and used this to drive the SMC filter conditioned on each of the different values for $\boldsymbol{\beta}$; this creates a different shaped likelihood function. Effectively, in this case, the marginal likelihood becomes highly constrained in a small region of parameter space.

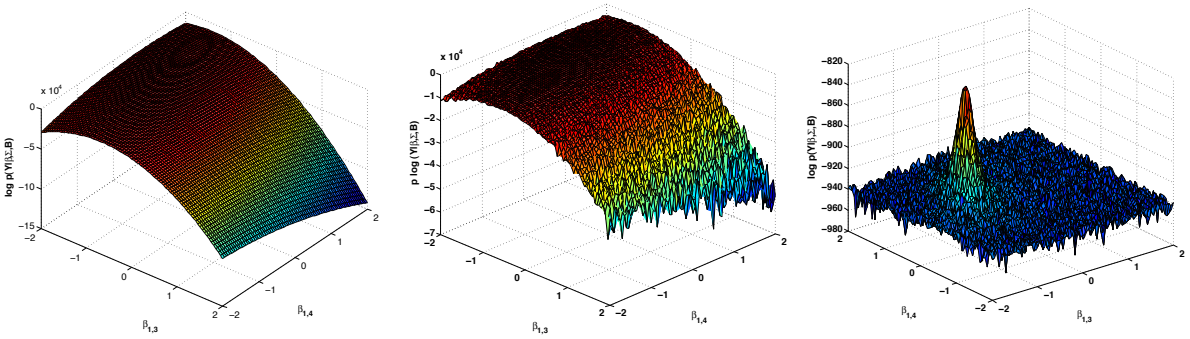


Figure 6.11: Introducing sampling of parameters that subsequently drive the embedded SMC filters will distort the likelihood surfaces. These plots show the surface for $p(Y|\boldsymbol{\beta}, \Sigma, B)$ either taking Σ as known (left plot) or sampling it exactly from its posterior distribution (centre plot) or taking Σ as known and sampling B. This may partially explain the observation that estimating more parameters in a PMMH algorithm generally makes the algorithms less stable and more difficult to tune. The rightmost plot shows the effect on the marginal likelihood surface calculated using the SMC algorithm when parameter B is sampled before each evaluation, conditional on the current value of $\boldsymbol{\beta}$.

6.4.4 Recovering Noise

As is clear from several of the examples given so far the recovery of the noise in the observation and latent systems is more or less effective dependent on the signal-to-noise ratio of the two parts of the system. For example, in a situation where the signal-to-noise ratio is high (i.e. there is more noise in the latent system than the observation system) it is possible to effectively recover the latent states, and from these estimate the noise level; an example is the left plot of Figure 5.8. However if the noise in both parts of the system is equally matched the filters are unable to completely capture the variations in the latent system and thus the reconstructed latent states do not contain the full complement of noise from the true latent system. In more extreme situations, if the observation noise is much greater than the latent system noise, only a fraction of the variations in the latent system are captured by the filters, see for example the right plot of Figure 5.8.

6.5 Model $\mathbf{M}_1^*$: Dynamic Non-Linear Latent Process

The PMMH algorithm is suitable for running on non-linear systems, which are outside the scope of application of the Kalman filter and the RB-PMMH. As an example, a non-linear dynamic latent system with equation:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + C\boldsymbol{\mu}_{t-1} + D\boldsymbol{\mu}_{t-1}^2 + \boldsymbol{w}_t \tag{6.1}$$

was simulated with parameter $C = 0.07 \times \mathbb{I}_n$, $D = 0.07 \times \mathbb{I}_n$, $\boldsymbol{\mu}_0 = [0.1\ 0.1\ 0.1\ 0.1]'$ and noise covariances $\Sigma_w = 0.1 \times \mathbb{I}_n$ and $\Sigma = 0.5 \times \mathbb{I}_n$. Examples of the performance of the SMC algorithm (running with $2^7 = 128$ particles) at recovering the latent states of this model are plotted in Figure 6.12 below. Samples from the PMMH algorithm (scheme I) are shown in Figure 6.13. The observation equations were from the Sugita rank 1 model $\mathbf{S}_1$.

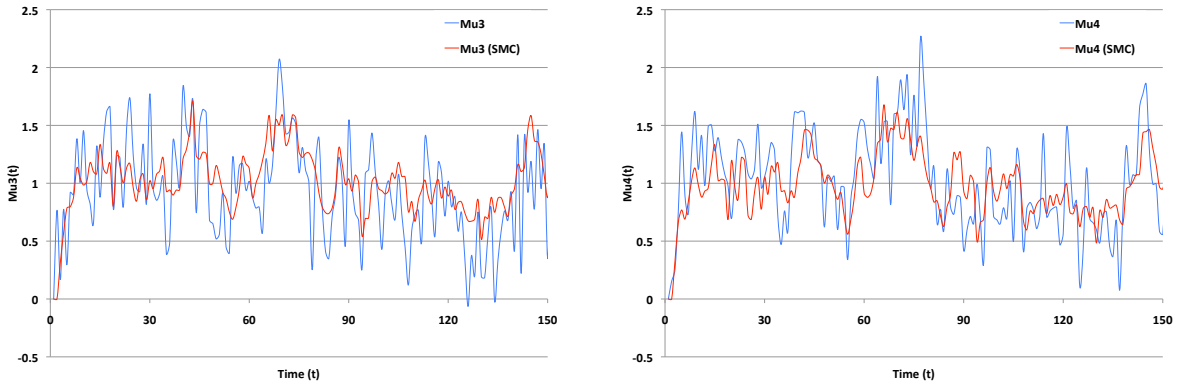


Figure 6.12: The plots show the latent states ($\mu_{3,t}$ and $\mu_{4,t}$) from the non-linear system recovered using the SMC filter.

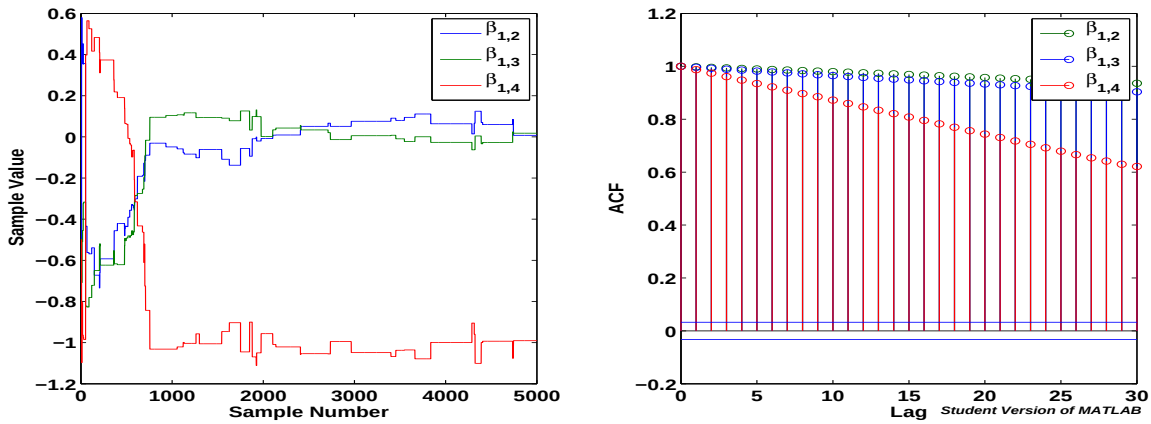


Figure 6.13: Plots showing the samples for $\boldsymbol{\beta}$ generated by a short chain (5000 samples) run on the non-linear model

The samples plot demonstrates that in the non-linear case we can use the PMMH algorithm for the joint estimation of static parameters and latent covariance states. Although the mixing is poor, the samples are clearly coming from the correct distribution for the $\boldsymbol{\beta}$ parameter. Further work is required to improve the tuning and mixing for this model.

6.6 Model $\mathbf{M}_2$: Dynamic Latent Covariance Process

Before demonstrating the operation of the PMMH sampler for the model with dynamic latent stochastic volatility observation error covariance, it is important to verify that the embedded SMC filter can recover the latent states $\boldsymbol{\mu}_t$ and the latent variances which are following the square Bessel process. The examples in Figure 6.14 show the filtered values of $\boldsymbol{\mu}_{1,t}$ and σ_t for three different values of parameter λ_0 in the square Bessel process driving the observation noise. In all the examples the latent states follow the static system:

$$\boldsymbol{\mu}_t = \boldsymbol{\mu}_0 + \boldsymbol{w}_t$$

with latent state noise covariance having value $\Sigma_w = 0.5 \times \mathbb{I}_n$, and $\boldsymbol{\mu}_0 = [0.1\ 0.1\ 0.1\ 0.1]'$.

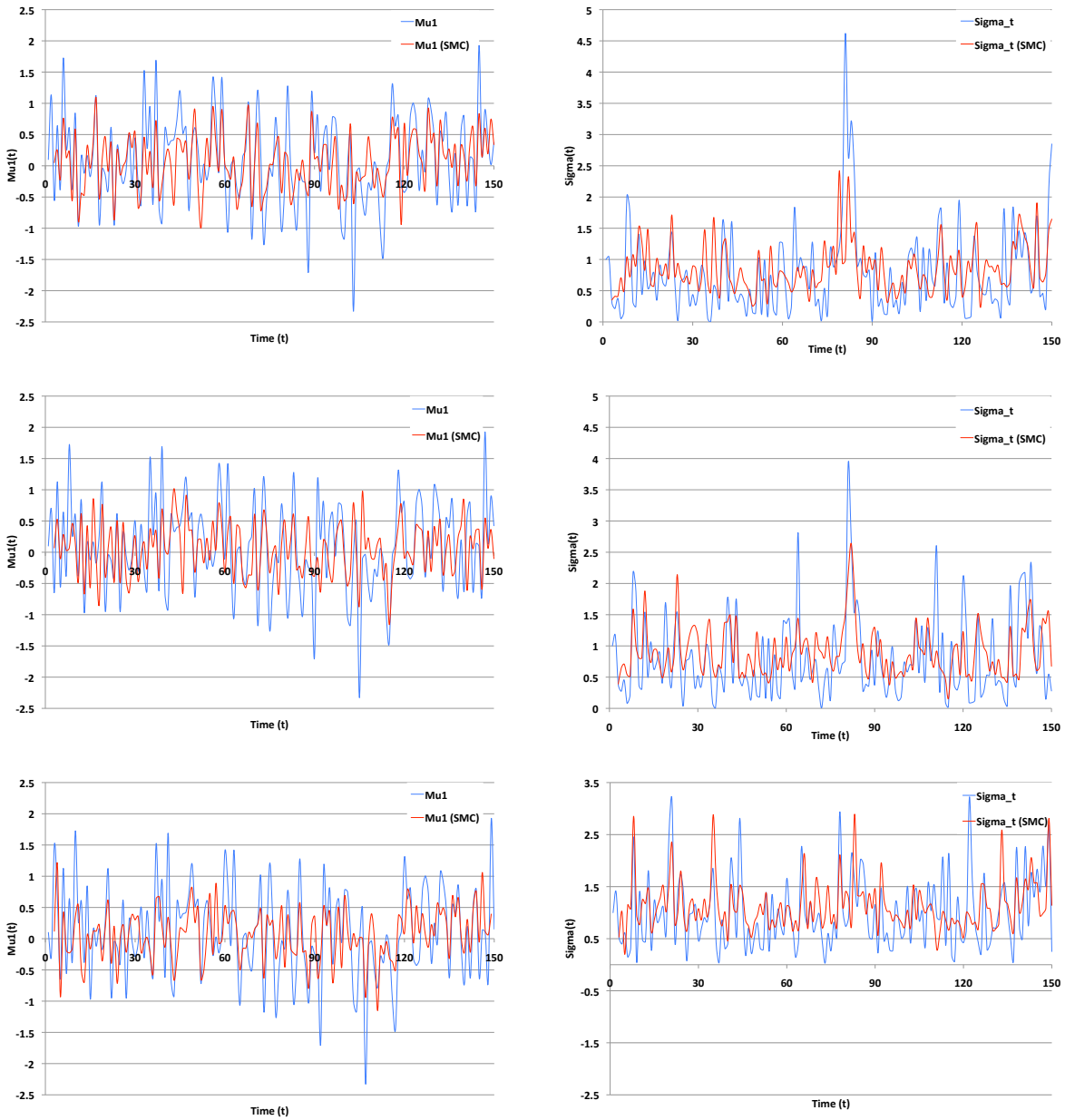


Figure 6.14: The left plots show the latent states (μ_t's) and the right plots show the observation noise variances (σ_t's) recovered using a 5-dimensional SMC filter. The top row has $\lambda_0 = 0.25$, the middle row has $\lambda_0 = 0.5$ and the bottom row has $\lambda_0 = 1$ resulting in observation noise processes with different means.

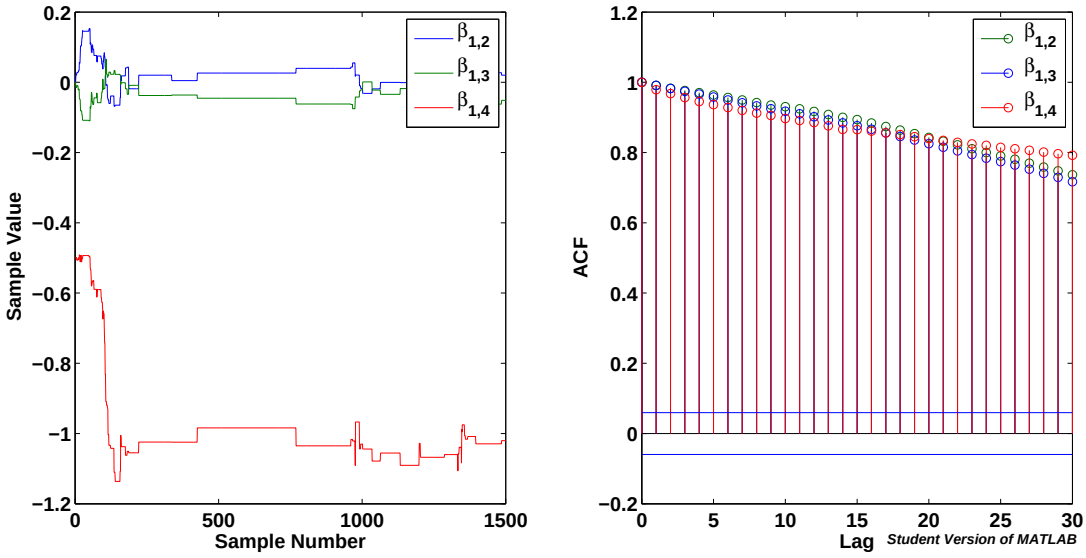


Figure 6.15: Plot showing the samples of $\boldsymbol{\beta}$ produced by running the PMMH algorithm in Scheme Ia; i.e. it is targeting $p(\boldsymbol{\beta}, \mathbf{z}_{1:T}|\mathbf{y}_{1:T})$.

The algorithm appears to correctly converge towards the true, known, components of $\boldsymbol{\beta}$, as shown in Figure 6.15. This example is limited due time/processor constraints which prevented a longer example from being run. The examples in Figure 6.14 demonstrate that augmenting the SMC filter to target both latent system states and observation noise covariances is achievable in practice. Therefore extensions to the joint estimation of latent states, static parameters and latent dynamically evolving covariances in new classes of flexible CVAR models is likely to be achievable too.

Chapter 7

Conclusions & Future Work

This thesis has explored the design and implementation of state-of-the-art PMCMC algorithms for the purposes of jointly estimating latent states and static parameters in the challenging matrix-variate context of cointegrated VAR models. Several classes of latent system have been explored; static, linear dynamic, and non-linear dynamic. Also, a class of models with dynamic covariance structure has been described, and basic estimation demonstrated.

In the case of the static latent systems the PMCMC algorithms display generally good mixing properties, and are able to recover known parameter values for $\boldsymbol{\beta}, \Sigma$ and B adequately. Two versions of the algorithm were coded up: a full PMMH algorithm with embedded particle filter, and a Rao-Blackwellised version which used a Kalman filter to perform the filtering and estimate the marginal likelihoods used in the PMMH acceptance ratio. The RBPMMH version is much faster to run than the full PMMH sampler, but its use is limited to state-space systems with linear equations.

A dynamic linear VAR system was used to test the PMMH and RBPMMH algorithms, and slightly modified sampling schemes (I, II and III) allowed the joint estimation of further parameters and latent states. For this category of models it is possible to sample from exact posterior distributions, not only for static parameters Σ and B, but also from the conditional posteriors of the static parameters of the *latent system* given the recovered latent states $F|\mathbf{z}_{1:T}$ and $\Sigma_w|\mathbf{z}_{1:T}$. Incorporating more sampled parameters into the filters driving the estimation alters the likelihood surface; for these schemes we may achieve superior results using the Particle Gibbs Algorithm, which embeds a *conditional* SMC filter.

The project represents the methodological development stage of a larger piece of work, which will apply the samplers that have been created to estimation of the parameters and latent states in extended classes of *dynamic* CVAR models. A simple example of the estimation for a model following a latent square Bessel process in the dynamic covariance structure of the observation errors has been given. It has been demonstrated that recovering the latent square Bessel process volatilities is possible with an augmented SMC filter, and that the marginal likelihood from such a filter can be used to drive a PMCMC algorithm to perform joint inference of static parameters and latent dynamic states.

Appendix A

Johansen Procedure Derivation

We proceed by minimizing the log-likelihood, Equation (2.11), with respect to the Ψ matrix, keeping all other parameters fixed. The matrix derivative of Equation (2.11), ignoring the constant term, is defined by,

$$\begin{aligned}\frac{\partial \ln L(\alpha,\beta,\Psi,\Omega)}{\partial \Psi_{kl}} &= \frac{\partial}{\partial \Psi_{kl}}\left\{-\frac{T}{2}\ln|\Omega| - \frac{1}{2}\sum_{t=1}^{T}(Z_{0t,\alpha} - (\alpha\beta' Z_{1t})_\alpha + \Psi_{\alpha\gamma} Z_{2t,\gamma})\right. \\ &\quad \left.\cdot \quad \Omega^{-1}_{\alpha\beta}(Z_{0t,\beta} - (\alpha\beta' Z_{1t})_\beta + \Psi_{\beta\delta} Z_{2t,\delta})\right\} \\ &= \frac{1}{2}\sum_{t=1}^{T}\left\{\left(Z_{0t,\alpha} - (\alpha\beta' Z_{1t})_\alpha + \Psi_{\alpha\gamma} Z_{2t,\gamma}\right)\Omega^{-1}_{\alpha k} Z_{2t,\gamma} + \right. \\ &\quad \left. Z_{2t,l}\Omega^{-1}_{k\beta}\left(Z_{0t,\beta} - (\alpha\beta' Z_{1t})_\beta + \Psi_{\beta\delta} Z_{2t,\delta}\right)\right\}. \end{aligned} \tag{A.1}$$

If we assume that the log-likelihood function is convex, the minima of Equation (A.1) is given by,

$$\sum_{t=1}^{T}(Z_{0t} - \alpha\beta' Z_{1t} + \Psi Z_{2t}) Z'_{2t} = 0. \tag{A.2}$$

Solving for Ψ, we substitute the results back into Equation (2.11) and proceed to minimise the log-likelihood with respect to α with β and Ω held fixed. By defining the product moment matrices,

$$M_{ij} = \frac{1}{T}\sum_{t=1}^{T} Z_{it} Z'_{jt} \quad \text{i, j} = 0,\ 1,\ 2,$$

the solution of Equation (A.2) with respect to Ψ is given by,

$$\hat{\Psi} = M_{02} M_{12}^{-1} - \alpha\beta' M_{12} M_{22}^{-1}. \tag{A.3}$$

Next the short run transitory effects, $\hat{\Psi} Z_{2t}$, in Equation (2.9), are removed. By using the Frisch-Waugh-Lovell theorem [Lovell, 2005], we are able to define the following auxiliary regressions,

$$\begin{aligned} Z_{0t} &= M_{02} M_{22}^{-1} Z_{2t} + R_{0t}, \\ Z_{1t} &= M_{12} M_{22}^{-1} Z_{2t} + R_{1t}. \end{aligned}$$

The residuals of the auxiliary regressions can now be used to define a *concentrated* model,

$$
\begin{aligned}
R_{0t} &= \alpha\beta' R_{1t} + \epsilon_t, & \epsilon_t &\sim MVN(0, \Sigma), \\
\epsilon_t &= R_{0t} - \alpha\beta' R_{1t}. & &
\end{aligned} \tag{A.4}
$$

The concentrated model is helpful for understanding both the statistical and economic properties of the VECM in Equation (2.8). The VECM model contains both short-run adjustment and intervention effects, whereas the concentrated model contains long-run adjustments [Juselius, 2006]. The log-likelihood of the concentrated model, Equation (A.4), can be expressed as,

$$
\ln L(\alpha, \beta, \Omega) = constants - \frac{T}{2}\ln|\Omega| - \frac{1}{2}\sum_{t=1}^{T}(R_{0t} - \alpha\beta' R_{1t})'\Omega^{-1}(R_{0t} - \alpha\beta' R_{1t}). \tag{A.5}
$$

Now we minimise Equation (A.5), with respect to α with Ω and β held fixed,

$$
\begin{aligned}
\frac{\partial \ln L(\alpha, \beta, \Omega)}{\partial \alpha_{kl}} &= \frac{\partial}{\partial \alpha_{kl}}\left\{-\frac{T}{2}\ln|\Omega| - \frac{1}{2}\sum_{t=1}^{T}\big(R_{0t,\alpha} - \alpha_{\alpha\gamma}(\beta' R_{1t})_\gamma\big)\Omega^{-1}_{\alpha\beta}\big(R_{0t,\beta} - \alpha_{\beta\delta}(\beta' R_{1t})_\delta\big)\right\} \\
&= \frac{1}{2}\sum_{t=1}^{T}\Big\{(R_{0t,\alpha} - \alpha_{\alpha\gamma}(\beta' R_{1t})_\gamma)\Omega^{-1}_{\alpha k}(\beta' R_{1t})_l + (\beta' R_{1t})_l\Omega^{-1}_{k\beta}\big(R_{0t,\beta} - \alpha_{\beta\delta}(\beta' R_{1t})_\delta\big)\Big\}.
\end{aligned} \tag{A.6}
$$

The minimum of Equation (A.7) corresponds to the solution of:

$$
\sum_{t=1}^{T}(R_{0t} - \alpha(\beta' R_{1t}))(\beta' R_{1t})' = 0. \tag{A.7}
$$

By defining the residual sum of squares as matrices,

$$
S_{ij} = \frac{1}{T}\sum_{t=1}^{T} R_{it}R_{jt}^{'} \quad \text{i, j = 0, 1}
$$

the solution of Equation (A.7) with respect to α is given by,

$$
\hat{\alpha} = S_{01}\beta(\beta' S_{11}\beta)^{-1}. \tag{A.8}
$$

Substituting Equation (A.8) into Equation (A.5), we minimise the result with respect to Ω with β held fixed. For convenience we will use the notation $M(\beta) = S_{01}\beta(\beta' S_{11}\beta)^{-1}\beta'$. Solving for Ω,

$$
\begin{aligned}
\frac{\partial \ln L(\beta, \Omega)}{\partial \Omega_{kl}} &= -\frac{T}{2}\Omega^{-1}_{kl} + \frac{1}{2}\sum_{t=1}^{T}(R_{0t,\alpha} - M(\beta)_{\alpha\gamma}R_{1t,\gamma})\Omega^{-1}_{\alpha k}\Omega^{-1}_{l\beta}(R_{0t,\beta} - M(\beta)_{\beta\delta}R_{1t,\delta}) \\
&= -\frac{T}{2}\Omega^{-1}_{kl} + \sum_{t=1}^{T}\Big\{\Omega^{-1}_{\alpha k}R_{0t,\alpha}R_{0t,\beta}\Omega^{-1}_{l\beta} - \Omega^{-1}_{\alpha k}R_{0t,\alpha}M(\beta)_{\beta\delta}R_{1t,\delta}\Omega^{-1}_{l\beta} \\
&\quad - \Omega^{-1}_{\alpha k}M(\beta)_{\alpha\gamma}R_{1t,\gamma}R_{0t,\beta}\Omega^{-1}_{l\beta} \\
&\quad + \Omega^{-1}_{\alpha k}M(\beta)_{\alpha\gamma}R_{1t,\gamma}R_{1t,\gamma}M(\beta)_{\beta\delta}R_{1t,\delta}\Omega^{-1}_{l\beta}\Big\}.
\end{aligned}
$$

In matrix form this reduces to,

$$\begin{aligned}\frac{\partial \ln L(\beta,\Omega)}{\partial \Omega} &= -\frac{T}{2}\Omega^{-1} + \frac{1}{2}\sum_{t=1}^{T}(\Omega^{-1})'\left\{R_{0t}R_{0t}' - R_{0t}R_{1t}'M(\beta)' - M(\beta)R_{1t}R_{0t}'\right. \\ &+ \left. M(\beta)R_{1t}R_{1t}'M(\beta)'\right\}(\Omega^{-1})' \\ &= -\frac{T}{2}\Omega^{-1} + -\frac{T}{2}(\Omega^{-1})' + \left\{S_{00} - S_{01}M(\beta)' - M(\beta)S_{10}\right. \\ &+ \left. M(\beta)S_{11}M(\beta)'\right\}(\Omega^{-1})'. \end{aligned} \tag{A.9}$$

Multiplying both sides of Equation (A.9) by Ω yields,

$$\Omega = S_{00} - S_{01}M(\beta)' - M(\beta)S_{10} + M(\beta)S_{11}M(\beta)'. \tag{A.10}$$

By noting that,

$$\begin{aligned} S_{01}M(\beta)' &= S_{01}S_{01}\beta(\beta'S_{11}\beta)^{-1}\beta'' \\ &= S_{01}\beta(\beta'S_{11}'\beta)^{-1'}\beta'S_{01}' \\ &= S_{01}\beta(\beta'S_{11}'\beta)^{-1}\beta'S_{01} \\ &= M(\beta)S_{10}, \end{aligned}$$

and

$$\begin{aligned} M(\beta)S_{11}M(\beta)' &= S_{01}\beta(\beta'S_{11}'\beta)^{-1}\beta'S_{11}\beta(\beta'S_{11}'\beta)^{-1}\ \beta'S_{10} \\ &= S_{01}\beta(\beta'S_{11}'\beta)^{-1}\beta'S_{01}, \end{aligned}$$

Equation (A.10), can be written as,

$$\hat{\Omega} = S_{00} - S_{01}\beta(\beta'S_{11}'\beta)^{-1}\beta'S_{01}. \tag{A.11}$$

For a multivariate normal distribution, up to an overall constant, the maximum possible value of the likelihood function is $|\Omega|^{-T/2}$, which is also denoted by,

$$\begin{aligned} L_{\max}^{-2/T}(\beta) &= |\Omega(\beta)| \\ &= |S_{00} - S_{01}\beta(\beta'S_{11}'\beta)^{-1}\beta'S_{01}|. \end{aligned} \tag{A.12}$$

Thus, the problem of maximising the likelihood function reduces to finding the matrix β which maximises the determinant of Equation (A.12). To find the determinant we make use of the expression,

$$M = \begin{pmatrix} S_{00} & S_{01}\beta \\ \beta'S_{01} & \beta'S_{01}\beta \end{pmatrix},$$

the determinant of M is:

$$\begin{aligned} |M| &= |S_{00}||\beta'S_{01}\beta - \beta'S_{10}S_{00}^{-1}S_{01}\beta| \\ &= |\beta'S_{01}\beta||S_{00} - S_{01}\beta(\beta'S_{11}\beta)^{-1}\beta'S_{10}|. \end{aligned} \tag{A.13}$$

The determinant of Ω is contained in Equation (A.13). Solving for $|\Omega|$ yields:

$$|\Omega| = |S_{00}|\frac{|\beta' S_{11}\beta - \beta' S_{10}S_{00}^{-1}S_{01}\beta|}{|\beta' S_{11}\beta|}$$
$$= |S_{00}|\frac{|\beta'(S_{11} - S_{10}S_{00}^{-1}S_{01})\beta|}{|\beta' S_{11}\beta|}.$$

The values of β which maximise the likelihood function are the solutions to,

$$(S_{11} - S_{10}S_{00}^{-1}S_{01})\mathbf{v}^i = p^i S_{11}\mathbf{v}^i,$$

or equivalently for $\lambda^i = 1 - p^i$,

$$S_{10}S_{00}^{-1}S_{01}\mathbf{v}^i = \lambda^i S_{11}\mathbf{v}^i. \tag{A.14}$$

The vectors, $\mathbf{v}^i$, represent cointegration relations. By choosing the normalisation $\mathbf{v}^i S_{11}\mathbf{v}^j = \mathbb{I}_n$ (if $i = j$ and 0 otherwise) then $\beta' S_{10}S_{00}^{-1}S_{01}\beta = \text{diag}(\lambda_1, \lambda_2, \ldots)$. With this choice of β, the maximum likelihood function is then,

$$L_{\max}^{-2/T} = \left(|S_{00}|\prod_{i=1}^{r}(1 - \hat{\lambda}_i)\right). \tag{A.15}$$

In summary, the Johansen Maximum Likelihood method produces estimates of the parameters of a Vector Error Correction Model (VECM) of the form given in Equation (2.8). The parameter estimates and the resulting likelihood function are summarised in Table (A.1), where, r, represents the cointegration rank and the other symbols are as defined previously.

Estimator	Form
$\hat{\alpha}$	$S_{01}\beta(\beta' S_{11}\beta)^{-1}$
$\hat{\beta}$	$\left[\mathbf{v}^1, \ldots, \mathbf{v}^r\right]' S_{11}^{-1/2}$
$\hat{\Omega}$	$S_{00} - S_{01}\beta(\beta' S_{11}'\beta)^{-1}\beta' S_{01}$
$\hat{\Psi}$	$M_{02}M_{12}^{-1} - \alpha\beta' M_{12}M_{22}^{-1}$
Maximum Likelihood	$\left(\lvert S_{00}\rvert \prod_{i=1}^{r}(1 - \hat{\lambda}_i)\right)$

Table A.1: Summary of Estimates from the Johansen Maximum Likelihood Method

Appendix B

Selected Code Components

B.1 Bayesian CVAR Probabilities

This object is used by the samplers to calculate, and sample from the priors and posteriors at various stages.

```
classdef BayesianCVARProbabilities < handle
    % BayesianCVARProbabilities Class
    % This file contains code to calculate the CVAR Likelihood, Prior and Posterior
    % Probabilities under a Bayesian Hierarchical Model

    % PROPERTIES (Public)
    properties(GetAccess = public, SetAccess = protected)

        dimension   % Dimensionality of the system process
        t           % t=T-p+1 (where T is the number of observations and p is the number of
 lags)
        k           % k=n(p-1)+r+1
        c           %
        % The Stored ECM model object & data:
        ECMModelObject
        ECMParStruct
        rank
        Y       % Transformed Data = (Xt-Xt-1)
        YBlock  % This is Y as it appears in the literature (del(x_p), ... del(x_T))'
        X       % Data
        XBlock  % This is X as it appears in the literature (augmented with (t*n) block of
ones)...
        ZBlock  % This is Z from the literature dimension (t*n)
        Gamma   % This is the matrix with the mean level, and the Psi matrices
        Mu      % The Mean level from the System Process
        Psi     % The Psi Matrices
        B

        % Transformed Datasets for easy incorporation into the marginalised
        % posterior formulae:
        W
        WW
        B_Hat
        B_Star
        A_Star
        S_Hat
        S_Star

        % The Heirarchical Prior Parameters (Observation Model)
        % Beta ~ N(BetaBar, Q (*) H^-1)
        Beta_Bar     % The prior mean for Beta
        Beta_Q       % (r*r) PSD matrix
        Beta_H       % (n*n) PSD matrix

        % Sigma ~ IW(S,h)
        S           % (n*n) PSD matrix
        h           % DOF

        % B|Sigma ~ N(P, Sigma (*) A^-1)
        Beta_Hat
        P           % The prior mean for B (k*n)
        A           % (k*n) PSD matrix
        AInv

        % The Heirarchical Prior Parameters (System Model)

        % SigmaW ~ IW(Sw,hw)
        Sw          % (n*n) PSD 'mean' matrix
        hw          % IW DOF
```

```
        % F|SigmaW ~ N(F_Bar, SigmaW * A_F^-1)
        F_Bar       % (n+1)*n prior mean for F
        A_F         % A_F {(n+1)*(n+1)} PSD covariance matrix
    end

    % PROPERTIES (Private)
    properties(GetAccess = private, SetAccess = protected)

        YminusWB_Hat
        PminusB_Hat
        BminusB_Hat
        BminusB_Star
        posteriorArg
    end


    % METHODS (Public)
    methods(Access = public)

        % Function: Construct Bayesian CVAR Probability Object
        % Input:    ecmModelObject
        % Output:   None
        % NOTES:
        function this = BayesianCVARProbabilities(ecmModelObject, ecmParStruct)
            % BayesianCVARProbabilities Default Constructor
            this.ECMModelObject = ecmModelObject;
            this.ECMParStruct = ecmParStruct;
            this.rank = size(this.ECMParStruct.MeasurementProcessParameters.Beta,2);
            this.dimension  = size(this.ECMParStruct.SystemProcessParameters.InitialState,2
);
            this.Psi        = this.ECMParStruct.MeasurementProcessParameters.LagMatrices;
            if(size(this.Psi,2)==0)
                this.t          = this.ECMParStruct.NumberObservations...
                    - this.ECMParStruct.MeasurementProcessParameters.NumberLags + 1;
            else
                this.t          = this.ECMParStruct.NumberObservations...
                    - this.ECMParStruct.MeasurementProcessParameters.NumberLags;
            end;
            this.k              = this.dimension...
                * (this.ECMParStruct.MeasurementProcessParameters.NumberLags-1)+this.rank +
 1;
            this.Mu         = ...
sum(this.ECMParStruct.SystemProcessParameters.SystemProcessParameters.TransitionMatrices(:,
:,1),2);
            this.Gamma      = [this.Mu this.Psi(:,:,1)]';
            this.B          = [this.Gamma; this.ECMParStruct.MeasurementProcessParameters.A
lpha'];
            this.X      = this.ECMModelObject.getDataSet(1, 0);
            this.Y      = [this.X(2:this.t,:)-this.X(1:this.t-1,:)];
            tBlock            = ones(this.t-1,1);
            % Note: at the moment this can only handle p=1 or 2:
            if this.ECMParStruct.MeasurementProcessParameters.NumberLags == 1
                this.XBlock = [tBlock];
            else
                this.XBlock = [tBlock this.Y(1:this.ECMParStruct.NumberObservations-1,:)];
            end
            this.YBlock = this.Y(1:this.ECMParStruct.NumberObservations-1,:);
            this.ZBlock = this.X(1:this.ECMParStruct.NumberObservations-1,:);
        end
```

```
        % Function: SetBetaPriorParams
        % Input:    BetaBar, Q, H
        % Output:
        % NOTES:
        function SetBetaPriorParams(this,BetaBar, Q, H)
            this.Beta_Bar = BetaBar;
            this.Beta_Q = Q;
            this.Beta_H = H;
        end

        % Function: SetSigmaPriorParams
        % Input:    S, h
        % Output:
        % NOTES:
        function SetSigmaPriorParams(this, S, h)
            this.S = S;
            this.h = h;
        end

        % Function: SetBPriorParams
        % Input:    P, A
        % Output:
        % NOTES:
        function SetBPriorParams(this, P, A, Beta_Hat)
            this.P = P;
            this.A = A;
            this.AInv = inv(this.A);
            this.Beta_Hat = Beta_Hat;
        end

        % Function: SetFPriorParams
        % Input:    F_Bar, A_F
        % Output:
        % NOTES:
        function SetFPriorParams(this, F_Bar, A_F)
            this.F_Bar = F_Bar;
            this.A_F = A_F;
        end

        % Function: SetSigmaWPriorParams
        % Input:    Sw, hw
        % Output:
        % NOTES:
        function SetSigmaWPriorParams(this, Sw, hw)
            this.Sw = Sw;
            this.hw = hw;
        end


        % Function: PriorProbBeta
        % Input:    Beta
        % Output:   p(Beta)
        % NOTES:
        function [pBeta] = PriorProbBeta(this, Beta)
            preFactor = det(this.Beta_Q)^(-this.dimension/2)*det(this.Beta_H)^(this.rank/2)
;
```

```
            argument = -0.5*trace(this.Beta_Q\(Beta-this.Beta_Bar)'*this.Beta_H*(Beta-this.
Beta_Bar));
            pBeta = preFactor * exp(argument);
        end

        % Function: logPriorProbBeta
        % Input:    Beta
        % Output:   log p(Beta)
        % NOTES:
        function [logPBeta] = logPriorProbBeta(this, Beta)
            preFactor = det(this.Beta_Q)^(-this.dimension/2)*det(this.Beta_H)^(this.rank/2)
;
            argument = -0.5*trace(this.Beta_Q\(Beta-this.Beta_Bar)'*this.Beta_H*(Beta-this.
Beta_Bar));
            logPBeta = log(preFactor) + argument;
        end

        % Function: logPriorProbSigma
        % Input:    Sigma
        % Output:   log p(Sigma)
        % NOTES:
        function [logPSigma] = logPriorProbSigma(this, Sigma)
            preFactor = (this.h/2)*log(det(this.S))-(this.h+this.dimension+1)*log(det(Sigma
));
            argument = -0.5*trace(Sigma\this.S);
            logPSigma = preFactor + argument;
        end

        % Function: logPriorProbB
        % Input:    B
        % Output:   log p(B)
        % NOTES:
        function [logPSigma] = logPriorProbB(this, Sigma, B)
            preFactor   = -(this.k/2)*log(det(Sigma))+(this.dimension/2)*log(det(this.A));
            BminusP     = B-this.P;
            argument    = -0.5*trace(Sigma\BminusP'*this.A*BminusP);
            logPSigma   = preFactor + argument;
        end

        % Function: GetZBlock
        % Input:    dsNum (data set number)
        % Output:
        % NOTES:
        function [zBlock] = GetZBlock(this)
            zBlock = this.ZBlock;
        end

        % Function: GetXBlock
        % Input:    dsNum (data set number)
        % Output:
        % NOTES:
        function [xBlock] = GetXBlock(this)
            xBlock = this.XBlock;
        end

        % Function: GetYBlock
        % Input:    dsNum (data set number)
        % Output:
        % NOTES:
        function [yBlock] = GetYBlock(this)
            yBlock = this.YBlock;
```

```
        end

        % Function: logPBetaGivenY, marginal posterior probability of Beta
        % given the observations Y.
        % Input:    Individual Beta Parameters:
        % Output:   log[p(Beta|Y)]
        % NOTES:
        function [logPBetaGivenY] = logP_Beta_GivenY(this, Beta)
            this.W           = [this.XBlock this.ZBlock * Beta];
            this.WW          = this.W'*this.W;
            this.B_Hat       = this.WW\(this.W'*this.YBlock);
            this.A_Star      = this.A + this.WW;
            this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
            this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
            this.PminusB_Hat = this.P - this.B_Hat;
            this.S_Star      = this.S + this.S_Hat...
                + this.PminusB_Hat'/(this.AInv + inv(this.WW))*this.PminusB_Hat;
            logPBetaGivenY = this.logPriorProbBeta(Beta) + ...
                (-(this.t+this.h+1)/2) * log(det(this.S_Star)) + ...
                (-this.dimension/2)*log(det(this.A_Star));
            % Or set the prior to 1 (for testing):
            % logPBetaGivenY = (-(this.t+this.h+1)/2) * log(det(this.S_Star)) + ...
            %   (-this.dimension/2)*log(det(this.A_Star));

        end

        % Function: logP_BetaSigmaB_GivenY, log posterior probability
        % of Beta, Sigma and B given the observations Y.
        % Input:    Beta, Sigma, Y
        % Output:   log[p(Beta,Sigma,B|Y)]
        % NOTES:
        function [logP_BetaSigmaB_GivenY] = logP_BetaSigmaB_GivenY(this, Beta, Sigma, B)
            this.W           = [this.XBlock this.ZBlock * Beta];
            this.WW          = this.W'*this.W;
            this.B_Hat       = this.WW\(this.W'*this.YBlock);
            this.A_Star      = this.A + this.WW;
            this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
            this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
            this.PminusB_Hat = this.P - this.B_Hat;
            this.S_Star      = this.S + this.S_Hat...
                + this.PminusB_Hat'*((this.AInv + inv(this.WW))\this.PminusB_Hat);
            this.B_Star      = (this.A + this.WW)\(this.A * this.P + this.WW*this.B_Hat);
            this.BminusB_Star= B-this.B_Star;
            this.c                  = this.t+this.k+this.h+this.dimension+1;
            this.posteriorArg       = this.S_Star+this.BminusB_Star'*this.A_Star*this.Bminu
sB_Star;
            logP_BetaSigmaB_GivenY = this.logPriorProbBeta(Beta) + ...
                (-this.c/2) * log(det(Sigma)) + ...
                (-1/2)*trace(Sigma\this.posteriorArg);
            % Or set the prior to 1 (for testing):
            % logP_BetaSigmaB_GivenY = (-this.c/2) * log(det(Sigma)) + ...
            %    (-1/2)*trace(Sigma\this.posteriorArg);
        end

        % Function: logP_Y_GivenBetaSigmaB, log Likelihood
        % of Y given Beta, Sigma and B.
        % Input:    Beta, Sigma, B, Y
        % Output:   log[p(Y|Beta,Sigma,B)]
        % NOTES:
        function [logP_Y_Given_BetaSigmaB] = logP_Y_Given_BetaSigmaB(this, Beta, Sigma, B)
            this.W           = [this.XBlock this.ZBlock * Beta];
            this.WW          = this.W'*this.W;
```

```
        this.B_Hat       = this.WW\(this.W'*this.YBlock);
        this.A_Star      = this.A + this.WW;
        this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
        this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
        this.BminusB_Hat = B - this.B_Hat;
        likelihoodArg       = this.S_Hat+this.BminusB_Hat'*this.WW*this.BminusB_Hat;
        logP_Y_Given_BetaSigmaB = (-this.t/2) * log(det(Sigma)) + ...
            (-1/2)*trace(Sigma\likelihoodArg);
    end

    % Function: (1) sample_SigmaGivenBetaY
    % Input:    Beta
    % Output:   An exact sample from sigma_xx~(Sigma|Beta, Y)
    % NOTES:
    function [Sigma] = sample_SigmaGivenBetaY(this, Beta)
        this.W           = [this.XBlock this.ZBlock * Beta];
        this.WW          = this.W'*this.W;
        this.B_Hat       = this.WW\(this.W'*this.YBlock);
        this.A_Star      = this.A + this.WW;
        this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
        this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
        this.PminusB_Hat = this.P - this.B_Hat;
        this.S_Star      = this.S + this.S_Hat...
            + this.PminusB_Hat'/(this.AInv + inv(this.WW))*this.PminusB_Hat;
        dof = this.t + this.h;
        Sigma = iwishrnd(this.S_Star, dof);
    end

    % Function: (2) sample_BGivenBetaSigmaY
    % Input:    Beta, Sigma
    % Output:   An exact sample from B~(B|Beta, Sigma, Y)
    % NOTES:    NOTE: This function does not take account of the latent
    % path-space!
    function [B] = sample_BGivenBetaSigmaY(this, Beta, Sigma)
        this.W           = [this.XBlock this.ZBlock * Beta];
        this.WW          = this.W'*this.W;
        this.B_Hat       = this.WW\(this.W'*this.YBlock);
        this.A_Star      = this.A + this.WW;
        this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
        this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
        this.PminusB_Hat = this.P - this.B_Hat;
        this.S_Star      = this.S + this.S_Hat...
            + this.PminusB_Hat'/(this.AInv + inv(this.WW))*this.PminusB_Hat;
        this.B_Star      = (this.A + this.WW)\(this.A * this.P + this.WW*this.B_Hat);
        Vec_B_Star = reshape(this.B_Star, this.k*this.dimension, 1);
        Kron_Sigma_AStar = kron(Sigma, inv(this.A_Star));
        Vec_B = mvnrnd(Vec_B_Star, Kron_Sigma_AStar);
        B = reshape(Vec_B, this.k, this.dimension);
    end

    % Function: (3) sample_BGivenBetaY
    % Input:    Beta,
    % Output:   An exact sample from B~(B|Beta, Y)
    % NOTES:
    function [B] = sample_BGivenBetaY(this, Beta)
        this.W           = [this.XBlock this.ZBlock * Beta];
        this.WW          = this.W'*this.W;
        this.B_Hat       = this.WW\(this.W'*this.YBlock);
        this.A_Star      = this.A + this.WW;
        this.YminusWB_Hat= this.YBlock-this.W*this.B_Hat;
        this.S_Hat       = this.YminusWB_Hat'*this.YminusWB_Hat;
```

```
            this.PminusB_Hat = this.P - this.B_Hat;
            this.S_Star      = this.S + this.S_Hat...
                + this.PminusB_Hat'/(this.AInv + inv(this.WW))*this.PminusB_Hat;
            this.B_Star      = (this.A + this.WW)\(this.A * this.P + this.WW*this.B_Hat);
            B = MatricvariateStudent(this.B_Star, this.A_Star, this.S_Star, this.t);
        end

        % Function: (4) sample_SigmaGivenBetaY_tilde
        % Input:    Beta, M(Latent Variables)
        % Output:   An exact sample from Sigma~(Sigma|Beta, M)
        % NOTES:    This works by subtracting off the latent variables Mu's to
        % generate Y_tilde internally.
        function [Sigma] = sample_SigmaGivenBetaY_tilde(this, Beta, M)
            % Translate the Y variables by subtracting the Latent paths:
            Y_T             = this.YBlock - M(2:this.ECMParStruct.NumberObservations,:);
            tBlock          = ones(this.t-1,1);
            % Note: at the moment this can only handle p=1 or 2:
            if this.ECMParStruct.MeasurementProcessParameters.NumberLags == 1
                XBlock_T = [tBlock];
            else
                XBlock_T = [tBlock Y_T(1:this.ECMParStruct.NumberObservations-1,:)];
            end
            YBlock_T = Y_T(1:this.ECMParStruct.NumberObservations-1,:);
            ZBlock_T = this.X(1:this.ECMParStruct.NumberObservations-1,:);
            W             = [XBlock_T ZBlock_T * Beta];
            WW            = (W'*W);
            B_Hat         = WW\(W'*YBlock_T);
            W_B_Hat       = W*B_Hat;
            YminusWB_Hat  = YBlock_T-W_B_Hat;  %#ok<*PROP>
            YminusWB_Hat  = YminusWB_Hat(2:end,:);
            S_Hat         = YminusWB_Hat'*YminusWB_Hat;
            PminusB_Hat   = this.P - B_Hat;
            S_Star        = this.S + S_Hat + PminusB_Hat'/(this.AInv + inv(WW))*PminusB_Hat
;
            % Sample from an inverse Wishart distribution:
            dof = this.t + this.h - 2;
            Sigma = iwishrnd(S_Star, dof);
        end

        % Function: (5) sample_BGivenBetaY_tilde
        % Input:    Beta, Z
        % Output:   An exact sample from B~(B|Beta, Y~)
        % NOTES:    This takes in the discovered latent variables (Mu's) and
        % samples exactly from ~(B|Beta_xx,Y~), where Y~=Y-Z
        function [B] = sample_BGivenBetaY_tilde(this, Beta, M)
            % Translate the Y variables by subtracting the Latent paths:
            Y_T             = this.YBlock - M(2:this.ECMParStruct.NumberObservations,:);
            tBlock          = ones(this.t-1,1);
            % Note: at the moment this can only handle p=1 or 2:
            if this.ECMParStruct.MeasurementProcessParameters.NumberLags == 1
                XBlock_T = [tBlock];
            else
                XBlock_T = [tBlock Y_T(1:this.ECMParStruct.NumberObservations-1,:)];
            end
            YBlock_T = Y_T(1:this.ECMParStruct.NumberObservations-1,:);
            ZBlock_T = this.X(1:this.ECMParStruct.NumberObservations-1,:);
            W             = [XBlock_T ZBlock_T * Beta];
            WW            = (W'*W);
            B_Hat         = WW\(W'*YBlock_T);
            W_B_Hat       = W*B_Hat;
            YminusWB_Hat  = YBlock_T-W_B_Hat;
            % Remove the first element - since it is often inaccurate...
```

```
            YminusWB_Hat    = YminusWB_Hat(2:end,:);
            S_Hat           = YminusWB_Hat'*YminusWB_Hat;
            PminusB_Hat     = this.P - B_Hat;
            S_Star          = this.S + S_Hat + PminusB_Hat'/(this.AInv + inv(WW))*PminusB_H
at;
            A_Star          = this.A + WW;
            B_Star          = (A_Star)\(this.A * this.P + WW*B_Hat);
            B               = MatricvariateStudent(B_Star, A_Star, S_Star, this.t-2);
        end

        % Function: (6) sample_SigmaW
        % Input:    Latents
        % Output:   An exact sample from SigmaW~(.|M)
        % NOTES:    This takes in the discovered latent variables (Mu's) and
        % samples exactly from ~(SigmaW|M)
        function [SigmaW] = sample_SigmaW(this, M)
            tBlock          = ones(this.t,1);
            MBlock          = [zeros(1,this.ECMModelObject.dimension);...
                M(1:this.ECMParStruct.NumberObservations-1,:)];
            X               = [tBlock MBlock];
            F_Hat           = (X'*X)\X'*M;
            MminusXF_Hat    = M-X*F_Hat;
            MminusXF_Hat    = MminusXF_Hat(2:end,:);
            Sw_Hat          = MminusXF_Hat'*MminusXF_Hat;
            A_F_Star        = this.A_F+(X'*X);
            F_Star          = A_F_Star\(this.A_F*this.F_Bar+(X'*X)*F_Hat);
            F_Comb          = F_Hat'*(X'*X)*F_Hat + this.F_Bar'...
                *this.A_F*this.F_Bar - F_Star'*A_F_Star*F_Star;
            Sw_Star         = this.Sw + Sw_Hat + F_Comb;
            Sw_Star         = (Sw_Star+Sw_Star')/2;
            dof             = this.t + this.hw;
            SigmaW          = iwishrnd(Sw_Star, dof);
        end

        % Function: (7) sample_F
        % Input:    Latents
        % Output:   An exact sample from F~(.|M)
        % NOTES:    This takes in the discovered latent variables (Mu's) and
        % samples exactly from ~(F|M)
        function [F] = sample_F(this, M)
            tBlock          = ones(this.t,1);
            MBlock          = [zeros(1,this.ECMModelObject.dimension);...
                M(1:this.ECMParStruct.NumberObservations-1,:)];
            X               = [tBlock MBlock];
            F_Hat           = (X'*X)\X'*M;
            Sw_Hat          = (M-X*F_Hat)'*(M-X*F_Hat);
            A_F_Star        = this.A_F+(X'*X);
            F_Star          = A_F_Star\(this.A_F*this.F_Bar+(X'*X)*F_Hat);
            Sw_Star         = this.Sw + Sw_Hat + F_Hat'*(X'*X)*F_Hat...
                + this.F_Bar'*this.A_F*this.F_Bar - F_Star'*A_F_Star*F_Star;
            dof             = this.t;
            F               = MatricvariateStudent(F_Star, A_F_Star, Sw_Star, dof);
        end




    end


end
```

B.2 PMMH Sampler

This is the code for the adaptive PMMH sampler.

```
classdef PMMHSampler < handle
    % ParticleMarginalMetropolis(Hastings)Sampler Class
    % This file contains code for the PMMH Sampler

    % PROPERTIES (Public)
    properties(GetAccess = public, SetAccess = protected)

        % The name of the sampler
        name

        % Make BCVARProbs object a data member:
        BCVARProbs
        ParticleFilter

        % Class Data Members:
        initialState        % the initial parameter states
        initialOmega        % initial covariance matrix for the Q proposals
        Mu                  % Adaptive parameter mean
        Omega               % Adapted covariance matrix for the Q proposals
        stateStore          % Record of the accepted parameter states
        startCovMeanEstimation
        proposalCovScale


        NumberParticles     % #Particles in the SISR filter
        ResampleThreshold   % #Particles below which resampling is triggered
        ResamplingSchemeFunc%
        InitialParticles    % Initial Particles
        InitialWeights      % Initial Particle Weights
        SystemParams        % System Model Parameters
        ObservationParams   % Observation Model Parameters
        pfOptions           % The Particle Filter Options
        continueOnError
        NumberTimeSteps
        %TransformedDataSet
        DataSet

        chainLength         % Number of samples to generate
        numAccepts          % Total number of accepted moves
        numRejects          % Total number of rejected moves
        d                   % adaptive dimension d=(n-r)*r
        truncate            % Are we working with the compelete
                            % parameter matrix? or are there r^2 constraints imposed?
    end

    % PROPERTIES (Constant)
    properties(Constant)
        % The default flag for ContinueOnError
        DefaultContinueOnError = 0;
    end

    % METHODS (Public)
    methods(Access = public)

        % Function: PMMH Sampler
        % Input:    ParameterState
        % Output:   None
        % NOTES:
        function this = PMMHSampler(BCVARProbs, initialState, ...
            initialOmega, truncate, startCovMeanEstimation, ...
```

```
            proposalCovScale, NumberParticles, ResampleThreshold, ResamplingSchemeFunction)
            % Particle Marginal Metropolis-Hastings Sampler. The default Constructor.
            this.name = 'PMMH Sampler';
            this.BCVARProbs = BCVARProbs;
            this.initialState = initialState;
            this.initialOmega = initialOmega;
            this.NumberParticles = NumberParticles;
            this.ResampleThreshold = ResampleThreshold;
            this.ResamplingSchemeFunc = ResamplingSchemeFunction;
            this.truncate = truncate;
            this.startCovMeanEstimation = startCovMeanEstimation;
            this.proposalCovScale = proposalCovScale;
            if(this.truncate)
                this.d = (initialState.dimensionality - initialState.rank)...
                    * initialState.rank;
            else
                this.d = (initialState.dimensionality) * initialState.rank;
            end
            this.ParticleFilter = SMCParticleFilter(this.NumberParticles, ...
                this.ResamplingSchemeFunc, this.ResampleThreshold);
        end


        % Function: Initialise Particle Filter
        % Input:    Initial Particles, Initial Weights
        % Output:   None
        % Notes:
        function InitialiseParticleFilter(this, InitialEstimates, SystemParams, ...
                ObservationParams, pfOptions)
            this.InitialParticles   = InitialEstimates.InitialParticles;
            this.InitialWeights     = InitialEstimates.InitialWeights;
            this.SystemParams       = SystemParams;
            this.ObservationParams  = ObservationParams;
            this.pfOptions          = pfOptions;

            if(isfield(pfOptions, 'ContinueOnError'))
                this.continueOnError = this.pfOptions.ContinueOnError;
            else
                this.continueOnError = PMMHSampler.DefaultContinueOnError;
            end
            this.NumberTimeSteps    = this.BCVARProbs.t - 1;
            this.DataSet                        = this.BCVARProbs.ECMModelObject.getDataSet(
1, 0);
            % And remove the first two (empty) rows:
            this.DataSet            = this.DataSet(3:end,:);

        end

        % Function: runChain_BetaSigmaB_Adaptive
        % Input:    chainLength, ECM data object
        % Output:   states (Matrix Array)
        % NOTES:    This function runs the Metropolis-Hastings sampler to
        % create an instance of the Markov Chain for Beta, Sigma and B, and
        % the Latent States. It uses an SMC filter to calculate the
        % Marginal Likelihood
        function [outputStateStore] = runChain_BetaSigmaB_Adaptive(this, chainLength, start
Adapt)
            this.chainLength = chainLength;
            this.stateStore{1} = this.initialState;
            currentState = this.initialState;
            ids = currentState.parameterNames;

            if(this.truncate)
```

```
            if(this.truncate)
                this.Mu(:,1) = zeros(size(currentState.GetBetaTruncatedVec),1);
                this.Mu(:,this.startCovMeanEstimation) = ...
                    zeros(size(currentState.GetBetaTruncatedVec),1);
            else
                this.Mu(:,1) = zeros(size(currentState.GetBetaVec),1);
                this.Mu(:,this.startCovMeanEstimation) = zeros(size(currentState.GetBetaVec
),1);
            end

            this.Omega(:,:,1) = this.initialOmega;
            this.Omega(:,:,this.startCovMeanEstimation) = this.initialOmega;

            % Initial Parameter Beta:
            Beta0           = this.stateStore{1}.GetNamedParameter('Beta');
            Sigma0          = this.stateStore{1}.GetNamedParameter('Sigma');
            B0              = this.stateStore{1}.GetNamedParameter('B');
            logProbBetaOld  = this.BCVARProbs.logPriorProbBeta(Beta0);
            logProbSigmaOld = this.BCVARProbs.logPriorProbSigma(Sigma0);
            logProbBOld     = this.BCVARProbs.logPriorProbB(Sigma0, B0);

            sysModelFunc    = this.SystemParams.SystemEquationFunctionHandle;
            %measModelFunc = this.ObservationParams.MeasurementEquationFunctionHandle;
            %ObservationNoiseMatrix = this.ObservationParams.NoiseMatrix;
            ObservationNoiseMatrix = Sigma0;

            %-------------------------------------------------------------
            % Now generate an ECM parameter struct to store the current
            % values of the variable paramters (beta0)
            %-------------------------------------------------------------
            SystemModelParams.NoiseMatrix = ...
                this.BCVARProbs.ECMParStruct.SystemProcessParameters.SystemProcessParameter
s.NoiseMatrix;
            SystemModelParams.TransitionMatrix = ...
        this.BCVARProbs.ECMParStruct.SystemProcessParameters.SystemProcessParameters.Trans
itionMatrices(:,:,1);
            Alpha = this.BCVARProbs.ECMParStruct.MeasurementProcessParameters.Alpha;
            LagMatrices = this.BCVARProbs.ECMParStruct.MeasurementProcessParameters.LagMatr
ices;
            %ObservationNoiseMatrix = this.BCVARProbs.ECMParStruct.MeasurementProcessParame
ters.NoiseMatrix;

            ecmParameters = ecmParameterGenerator(this.BCVARProbs.ECMModelObject.dimension,
 ...
                this.BCVARProbs.ECMModelObject.numberDataSets, ...
                this.BCVARProbs.ECMModelObject.numberObservations, SystemModelParams, ...
                Alpha, Beta0, LagMatrices, ...
                ObservationNoiseMatrix);


            %-------------------------------------------------------------
            % The PARTICLE FILTER (iteration it = 1)
            %-------------------------------------------------------------
            try
                % Run the this.ParticleFilter on the dataSet for the specified
                % NumberTimeStemps
                [mmse, ess, marginalLogLikelihoodOld] = this.ParticleFilter.run4(this.Numbe
rTimeSteps,...
                    this.DataSet, ...
                    sysModelFunc, ecmParameters, this.InitialParticles, this.InitialWeights
, ...
                    ObservationNoiseMatrix);
```

```
            catch Exception
                fprintf('SISRParticleFilterStudy:Run:ERORR: DataSet Number %i\n', i);
                % Check whether to continue with computation or halt
                if(this.continueOnError == 1)
                    Exception
                    %continue
                else
                    rethrow(Exception)
                end
            end

            Z_t = mmse;
            % Add in some padding (for compatibility with the KF version):
            Z_t = [zeros(2,this.BCVARProbs.ECMModelObject.dimension); Z_t];
            logPYGivenBetaOld = marginalLogLikelihoodOld + logProbBetaOld + ...
                logProbBOld + logProbSigmaOld;
            it = 2; this.numAccepts = 0; this.numRejects = 0;
            while it < this.chainLength+1
                % Propose a new state (adaptively) for Beta:
                proposedState   = this.betaStateProposalAdaptive(this.stateStore{it-1}, it,
 startAdapt);
                Beta_xx         = proposedState.GetNamedParameter('Beta');
                logProbBetaNew  = this.BCVARProbs.logPriorProbBeta(Beta_xx);

                % Now sample (Sigma|Beta,Y) exactly:
                Sigma_xx        = this.BCVARProbs.sample_SigmaGivenBeta_tilde(Beta_xx, Z_t)
;
                logProbSigmaNew = this.BCVARProbs.logPriorProbSigma(Sigma_xx);

                % Now sample (B|Beta, Y) exactly:
                B_xx            = this.BCVARProbs.sample_BGivenBetaY_tilde(Beta_xx, Z_t);
                logProbBNew     = this.BCVARProbs.logPriorProbB(Sigma_xx, B_xx);

                Alpha_xx = B_xx(2:end,:)';
                ObservationNoiseMatrix = diag(diag(Sigma_xx));
                %ObservationNoiseMatrix = this.ObservationParams.NoiseMatrix;

                % Sample the parameters of the latent system:
                F_xx        = this.BCVARProbs.sample_F(Z_t);
                SigmaW_xx   = this.BCVARProbs.sample_SigmaW(Z_t);

                Params_complete = {Beta_xx, Sigma_xx, B_xx', SigmaW_xx, F_xx'};
                proposedState = ParameterState(ids, Params_complete, 'NewParameterState_xx'
);

                %-------------------------------------------------------------
                % Now generate an ECM parameter struct to store the current
                % values of the parameters in the model (Beta_xx, Sigma_xx, etc)
                %-------------------------------------------------------------
                ecmParameters = ecmParameterGenerator(this.BCVARProbs.ECMModelObject.dimens
ion,...
                    this.BCVARProbs.ECMModelObject.numberDataSets, ...
                    this.BCVARProbs.ECMModelObject.numberObservations, SystemModelParams, A
lpha,...
                    Beta_xx, LagMatrices, ...
                    ObservationNoiseMatrix);


                %-------------------------------------------------------------
                % The PARTICLE FILTER (iteration it > 1)
                %-------------------------------------------------------------
                try
```

```
                    % Run the this.ParticleFilter on the dataSet for the specified
                    % NumberTimeStemps with the updated parameters...
                    [mmse, ess, marginalLogLikelihoodNew] = ...
                        this.ParticleFilter.run4(this.NumberTimeSteps, this.DataSet, ...
                        sysModelFunc, ecmParameters, this.InitialParticles, ...
                        this.InitialWeights, ObservationNoiseMatrix);

                catch Exception
                    fprintf('SISRParticleFilterStudy:Run:ERORR: DataSet Number %i\n', i);
                    % Check whether to continue with computation or halt
                    if(this.continueOnError == 1)
                        Exception
                        %continue
                    else
                        rethrow(Exception)
                    end
                end
                Z_t = mmse;
                % Add in some padding:
                Z_t = [zeros(2,this.BCVARProbs.ECMModelObject.dimension); Z_t];
                % Evaluate the marginal probability of Y, and add the prior
                % probabilities:
                logPYGivenBetaNew = marginalLogLikelihoodNew + logProbBetaNew ...
                    + logProbBNew + logProbSigmaNew;

                %-----Calculate the acceptance probability for the state_xx:----
                acceptRatio =  logPYGivenBetaNew - logPYGivenBetaOld;
                if isnan(acceptRatio)
                    alphaProb = 0;
                else
                    alphaProb = exp(acceptRatio);
                end
                acceptProb = min(1, alphaProb);
                u = unifrnd(0,1);
                if u < acceptProb
                    currentState = proposedState;
                    logPYGivenBetaOld = logPYGivenBetaNew;
                    this.stateStore{it} = proposedState;
                    this.numAccepts = this.numAccepts+1;
                else
                    this.stateStore{it} = currentState;
                    this.numRejects = this.numRejects+1;
                end

                if(it > this.startCovMeanEstimation)
                    if(this.truncate)
                        this.Mu(:,it) = this.Mu(:,it-1)+...
                            (currentState.GetBetaTruncatedVec-this.Mu(:,it-1))/(it+1);
                        this.Omega(:,:,it) = this.Omega(:,:,it-1)+...
                            ((currentState.GetBetaTruncatedVec-this.Mu(:,it-1))*...
                            (currentState.GetBetaTruncatedVec-this.Mu(:,it-1))'...
                            -this.Omega(:,:,it-1))/(it-1);
                    else
                        this.Mu(:,it) = this.Mu(:,it)+(currentState.GetBetaVec...
                            -this.Mu(:,it))/(it+1);
                        this.Omega(:,:,it) = this.Omega(:,:,it)+...
                            ((currentState.GetBetaVec-this.Mu(:,it))*...
                            (currentState.GetBetaVec-this.Mu(:,it))'-this.Omega(:,:,it-1))/
(it-1);
                    end
                end
                if(rem(it,20)==0)
```

```
                    it
                    this.Mu(:,it)'
                    this.Omega(:,:,it)
                    if(this.BCVARProbs.dimension==4 && this.BCVARProbs.rank==1)
                        disp('  |--Beta-| |-----------------Sigma---------------||--------B
--------| |-----------------SigmaW--------------|')
                    end
                    disp(currentState.parametersCat)
                end
                it = it + 1;
            end
            outputStateStore = this.stateStore;
        end



        % Function: betaStateProposal
        % Input:    current state
        % Output:   new state proposal
        % NOTES:   This function uses the current value of Omega (Q proposal
        % covariance) to generate a new potential state of Beta:
        function [proposedState] = betaStateProposal(this, currentState)
            Omega = this.initialOmega;
            currentIds = currentState.parameterNames;
            if(this.truncate)
                currentBetaVec = currentState.GetBetaTruncatedVec;
            else
                currentBetaVec = currentState.GetBetaVec;
            end

            % this is where we draw a proposed new state:
            proposedBetaVec = mvnrnd(currentBetaVec, Omega)';
            % now arrange into individual parameter matrices (stored as
            % cells) to construct the new state:
            currentParamsMat = reshape(currentState.GetParametersVec,...
                currentState.dimensionality, []);

            if (this.truncate)
                newBetaMat = [eye(currentState.rank) ...
                    reshape(proposedBetaVec, currentState.rank,[])'];
            else
                newBetaMat = [reshape(proposedBetaVec, currentState.dimensionality,[])'];
            end
            % Use the current state parameters to create a new State object:
            currentParams = mat2cell(currentParamsMat, currentState.dimensionality, ...
                GetParametersColumnDims(currentState))';
            proposedState = ParameterState(currentIds, currentParams, 'newParameterState');
            proposedState.SetNamedParameter('Beta', newBetaMat);
        end

        % Function: betaStateProposalAdaptive
        % Input:    current state
        % Output:   new state proposal
        % NOTES:    This function uses an adaptive scheme to generate the
        % proposed state:
        function [proposedState] = betaStateProposalAdaptive(this, currentState, iter, star
tAdapt)
            currentIds = currentState.parameterNames;
            if(this.truncate)
                currentBetaVec = currentState.GetBetaTruncatedVec;
            else
                currentBetaVec = currentState.GetBetaVec;
            end
```

```
            % this is where we draw a proposed new state:
            u = unifrnd(0,1);
            if(iter > startAdapt)
                w1 = 0.95;
                if u < w1
                    covariance = this.proposalCovScale * ...
                        this.Omega(:,:,iter-1)*(2.38)^2/this.d;
                    proposedBetaVec = mvnrnd(currentBetaVec, covariance)';
                else
                    covariance = this.proposalCovScale * ...
                        eye(this.d) * (0.1)^2/this.d;
                    proposedBetaVec = mvnrnd(currentBetaVec, covariance)';
                end
            else
                covariance = this.initialOmega;
                proposedBetaVec = mvnrnd(currentBetaVec, covariance)';
            end

            currentParamsMat = reshape(currentState.GetParametersVec,...
                currentState.dimensionality, []);

            if (this.truncate)
                newBetaMat = [eye(currentState.rank) ...
                    reshape(proposedBetaVec, currentState.rank,[])]';
            else
                newBetaMat = [reshape(proposedBetaVec, ...
                    currentState.dimensionality,[])]';
            end
            % Use the current state parameters to create a new State object:
            currentParams = mat2cell(currentParamsMat, currentState.dimensionality,  ...
                GetParametersColumnDims(currentState))';
            proposedState = ParameterState(currentIds, currentParams, 'newParameterState');
            proposedState.SetNamedParameter('Beta', newBetaMat);
        end

        % Function: GetParameterSamples
        % Input:    ParameterName (string)
        % Output:   Individual Parameter Markov Chains
        % NOTES:    This function returns the MCs produced by the sampler
        % in a matrix format: dim=(#params * chainLength)
        % Row1: Param1->
        % Row2: Param2->
        % ... etc

        function [parameterSamples] = GetParameterSamples(this, ParameterName)
            paramIndex = this.stateStore{1}.GetNamedParameterIndex(ParameterName);
            numSubParams = size(reshape(this.stateStore{1}.GetIndexedParameter(paramIndex),
[],1),1);
            parameterSamples = zeros(this.chainLength, numSubParams);
            for i = 1:this.chainLength
                parameterSamples(i,:) = reshape(this.stateStore{i}.GetIndexedParameter(para
mIndex)',[], 1);
            end


        end

    end
end
```

Remark *In some senses this work continues where the work of GWP's former student Shamin Kinathil left off, and as such utilises some underlying code components that he developed: for example methods to generate from the model equations, a basic Kalman filter implementation etc. see [Kinathil, 2011]*

Bibliography

[Alexander, 1999] Alexander, C. (1999). Optimal hedging using cointegration. *Philisophical Transactions of the Royal Society, Series A*, 357:pp. 2039–2058.

[Alexander and Dimitriu, 2002] Alexander, C. and Dimitriu, A. (2002). The cointegration alpha: Enchanced index tracking and long-short equity market neutral stragies. ICMA Centre Discussion Papers in Finance icma-dp2002-08, Henley Business School, Reading University.

[Anderson and Moore, 1979] Anderson, B. D. O. and Moore, J. B. (1979). *Optimal filtering*. Prentice-Hall, Englewood Cliffs, N.J.

[Andrieu and Atachade, 2005] Andrieu, C. and Atachade, Y. (2005). On adaptive markov chain monte carlo algorithms. *Bernoulli*, 11(5):pp. 815–828.

[Andrieu and Atachade, 2007] Andrieu, C. and Atachade, Y. (2007). On the efficiency of adaptive MCMC algorithms. *Communications in Probability*, 12:pp. 336–349.

[Andrieu et al., 2010] Andrieu, C., Doucet, A., and Holenstein, R. (2010). Particle markov chain monte carlo methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 72(3):269–342.

[Andrieu and Moulines, 2006] Andrieu, C. and Moulines, E. (2006). On the ergodicity properties of sone adaptive MCMC algorithms. *Annals of Applied Probability*, 16.

[Baum, 1972] Baum, L. (1972). An inequality and associated maximization technique in statistical estimation of probabilistic functions of markov processes. *Inequalities*, 3:pp. 1–8.

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer.

[Bru, 1991] Bru, M.-F. (1991). Wishart processes. *Journal of Theroetical Probability*, 4:pp. 725–751.

[Carpenter et al., 1999] Carpenter, J., Clifford, P., and Fearnhead, P. (1999). An improved particle filter for non-linear problems. *IEE proceedings - Radar, Sonar and Navigation*, 146:pp. 2–7.

[Chopin, 2004] Chopin, N. (2004). Central limit theorem for sequential monte carlo methods and its application to bayesian inference. *The Annals of Statistics*, 32(6):pp. 2385–2411.

[Chopin, 2010] Chopin, N. (2010). SMC^2: an efficient algorithm for sequential analysis of state-space models. *Journal of the Royal Statistical Society: Series B*, 75(3):pp. 397–426.

[Crisan and Doucet, 2002] Crisan, D. and Doucet, A. (2002). A survey of convergence results on particle filtering methods for practitioners. *IEEE Journal of Signal Processing*, 50(3):pp. 736–746.

[Cuciero et al., 2011] Cuciero, C., Filipovic, D., Mayerhofer, E., and Teichmann, J. (2011). Affine processes on positive semidefinite matrices. *The Annals of Applied Probability*, 21(2):pp. 397–463.

[Douc and Cappe, 2005] Douc, R. and Cappe, O. (2005). Comparison of resampling schemes for particle filtering. In *Image and Signal Processing and Analysis, 2005. ISPA 2005. Proceedings of the 4th International Symposium on*, pages 64 – 69.

[Doucet and Johansen, 2011] Doucet, A. and Johansen, A. M. (2011). *A Tutorial on Particle filtering and smoothing: Fiteen years later. In The Oxford Handbook of Nonlinear Filtering.* Oxford University Press.

[Doucet et al., 2009] Doucet, A., Kantas, N., Singh, S., and Maciejowski, J. (2009). An overview of sequential monte carlo methods for parameter estimation in general state-space models. *Proceedings IFAC System Identification (SySid) Meeting.*

[Engle and Granger, 1987] Engle, R. F. and Granger, C. W. J. (1987). Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):pp. 251–276.

[Engle and Yoo, 1987] Engle, R. F. and Yoo, B. S. (1987). Forecasting and testing in co-integrated systems. *Journal of Econometrics*, 35(1):143 – 159.

[Fearnhead et al., 2010] Fearnhead, P., Papaspilioupoulos, O., Roberts, G., and Stuart, A. (2010). Random weight particle filtering of continous time processes. *Journal of the Royal Statistical Society: Series B*, 72:pp. 497–513.

[Fox and West, 2011] Fox, E. B. and West, M. (2011). Autoregressive models for variance matrices: Stationary inverse wishart processes. *The Annals of Applied Probability*, (arXiv:1107.5239).

[Geman and Geman, 1984] Geman, S. and Geman, D. (1984). Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):pp. 721–741.

[Geweke, 1996] Geweke, J. (1996). Bayesian reduced rank regression in econometrics. *Journal of Econometrics*, 75:pp. 121–146.

[Gupta and Nagar, 1999] Gupta, A. and Nagar, D. (1999). *Matrix variate distributions.* Chapman Hall CRC.

[Haario et al., 2001] Haario, H., Saksman, E., and Tamminen, J. (2001). An adaptive metropolis algorithm. *Bernoulli*, 7:pp. 223–242.

[Hastings, 1970] Hastings, W. (1970). Monte carlo sampling methods using markov chains and their applications. *Biometrika*, 57:pp. 97–109.

[Hendry and Richard, 1983] Hendry, D. F. and Richard, J.-F. (1983). The econometric analysis of economic time series. *International Statistical Review / Revue Internationale de Statistique*, 51(2):pp. 111–148.

[Jensen et al., 1995] Jensen, C., Kong, A., and Kjaerulff, U. (1995). Blocking Gibbs sampling in very large probabilistic expert systems. *International Journal of Human Computer Studies*, 42:pp. 647–666.

[Johansen and Evans, 2007] Johansen, A. M. and Evans, L. (2007). Monte carlo methods: Lecture notes. University of Bristol.

[Johansen, 1991] Johansen, S. (1991). Estimation and hypothesis testing of cointegration vectors in gaussian vector autoregressive models. *Econometrica: Journal of the Econometric Society*, 59(6):1551–1580.

[Johansen, 1995] Johansen, S. (1995). *Likelihood-Based Inference in Cointegrated Vector Autoregressive Models.* Number 9780198774501 in OUP Catalogue. Oxford University Press.

[Juselius, 2006] Juselius, K. (2006). *The Cointegrated VAR Model: Methodology and Applications*. Number 9780199285679 in OUP Catalogue. Oxford University Press.

[Kalman, 1960] Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45.

[Kinathil, 2011] Kinathil, S. (2011). Nonlinear filtering for non-stationary multivariate cointegration models. MSc thesis, University of New South Wales.

[Koop and Korobilis, 2010] Koop, G. and Korobilis, D. (2010). Bayesian multivariate time series methods for empirical macroeconomics. Working Paper Series 47-09, Rimini Centre for Economic Analysis.

[Koop et al., 2006] Koop, G., Strachan, R., van Dijk, H., and Villani, M. (2006). *Bayesian Approaches to Cointegration*, pages pp. 871–898. Palgrave Macmillan.

[Künsch, 2005] Künsch, H. R. (2005). Recursive monte carlo filters: Algorithms and theoretical analysis. *The Annals of Statistics*, 33(5):pp. 1983–2021.

[Lovell, 2005] Lovell, M. C. (2005). A simple proof of the fwl (frisch-waugh-lovell) theorem. Wesleyan Economics Working Papers 2005-012, Wesleyan University, Department of Economics.

[Lütkepohl, 2007] Lütkepohl, H. (2007). *New Introduction to Multiple Time Series Analysis*. Springer-Verlag Berlin Heidelberg.

[MacKinnon et al., 1999] MacKinnon, J. G., Haug, A. A., and Michelis, L. (1999). Numerical distribution functions of likelihood ratio tests for cointegration. *Journal of Applied Econometrics*, 14(5):563–77.

[Makarov and Glew, 2009] Makarov, R. and Glew, D. (2009). Exact simulation of bessel diffusions. *Monte Carlo Methods and Applications*, 16(3):pp. 283–306.

[Metropolis et al., 1953] Metropolis, N., Rosenthal, A., Rosenbluth, M., A.H.Teller, and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):pp. 1087–1092.

[Metropolis and Ulam, 1949] Metropolis, N. and Ulam, S. (1949). The monte carlo method. *Journal of the American Statistical Association*, 44(247):pp. 335–341.

[Papaspiliopoulos et al., 2007] Papaspiliopoulos, O., Roberts, G., and Sköld, M. (2007). A general framework for the parameterization of hierarchical models. *Statistical Science*, 22(1):pp. 59–73.

[Peters et al., 2010] Peters, G. W., Kannan, B., Lasscock, B., and Mellen, C. (2010). Model Selection and Adaptive Markov Chain Monte Carlo for Bayesian Cointegrated VAR Model. (arXiv:1004.3830v1).

[Phillips, 1954] Phillips, A. W. (1954). Stabilisation policy in a closed economy. *The Economic Journal*, 64(254):pp. 290–323.

[Phillips, 1991] Phillips, P. C. B. (1991). Optimal inference in cointegrated systems. *Econometrica*, 59(2):pp. 283–306.

[Roberts and Rosenthal, 2009] Roberts, G. and Rosenthal, J. (2009). Examples of adaptive MCMC. *Journal of Computational and Graphical Statistics*, 18.

[Sargan, 1964] Sargan, J. (1964). Wages and prices in the united kingdom: A study of econometric methodology. In Hart, P., Mills, G., and Whitaker, J., editors, *Econometric Analysis for National Economic Planning*, pages 25–63. Butterworth Co., London, UK.

[Stock and Watson, 1988] Stock, J. H. and Watson, M. W. (1988). Testing for common trends. *Journal of the American Statistical Association*, 83(404):pp. 1097–1107.

[Sugita, 2002] Sugita, K. (2002). Testing for cointegration rank using bayes factors. Royal Economic Society Annual Conference 2002 171, Royal Economic Society.

[von Neuman, 1951] von Neuman, J. (1951). Various techniques used in connection with random digits. *National Bureau of Standards, Applied Math Series*, 11:pp. 36–38.