

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

**A NOVEL MULTIVARIATE STOCHASTIC VOLATILITY MODEL AND
ESTIMATION WITH GPU COMPUTING**



Ph.D. THESIS

Halil Ertürk ESEN

Department of Computational Science and Engineering

Computational Science and Engineering Programme

JUNE 2016

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

**A NOVEL MULTIVARIATE STOCHASTIC VOLATILITY MODEL AND
ESTIMATION WITH GPU COMPUTING**



Ph.D. THESIS

**Halil Ertürk ESEN
(702062002)**

Department of Computational Science and Engineering

Computational Science and Engineering Programme

**Thesis Advisor: Prof. Dr. Burç ÜLENGİN
Thesis Co-advisor: Prof. Dr. M. Serdar ÇELEBİ**

JUNE 2016

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

**YENİ BİR ÇOK DEĞİŞKENLİ STOKASTİK OYNAKLIK MODELİ VE GPU
TABANLI HESAPLAMA İLE KESTİRİMİ**

DOKTORA TEZİ

**Halil Ertürk ESEN
(702062002)**

Hesaplamalı Bilim ve Mühendislik Anabilim Dalı

Hesaplamalı Bilim ve Mühendislik Programı

**Tez Danışmanı: Prof. Dr. Burç ÜLENGİN
Tez Eş Danışmanı: Prof. Dr. M. Serdar ÇELEBİ**

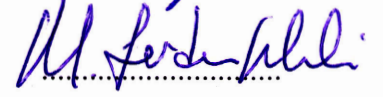
HAZİRAN 2016

Halil Ertürk ESEN, a Ph.D. student of ITU Informatics Institute student ID 702062002, successfully defended the thesis/dissertation entitled “A NOVEL MULTIVARIATE STOCHASTIC VOLATILITY MODEL AND ESTIMATION WITH GPU COMPUTING”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

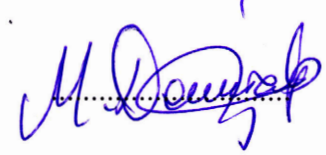
Thesis Advisor : **Prof. Dr. Burç ÜLENGİN**
Istanbul Technical University



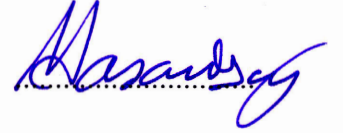
Co-advisor : **Prof.Dr. M. Serdar ÇELEBİ**
Istanbul Technical University



Jury Members : **Prof.Dr. Metin DEMİRALP**
Istanbul Technical University



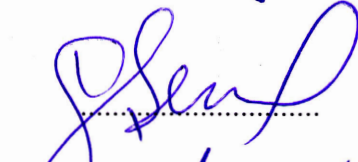
Prof.Dr. Hasan DAĞ
Kadir Has University



Prof.Dr. Oktay TAŞ
Istanbul Technical University



Prof.Dr. Kerem ŞENEL
Istanbul Commerce University



Assoc.Prof.Dr. Cumhuri E. EKİNCİ
Istanbul Technical University



Date of Submission : 18 January 2016

Date of Defense : 06 June 2016



FOREWORD

First and foremost I would like to thank my advisor Prof. Dr. Burç ÜLENGİN whose precious guidance and mentorship guided me through not only this study but also my whole doctoral journey and helped me for surviving and reaching to this point. I am also grateful to my co-advisor Prof. Dr. M. Serdar ÇELEBİ who provided great support and motivation in this study and inspired me with his enthusiasm for the field of computational sciences from the beginning. As I feel very lucky for being their students, I would like to let them know that their kindness and understanding are greatly appreciated.

I would also like to thank my committee member and professor, Prof. Dr. Metin DEMİRALP whom I learned really too much, from both his solid courses and his wisdom.

I would also like to thank my committee members Prof. Dr. Hasan DAĞ and Prof. Dr. Oktay TAŞ who provided valuable opinions, comments and advice which significantly contributed in developing and shaping the study.

For her unique patience and great help, I would like to thank my loving wife Ayla who always stands by me and special thanks to my son Arda whose unbelievable pace of growth motivated and reminded me that I must go on and speed up when things get slower.

As it has always been, I have constantly felt the blessings of my mother and hearty support of my father and sisters which kept me stronger in this journey too.

June 2016

Halil Ertürk ESEN



TABLE OF CONTENTS

	<u>Page</u>
FOREWORD	vii
TABLE OF CONTENTS	ix
ABBREVIATIONS	xi
LIST OF TABLES	xiii
LIST OF FIGURES	xv
SUMMARY	xvii
ÖZET	xix
1. INTRODUCTION	1
2. THEORETICAL AND MATHEMATICAL BACKGROUND	7
2.1 Stochastic Volatility Modeling	7
2.1.1 Overview of stochastic volatility models	7
2.1.2 A novel multivariate stochastic volatility model (MSV-D)	12
2.2 Estimation Algorithms for Stochastic Volatility Models	16
2.2.1 Stochastic volatility models as nonlinear state space models	16
2.2.1.1 Densities implied by the MSV-D model	17
2.2.1.2 Filtering, prediction and smoothing	19
2.2.1.3 Mean, variance and likelihood	22
2.2.2 Overview of the estimation methods	22
2.2.3 Estimation with Markov Chain Monte Carlo (MCMC) methods ...	27
2.2.3.1 Preliminaries on the MCMC methods	27
2.2.3.2 MCMC based estimation algorithms for the MSV-D	29
MCMC with EM algorithm for the MSV-D	29
Bayesian MCMC for the MSV-D	33
2.2.4 Estimation with sparse grid integration (SGI) method	43
2.2.4.1 Preliminaries on the SGI method	43
2.2.4.2 SGI based estimation algorithms for the MSV-D	45
3. COMPUTATIONAL IMPLEMENTATION	51
3.1 Computational Aspects of Estimation Algorithms	51
3.2 Parallelization Approaches for the MCMC Based Algorithms	57
3.3 Parallelization Approach for the SGI Based Algorithms	58
3.4 Notes on the GPU Implementation	60
4. METHODOLOGY	63
4.1 Software Programs and Hardware	63
4.2 Assessment of the MSV-D Model	64
4.3 Assessment of the Estimation Algorithms	68
5. RESULTS	71
5.1 MSV-D Model on Simulated Returns Data	71
5.2 MSV-D Model on Empirical Data	84
5.3 Comparative Simulations for Estimation Algorithms	89
5.4 SGI Based Estimation Algorithms on Empirical Data	94

5. CONCLUSION	99
REFERENCES	103
APPENDICES	107
CURRICULUM VITAE	125



ABBREVIATIONS

AR	: Autoregressive
ARCH	: Autoregressive Conditional Heteroskedasticity
EKF	: Extended Kalman Filter
EM	: Expectation Maximization
GARCH	: Generalized Autoregressive Conditional Heteroskedasticity
GPU	: Graphics Processing Unit
MCMC	: Markov Chain Monte Carlo
MSV	: Multivariate Stochastic Volatility
PCA	: Principle Component Analysis
RMSE	: Root Mean Squared Error
SGI	: Sparse Grid Integration
SV	: Stochastic Volatility
VaR	: Value at Risk



LIST OF TABLES

	<u>Page</u>
Table 2.1 : Multidimensional grid sizes based on the trapezoid rule.....	46
Table 4.1 : Implemented estimation algorithms.....	64
Table 5.1 : Static MSV-D model parameter estimation results of $\gamma_i, \varphi_i, \sigma_{\eta,i}$	73
Table 5.2 : Static MSV-D model parameter estimation results of $\rho_{\varepsilon\varepsilon,ij}, \rho_{\varepsilon\eta,ii}, \rho_{\varepsilon\eta,ij}$	74
Table 5.3 : Static MSV-D model log-volatility estimation results.....	76
Table 5.4 : Dynamic MSV-D model parameter estimation results of $\gamma_i, \varphi_i, \sigma_{\eta,i}$	79
Table 5.5 : Dynamic MSV-D model parameter estimation results of δ_i	79
Table 5.6 : Dynamic MSV-D model parameter estimation results of θ_i	80
Table 5.7 : Dynamic MSV-D model parameter estimation results of $\sigma_{\omega,i}$..	80
Table 5.8 : Dynamic MSV-D model log-volatility estimation results	81
Table 5.9 : Dynamic MSV-D model estimation results of the dynamic correlations, $\rho_{\varepsilon\varepsilon,ij,t}, \rho_{\varepsilon\eta,ii,t}, \rho_{\varepsilon\eta,ij,t}$	81
Table 5.10 : MSV-D model parameter estimation results of $\gamma_i, \varphi_i, \sigma_{\eta,i}$ on S&P500, IBM and Intel (INTC) returns.....	85
Table 5.11 : MSV-D model parameter estimation results of δ_i on S&P500, IBM and Intel (INTC) returns.....	85
Table 5.12 : MSV-D model parameter estimation results of θ_i on S&P500, IBM and Intel (INTC) returns.....	85
Table 5.13 : MSV-D model parameter estimation results of $\sigma_{\omega,i}$ on S&P500, IBM and Intel (INTC) returns.....	86
Table 5.14 : GARCH model parameter estimates on S&P500, IBM and Intel (INTC) returns.....	86
Table 5.15 : Log-volatility estimates MSV-D vs. GARCH on S&P500, IBM and Intel (INTC) returns.....	87
Table 5.16 : Accuracy comparison of SGI and MCMC based estimation algorithms.....	90
Table 5.17 : Parameter estimation accuracy comparison of the SGI and MCMC based algorithms.....	91
Table 5.18 : Execution times of SGI and MCMC based estimation algorithms.....	92
Table 5.19 : MSV-B parameter estimation results on EUR/TL and USD/TL returns.....	95
Table 5.20 : CCC-GARCH parameter estimation results on EUR/TL and USD/TL returns.....	96
Table 5.21 : Log-volatility estimation comparisons on EUR/TL and USD/TL returns.....	96



LIST OF FIGURES

	<u>Page</u>
Figure 3.1 : Estimation algorithm dependencies.....	52
Figure 3.2 : Sequential (i.e. on-line) vs. batch algorithm.....	52
Figure 3.3 : Smoothing algorithm of MCMC with EM approach.....	53
Figure 3.4 : Smoothing algorithm of Bayesian MCMC approach.....	54
Figure 3.5 : Filtering algorithm of SGI approach.....	55
Figure 3.6 : Parallel MCMC based smoothing algorithm.....	57
Figure 3.7 : Parallel SGI based filtering algorithm.....	59
Figure 5.1 : Simulated series based on the static MSV-D model.....	72
Figure 5.2 : Log-volatility fits for the static MSV-D model.....	75
Figure 5.3 : Simulated series based on the dynamic MSV-D model.....	77
Figure 5.4 : Examples of simulated dynamic correlations based on the dynamic MSV-D model.....	78
Figure 5.5 : Log-volatility fits for the dynamic MSV-D model.....	82
Figure 5.6 : Dynamic correlation fits for the dynamic MSV-D model.....	83
Figure 5.7 : Return series of S&P500, IBM and Intel (INTC).....	84
Figure 5.8 : Log-volatility estimates of S&P500, IBM and Intel (INTC)...	87
Figure 5.9 : Correlations between returns of S&P500, IBM and Intel (INTC).....	88
Figure 5.10 : Examples of dynamic leverage, cross-leverage and volatility spillover estimates.....	89
Figure 5.11 : Accuracy comparison of SGI and MCMC based estimation algorithms.....	90
Figure 5.12 : Execution times of serial and GPU accelerated estimation algorithms.....	93
Figure 5.13 : Speed up by dimension in SGI and MCMC based algorithms	94
Figure 5.14 : Return series of EUR/TL and USD/TL.....	95
Figure 5.15 : Log-volatility smoothing estimates on EUR/TRL and USD/TRL returns.....	97
Figure B.1 : Time varying correlation matrix construction mechanism of the MSV-D model.....	109



A NOVEL MULTIVARIATE STOCHASTIC VOLATILITY MODEL AND ESTIMATION WITH GPU COMPUTING

SUMMARY

Modeling and estimation of volatilities of asset returns in financial markets have been a major research area for the last three decades because of the prominent role of volatility concept in mathematical and quantitative finance. Reliable volatility estimates of asset returns are indispensable inputs to several mathematical models in financial frameworks including but not limited to risk management and measurement, option pricing, portfolio and asset management.

Volatilities of asset returns show several well studied and reported structural patterns which are called stylized facts including time varying and persistent dynamics, leverage effects and spillovers. Models and estimation methods for addressing those stylized facts about volatility for asset returns are central to the contemporary volatility estimation research.

Stochastic volatility (SV) models constitute a family of models considering the conditional variance of returns as latent variables driven by a stochastic process instead of explicitly modeling it as in the Autoregressive Conditional Heteroskedasticity (ARCH) models which constitute an other family of models in the volatility modeling research field. By construction, SV models are quite flexible and versatile in capturing the stylized facts, however because of their nonlinear structures, linear approximations or computationally demanding numerical methods are required for the associated estimation problems.

An appreciable amount of research composed of several multivariate model specifications and parameterization addressing different and more complicated stylized facts not only about volatility but also about co-volatility and their multidimensional dynamics is available. In the multivariate stochastic volatility (MSV) modeling research the control mechanisms and parameterizations of the covariance and/or correlation matrices in MSV models and their handling in time-varying settings are the core topics since almost all stylized facts are imposed through the structure of those matrices which have special structures and restrictions on their entries in MSV models. Addressing several stylized facts at the same time in a single model is not a trivial task and requires appropriate mechanisms and most of the available models in the literature address only a subset of stylized facts at the same time. In this context, a novel MSV model referred as MSV-D is proposed as one of the objectives of this thesis. The proposed MSV-D model can accommodate most of the common stylized facts, namely correlations between asset returns, leverage effect (i.e. asymmetry) cross-leverage effect and volatility spillovers and furthermore it allows replacing the static versions of the listed stylized facts with the time-varying (dynamic) counterparts completely or partially. The proposed MSV-D model achieves this flexibility and generality by modeling the correlations as

separate stochastic processes like the volatilities. The proposed MSV-D model includes a specially designed mechanism for handling the time-varying correlation matrices and controlling the stochastic processes driving correlations. Having been proposed a MSV model, its estimation algorithm based on the Markov Chain Monte Carlo (MCMC) methods in a Bayesian setting is also developed. The proposed MSV-D model and its Bayesian MCMC estimation method are illustrated on simulated and empirical data and it is shown that the proposed MSV-D model and its Bayesian estimation algorithm perform well in both static and dynamic settings.

As being nonlinear state space models, MSV models require estimation methods that can handle high dimensional integrals for obtaining smoothing, filtering and prediction estimates of log-volatilities and parameter estimates. Mainstream estimation method for the MSV models are based on the MCMC methods including the Gibbs sampling and Metropolis-Hastings algorithms. MCMC methods are not affected by the high dimensionality in contrast with the any other alternative methods available including other Monte Carlo based probabilistic methods such as resampling, importance sampling and rejection sampling and exact methods such as the numerical integration. Moreover, MCMC methods can be extended naturally in a Bayesian setting where parameter estimation can also be performed by the sampling schemes offered by MCMC without the need for explicit calculation and separate routines for maximizing the log likelihood. The drawbacks with the MCMC method are the issues in convergence and error control and selection of the proposal density where the posterior density is not analytically tractable. Poor mixing chains with high inefficiency factors are common in applications. In search of an alternative estimation approach for the MSV models which would have better error control and convergence properties and computational features competing with the MCMC approach, Sparse Grid Integration (SGI) based estimation algorithms which have not been studied for MSV models previously, are developed and evaluated for the second objective of the study. SGI method is a smartly reshaped version of the conventional numerical integration method for handling multi-dimensional integrals by constructing multi-dimensional integration formulas in a way that the dimensionality effect is decreased to a certain extent which allows practical implementation in higher dimensional cases in contrast to the conventional numeric integration methods.

The proposed SGI based estimation algorithms are illustrated on simulated and empirical data and it is shown that the proposed algorithms perform as well as the MCMC based algorithms and in certain conditions surpass the MCMC methods in terms of both accuracy and computational performance. Although the issues with dimensionality is significantly reduced with the SGI based approach, high dimensional problems can still be problematic from the computational perspective.

The computational requirements of the both MCMC and SGI based algorithms are quite high. In this context, computational improvements that can be achieved with the usage of graphics processing unit (GPU) for estimation algorithms are evaluated by developing and implementing parallelization approaches for MCMC and SGI based estimation algorithms as the third objective of the study.

In the simulation study conducted implemented parallel GPU estimation algorithms provided significant improvements in execution times with speed up values up to 16 for MCMC based algorithms and speed up values up to 25 for SGI based algorithms on single GPU which are promising results for larger scale parallel architecture implementations.

YENİ BİR ÇOK DEĞİŞKENLİ STOKASTİK OYNAKLIK MODELİ VE GPU TABANLI HESAPLAMA İLE KESTİRİMİ

ÖZET

Finansal oynaklık (ing: volatility) kavramının matematiksel ve sayısal finans alanında oldukça önemli bir yeri olması sebebiyle finansal piyasalarda oynaklık modellemesi ve kestirimi (ing: estimation), temel bir araştırma alanı olarak karşımıza çıkmaktadır. Güvenilir oynaklık tahminleri, risk yönetimi, opsiyon fiyatlama, portföy ve varlık yönetimi gibi birçok matematiksel model ve sayısal finans yaklaşımı için vazgeçilmez derecede önemli girdilerdir.

Finansal varlık getirilerindeki oynaklık üzerinde yapılan bir çok araştırma ve çalışma oynaklığın çeşitli yapısal desenler ve dinamikler gösterdiğini ortaya koymuştur. Zaman içinde değişkenlik, yer yer kalıcı özellikte dinamikler, getiriler ile oynaklık arasındaki ilişkiyi ifade eden kaldıraç (asimetri) ve birden fazla getiri sözkonusu olduğunda oynaklıklar arasındaki yayılma etkileri bu yapısal desenlerden önemli olanlardır. Oynaklıkla ilgili bu yapısal desenleri yansıtabilecek matematiksel modeller kurgulamak, bunlara ilişkin kestirim (ing: estimation) yöntemleri ve araçlar geliştirmek ile bunlar ile getiri verileri üzerinde gerçekleştirilen analizler, güncel oynaklık kestirim araştırmalarının ve çalışmalarının temel odağı durumundadır.

Stokastik oynaklık modelleri iki temel oynaklık modelleme yaklaşımından bir tanesidir. Stokastik oynaklık modelleri getirilerin koşullu varyansını, diğer bir önemli model ailesi olan Autoregressive Conditional Heteroskedasticity (ARCH) modellerinde olduğu gibi açık biçimde modellemek yerine koşullu varyansı, stokastik bir süreci takip eden örtülü (ing: latent) bir değişken olarak ele alan bir model ailesidir. Stokastik oynaklık modelleri koşullu varyansların ayrı bir stokastik süreç olarak ele alınışı dolayısıyla, oynaklıkla ilgili belirtilen yapısal desenleri ve dinamikleri yakalama konusunda oldukça esnek ve yetenekli modellerdir. Ancak doğrusal olmayan yapıları sebebiyle stokastik oynaklık modelleri, ilgili kestirim problemleri için ya doğrusal yaklaşıma dayalı yöntemler ya da yoğun hesaplama ihtiyacı duyan sayısal yöntemler gerektirmektedirler.

Yazında, oynaklık ve oynaklığın çok boyutlu dinamikleri ile ilgili yapısal desenleri gözetken ve modellemeye çalışan çok değişkenli modeller üzerine ciddi miktarda çalışma ve araştırma bulunmaktadır. Çok değişkenli stokastik oynaklık (ÇDSO) modelleme araştırmalarında, kovaryans ve korelasyon matrisleri hemen hemen tüm yapısal desenlerin ve dinamiklerin belirlenmesinde anahtar bir araç olduğundan, özel biçimlere sahip bu matrislerin zaman içinde değişken biçimde hareket etmelerine olanak tanıyan mekanizma ve kontrol yöntemleri en önemli araştırma başlıklarından bir tanesidir. Aynı anda birden fazla yapısal deseni tek bir çok değişkenli model içinde kurgulamak, uygun mekanizma ve kontrol yöntemi gerektirmesi bakımından çok kolay değildir ve bu zorluk sebebiyle yazında bulunan modellerin bir çoğunun bahsedilen yapısal desenlerin sadece küçük bir bölümünü aynı anda yansıtabildikleri

görülmektedir. Bu bağlamda, bu çalışmanın ilk amacı doğrultusunda yeni bir ÇDSO modeli geliştirilmiş ve önerilmiştir. Önerilen model, çalışma içerisinde MSV-D olarak anılmaktadır. Önerilen MSV-D modeli, yapısı itibariyle varlık getirileri arasındaki korelasyon, varlık getirileri ve bunların oynaklıkları arasındaki ilişki olarak ifade edilen kaldıraç etkisi ve çapraz kaldıraç etkileri ile oynaklıklar arasındaki geçişkenliği ifade eden oynaklık yayılımı özelliklerini aynı anda barındırabilmekte ve dahası bu özelliklerin zaman içinde değişebilen (dinamik) karşılıklarının kısmi ya da bütün olarak modele dahil edilebilmesine olanak tanımaktadır. Önerilen model, yazındaki mevcut dinamik modellerden farklı olarak dinamik kaldıraç ve çapraz kaldıraç etkileri ile dinamik oynaklık yayılımı etkilerinin modellenmesini sağlamaktadır. Önerilen model özelleştirilebilir genel bir yapıya sahiptir ve araştırmacıya farklı niteliklerde modeller deneme esnekliği vermektedir. Yazında bulunan temel ÇDSO modellerinin bir çoğu, önerilen MSV-D modelinin özel bir hali olarak parametreleştirilebilmektedir.

Önerilen modelin genelliği ve esnekliği temel olarak korelasyonları, koşullu varyanslara benzer şekilde, ayrı stokastik süreçler olarak ele alması fikrine dayanarak elde edilmektedir. Önerilen model, korelasyon matrislerinin stokastik süreçler vasıtasıyla zaman içerisinde değişimini sağlayan ve bu değişimler esnasında korelasyon matrislerinin artı tanımlılığının ve diğer biçim özelliklerinin korunabilmesini sağlayan özel olarak tasarlanmış bir mekanizma ve çift yönlü matematiksel dönüşümler içermektedir.

Önerilen MSV-D modelinin pratik olarak uygulanabilmesi için, modelin yapısına özel olarak Bayesian bir yaklaşım çerçevesinde kurgulanan Markov Chain Monte Carlo (MCMC) yöntemine dayanan bir kestirim algoritması da çalışmanın bir parçası olarak geliştirilmiştir. Kestirim algoritması önerilen MSV-D modelinin parametrelerinin ve zaman içinde değişebilen örtülü oynaklık ve korelasyon değişkenlerinin MCMC yöntemine dayalı örnekleme ile nasıl elde edileceğini göstermektedir ve MSV-D modelinin yapısına özgüdür.

Önerilen MSV-D modeli ve bunun için geliştirilen Bayesian MCMC kestirim yöntemi statik ve dinamik özellikler doğrultusunda simüle edilmiş getiri ve oynaklık verileri ile gerçek hisse senedi ve endeks getiri serileri üzerinde uygulanmış ve karşılaştırmalı olarak değerlendirilmiştir. Gerçekleştirilen uygulamalar, hem önerilen MSV-D modelinin hem de geliştirilen çözüm yönteminin, statik ve dinamik kurgularda yapısal desenleri yakalama konusunda iyi bir performans sergilediğini göstermiştir.

ÇDSO modelleri, temelde doğrusal olmayan durum uzayı (ing: state space) problemleri olmaları dolayısıyla, düzleme (ing: smoothing), filtreleme ve tahmin (ing: prediction) kestirimleri ile parametre kestirimlerinin elde edilmesi problemlerinde ortaya çıkan çok boyutlu tümlevler (ing: integral) ile başa çıkabilecek sayısal yöntemlere ihtiyaç duymaktadırlar. Son dönemlerde hesaplama teknolojisindeki ilerlemeler bu yöntemlerin uygulanabilirliğini arttırmıştır. ÇDSO modellerinin kestiriminde kullanılan algoritmalar, içerisinde Gibbs örnekleme ve Metropolis-Hastings algoritması da bulunan MCMC yöntemlerine dayanmaktadır. MCMC yöntemleri temel olarak, limit dağılımı hedefteki sonsal (ing: posterior) dağılım olan bir Markov zinciri üzerinden bağımlı örneklemler üreterek sonsal dağılımı elde etme esasına dayanır ve diğer rassal örneklemlere dayanan Monte Carlo tabanlı yöntemlerden çok farklıdırlar. MCMC yöntemlerinin karmaşık örnekleme mekanizmaları diğer Monte Carlo yöntemlerine göre genellikle daha çok sayıda örnekleme alınmasını gerektirir. MCMC yöntemleri kurguları itibariyle diğer

birçok Monte Carlo tabanlı yöntem de dahil olmak tüm alternatif yöntemlerden farklı olarak yüksek boyutluluğun getirdiği zorluklardan etkilenmezler. Ayrıca, MCMC yöntemleri parametre kestirimlerinin, en çok olabilirlik (ing: likelihood) fonksiyonunun maksimizasyonuna ve dolayısıyla bu fonksiyonun değerinin açık biçimde hesaplanmasına gerek duyulmadan örnekleme yolu ile gerçekleştirilmesine olanak tanıyan Bayesian bir yaklaşıma doğal bir biçimde uyarlanabilmektedirler. Bu özelliklerinden dolayı ÇDSO modelleri için MCMC tabanlı kestirim yöntemleri en çok tercih edilen yöntemler haline gelmişlerdir. MCMC yöntemleri birçok iyi özelliğine ve başarılı olmalarına karşın kusursuz yöntemler değildir. MCMC yöntemlerinin hata kontrolü ve yakınsama ile ilgili kendine özgü bir takım kusurları mevcuttur. MCMC yöntemlerinde yakınsamanın sağlanıp sağlanmadığı ya da ne kadarlık bir efordan sonra sağlanacağı hem teorik hem de pratik açıdan cevabı henüz net olarak verilememiş sorulardır. MCMC yöntemlerinde analitik olarak elde edilemeyen sonsal (ing: posterior) dağılımlardan örnekleme gerçekleştirilmesinde Metropolis-Hastings algoritması içerisinde kullanılan öneri (ing: proposal) dağılımlarının oluşturulması da yönlendirme gerektiren ayrı bir zorluk olarak karşımıza çıkmaktadır. Bu zorluklardan dolayı uygulamada etkin çalışmayan Markov zincirleri ile sıklıkla karşılaşmaktadır. Bu bağlamda, bu çalışmanın diğer bir amacı doğrultusunda, daha iyi hata kontrolü ve yakınsama özelliklerine sahip, hesaplama gereksinimleri açısından MCMC yöntemleri ile rekabet edebilecek, stokastik oynaklık kestirimi alanında daha önce hiç kullanılmamış yeni bir yöntem olan sparse grid integration (SGI) tabanlı kestirim algoritmaları geliştirilmiş ve değerlendirilmiştir. SGI yöntemi, geleneksel nümerik tümlevleme yönteminin çok boyutlu problemlere boyutsallığın olumsuz etkisinin azaltılarak genişletilmesi esasına dayanan ve geleneksel nümerik tümlevleme yöntemlerinin aksine çok boyutlu durumlarda uygulanabilen yöntemlerdir. Geleneksel nümerik tümlevleme yöntemleri tek değişkenli stokastik oynaklık modelleri için birkaç çalışmada incelenmiş olmakla beraber ÇDSO modelleri için nümerik tümlevleme yöntemleri, çok boyutluluğun bu yöntemlerdeki sınırlamaları sebebiyle, yazında göz ardı edilmiş ve yeterince incelenmemiştir. Doğrusal olmayan durum uzayı çalışmalarında nümerik tümlevlemeye dayanan kestirim yöntemleri deterministik yapıları sebebiyle yakınsama ile hata kontrol özellikleri, olasılıksal yöntemler olan MCMC ve diğer Monte Carlo yöntemlerinden daha üstündürler ve bunun yansıması olarak kesin (ing: exact) yöntemler olarak ifade edilirler. Önerilen SGI kestirim yaklaşımıyla, ÇDSO modelleri için bahsedilen kesinliğin en azından belirli bir ölçüde yakalanması amaçlanmıştır.

Stokastik oynaklık kestirimi için önerilen SGI tabanlı algoritmalar simüle edilmiş ve gerçek piyasa verileri üzerinde uygulanmış ve karşılaştırmalı olarak değerlendirilmiştir. Önerilen SGI tabanlı algoritmalar belirli koşullar altında MCMC tabanlı yöntemlerin performansını yakalamış ve hatta geçmiştir. SGI yöntemi gibi nümerik tümlevleme yöntemlerinin başta MCMC olmak üzere Monte Carlo tabanlı yöntemlere alternatif olabileceği gösterilmiştir.

Stokastik oynaklık modellerin kestiriminde kullanılan hem MCMC tabanlı hem de önerilen SGI tabanlı yöntemlerin işlem yoğunluğu ve hesaplama gereksinimleri oldukça fazladır. Bu bağlamda, çalışmanın üçüncü ve son amacı doğrultusunda incelenen MCMC ve SGI tabanlı kestirim algoritmaları için paralel hesaplama yaklaşımları ve algoritmaları oluşturulmuş ve bu yaklaşımlar kullanılarak grafik işlemciler (ing: graphics processing unit, GPU) üzerinde çalışan programlar

geliştirilerek bu cihazların hesaplama yönünden kestirim görevlerine katkıları değerlendirilmiştir.

Gerçekleştirilen simülasyon çalışmasında GPU üzerinde çalışan paralel algoritmaların işlem zamanlarını önemli biçimde azalttığı görülmüştür. Tek GPU üzerinde MCMC tabanlı algoritmalarda 16 kata kadar ve SGI tabanlı algoritmalarda 25 kata kadar hızlanma kaydedilmiştir. Tek GPU üzerinde uygulama teorik hızlanma sınırlarını ve ölçeklenebilirliği test etmek için yeterli olmamakla birlikte elde edilen sonuçlar daha büyük paralel mimarilerde uygulamalar için umut vericidir. GPU desteğinin, pratik stokastik oynaklık kestirimi uygulamaları için oldukça fark yaratabilecek etkin ve ucuz bir çözüm olduğu gösterilmiştir.



1. INTRODUCTION

Modeling, analysis, and estimation of volatilities of asset returns in financial markets have been a major research area for the last three decades because of the prominent role of volatility concept in mathematical and quantitative finance. Reliable volatility estimates of asset returns are indispensable inputs to several mathematical models in financial frameworks including but not limited to risk management and measurement, option pricing, portfolio and asset management. For example risk metrics such as the value at risk (VaR) used by many financial institutions for measuring the risk are directly calculated using the volatility forecasts. In option pricing models including the famous model of Black and Scholes (1973) and portfolio optimization models volatility and its estimates are direct inputs.

A considerably rich literature on volatility research showed that volatilities and correlations regarding financial asset returns are time varying with persistent dynamics. In addition to the time varying nature, various patterns and properties inherent in asset returns and volatilities were well studied and reported in the literature including leverage effects and volatility spillovers which are referred as stylized facts. Analysis of time varying structures and tools for addressing the stylized facts about volatility for asset returns are central to the contemporary volatility estimation research.

Volatility modeling research field has two main branches having different modeling approaches to address the mentioned stylized facts. First branch deals with models which are called Autoregressive Conditional Heteroskedasticity (ARCH) models introduced by Engle (1982) and second branch deals with models so called Stochastic Volatility (SV) models introduced by Taylor (1982). The essential feature of ARCH type models is that they explicitly model the conditional variance of returns given the past returns whereas the SV models consider the conditional variance of returns as a separate stochastic process as latent variable instead of explicitly modeling it. Because of the modeling approach, SV models are quite flexible and versatile in capturing the stylized facts, however their nonlinear structure

bears computational challenges in estimation. SV models require linear approximations or computationally demanding numerical methods for the associated estimation problems. Despite their powerful features, the computational challenges in estimation of SV models prevented them to be popular in practice and resulted in the dominance of ARCH type models in the early research. However, with the advances in computational resources allowing the usage of computationally intensive algorithms and methods, SV models has started to draw attention in recent research.

Several extensions on the univariate SV models addressing the stylized facts have been studied and proposed after the SV model of Taylor (1982) which dealt only with volatility clustering. The first multivariate stochastic volatility (MSV) model due to Harvey et al. (1994) is followed by an appreciable amount of research composed of several multivariate model specifications addressing different and more complicated stylized facts not only about volatility but also about co-volatility and their multidimensional dynamics. The control mechanisms and parameterizations of the covariance and/or correlation matrices in MSV models and their handling in time-varying settings are the core topics of the MSV modeling research since almost all stylized facts are imposed through the structure of those matrices in MSV models.

While addressing the stylized facts and flexibility in model specifications, another objective was keeping the complexity under control and developing appropriate estimation methods in those MSV modeling efforts since dimensionality brought additional complexity on top of the inherent complexity due to the nonlinearity in SV models. Being nonlinear state space models, even univariate SV models require methods that can handle high dimensional integrals for obtaining smoothing, filtering and prediction estimates of time-varying volatilities and parameter estimates. An extra complexity is introduced in MSV models due to the dimensionality of latent volatilities.

For the estimation, several early studies incorporated practical algorithms providing either fast or simplified approximations based on the well-known Kalman filter and its extensions, using Laplace approximations, variations of moment matching and method of moments, and quasi likelihood methods. Although being fast and simple those methods generally suffered from poor performance. Illustrations and examples of these methods can be found in (Taylor, 1986), (Harvey et al., 1994), (Harvey & Shephard, 1996) and (Galant & Hsieh, 1997).

Poor results of linear approximation based methods and challenges in the numerical estimation of SV models incited the usage of computationally intensive simulation based Monte Carlo methods for better estimations and approximations in parallel with the advances in computational resources. Various Monte Carlo based methods incorporating the algorithms such as resampling, particle filters, rejection sampling and importance sampling have been proposed with examples in (Watanabe, 1999), (Tanizaki, 1997), (Carlin et al., 1992) and (Sandman & Koopman, 1998).

A major breakthrough in SV estimation research was started with the works of Tierney (1994), Chib and Greenberg (1995, 1996) which introduced the Markov Chain Monte Carlo (MCMC) methods to the econometrics and SV fields. MCMC methods including the influential Metropolis-Hastings and Gibbs sampling algorithms quickly became central to the SV modeling and estimation studies, and a vast amount of literature on the applications of different variations of MCMC methods on various types of SV models, especially the MSV models was built up. Particularly, MSV models have benefited from the MCMC methods since MCMC methods are immune to the curse of dimensionality by construction unlike the other Monte Carlo techniques and exact filter methods such as the numerical integration. Another advantage of MCMC was the ease of implementation of these methods in Bayesian settings where the parameter estimation can also be handled without a maximization routine for the likelihood, hence without an explicit evaluation of the likelihood function. These appealing features of MCMC methods made them a natural first choice in MSV estimation studies. However, MCMC algorithms are not flawless. They still require intense computational resources for complicated iterative sampling schemes for estimation. Although having a quite different philosophy than the other Monte Carlo methods they are still simulation based Monte Carlo methods, thus are not exact methods. Furthermore, certain issues on error control and convergence are inherent particularly for the MCMC methods. A detailed treatment of MCMC methods can be found in (Chib, 2001).

Multidimensional integrals arising in estimation of SV models can be handled by classical numerical integration methods as discussed in (Kitagawa, 1987) and (Tanizaki, 1997) in a nonlinear state space modeling framework. Being exact methods with a deterministic structure, convergence properties of classical numerical integration methods are superior to simulation based Monte Carlo methods.

However, when the state-space dimension increases as in MSV models, these methods become computationally infeasible since the number of dimensions increases the complexity of these type of algorithms exponentially. Unsurprisingly, studies on the application of the numerical integration methods to nonlinear state space models and particularly MSV models are quite rare compared to the approximation based methods and Monte Carlo simulation based methods including the MCMC methods.

Sparse grid integration (SGI) method is a smartly reshaped version of classical numerical integration method to handle multidimensional integrals by constructing multi-dimensional integration formulas in a way that the dimensionality effect is decreased to a certain extent which allows practical implementation in higher dimensional cases in contrast to the classical numeric integration methods. Sparse grid integration approach is based on the work of Smolyak (1963) and was applied to some economic and financial problems with examples of discrete choice analysis in (Bungarts and Griebel, 2004), collateral mortgage optimization problem in (Gerstner and Griebel, 1998), derivative and option pricing in (Gerstner, 2007) and asset liability in life insurance in (Holtz, 2010). However, estimation algorithms based on the SGI approach for SV models have been neither studied nor mentioned in the literature.

One of the mentioned advances in computational resources is the high performance computing paradigm on massively parallel architectures such as graphic processing units (GPUs) or compute processors hosting many processors. Advances in the capabilities of GPUs and the introduction of easier to use platforms and tools for programming such devices resulted in deployment of several scientific and industrial applications benefiting from the cheap and efficient computing power provided by those devices. Quantitative finance has always been one of the first fields quickly adopting new technologies. In this context, the potential contributions of the high performance computing paradigms on massively parallel architectures such as to the computationally demanding task of SV estimation is one of the focus of this study.

The objectives of the study are summarized as follows. First objective of this study is the search for alternative MSV model specifications that can capture the stylized facts and dynamics of asset returns in a more realistic and flexible way than the

available models in the literature and contributions from the MSV modeling perspective.

Second objective of the study is in the perspective of estimation methodology where estimation algorithms based on a different approach than the popular MCMC approach for the MSV models is studied to see whether it is possible or not to come up with an estimation approach that does not have the drawbacks of MCMC and provide better results. The SGI approach which is neglected in the SV field is the approach under question in this perspective.

Third and final objective of the study is the evaluation and assessment of the possible contributions and implications of the GPU computing and usage for easing the excessive computational burden in MSV estimation problems.

Organization of the study is as follows. In section 2, mathematical and theoretical background of the thesis is provided. After providing a brief overview of SV models, a novel MSV model specification is given in accordance with the first objective of the study. In the second part of section 2 an overview and background on estimation algorithms are presented first followed by detailed treatment and presentation of the MCMC based estimation algorithms and the proposed SGI based estimation algorithms for the second objective of the study.

In section 3, important topics on practical implementation of the estimation algorithms their computational aspects and parallelization approaches, particularly implementation with GPUs are discussed in accordance with the third objective of the study.

In section 4, the methodology followed in the study is presented. The section provides information about the software and hardware used in numerical applications, describes the simulation studies and analyses conducted and data sets used in the study.

Section 5 provides the results of the numerical applications and analyses for the proposed MSV model, proposed estimation algorithms and GPU implementations.

Section 6 concludes the study by compiling the important results followed by concluding remarks, comments and further research directions.



2. THEORETICAL AND MATHEMATICAL BACKGROUND

In this section, starting with an overview of literature on the foundations of mathematical construction of stochastic volatility models, a novel MSV model is developed and proposed in the first subsection. In the second subsection, estimation algorithms based on the MCMC method and the proposed SGI method for MSV models are developed and presented in detail after a literature review and some preliminaries on the MCMC and SGI methods.

To avoid confusion, the multiple integral notations and definitions regarding the multidimensional integrals with respect to vectors, sets of vectors and matrices frequently used throughout the study is provided in appendix A.

2.1 Stochastic Volatility Modeling

2.1.1 Overview of stochastic volatility models

SV model building has a natural flow starting from the construction of the basic univariate model, followed by the extensions on the basic univariate model and then construction of multivariate models with their extensions. Same flow is followed in this section.

First univariate stochastic volatility model in the literature is due to Taylor (1982) and detailed in (Taylor, 1986). The basic setup for modeling the changes in variance is to regard innovations in the mean as being a sequence of independent and identically distributed random variables, ε_t with zero mean and unit variance, multiplied by a factor $\sigma_t = \exp(h_t/2)$. The latent log-volatility, $h_t = \log(\sigma_t^2)$, is defined as a stationary first order autoregressive (AR(1)) process having an error term, η_t , with zero mean and variance σ_η , leading to the state-space model,

$$y_t = e^{(h_t/2)} \varepsilon_t, \quad (2.1)$$

$$h_{t+1} = \gamma + \phi h_t + \eta_t. \quad (2.2)$$

Here, equation 2.1 is known as measurement or observation equation, and equation 2.2 is the transition or state equation of the state space model. In this model, the measurements, y_t , are observable while the states, h_t , are unobservable (i.e. latent) variables. The univariate SV model, which is given by equation 2.1 and equation 2.2, is a state space model because it actually is a time varying parameter model. Furthermore, the multiplicative structure of equation 2.1 makes the model nonlinear. This basic univariate SV model, successfully captures the time varying variance and volatility clusterings observed in asset return series. The latent structure of the log-volatilities and the approach modeling the log-volatilities as a separate stochastic process makes the SV models flexible and versatile in capturing the stylized facts of asset return series. For further discussion on properties of the SV models see (Ghysels et al., 1996).

Not long after the first univariate model described above, several extensions to the basic univariate SV model were proposed in literature. An important extension to the basic univariate SV model was addressing the stylized fact called asymmetry or leverage effect. Leverage effect simply describes the negative correlation between the asset returns and volatility shocks. To capture the leverage effect, SV models with correlated errors were proposed and discussed in (Harvey and Shephard, 1996). Correlated errors model to address the leverage effect is given by,

$$\begin{aligned}
 y_t &= e^{(h_t/2)} \varepsilon_t, \\
 h_{t+1} &= \gamma + \phi h_t + \eta_t, \\
 \begin{pmatrix} \varepsilon_t \\ \eta_t \end{pmatrix} &\sim \mathbf{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & \rho\sigma_\eta \\ \rho\sigma_\eta & \sigma_\eta^2 \end{pmatrix} \right).
 \end{aligned} \tag{2.3}$$

In this specification, the parameter, ρ , is the correlation between ε_t and η_t representing the leverage effect. Typically, negative correlation implies that a negative return tends to increase the volatility of an asset price.

After the first multivariate stochastic volatility (MSV) model in the literature, given in (Harvey et al., 1994), several model specifications addressing the stylized facts such as correlated asset returns, leverage effects and volatility spillovers are proposed with examples in (Asai and McAleer, 2006) and (Ishihara and Omori, 2012). The specification of a general MSV model based on these studies can be cast as,

$$\begin{aligned} \mathbf{y}_t &= \mathbf{V}_{y,t}^{1/2} \boldsymbol{\varepsilon}_t, \\ \mathbf{h}_{t+1} &= \boldsymbol{\gamma} + \boldsymbol{\phi} \mathbf{h}_t + \boldsymbol{\eta}_t, \end{aligned} \quad (2.4)$$

where, $\mathbf{y}_t = (y_{1t}, \dots, y_{pt})'$ is the p -dimensional vector of asset returns and $\mathbf{h}_t = (h_{1t}, \dots, h_{pt})'$ is the p -dimensional vector of log-volatilities, $\boldsymbol{\gamma}_t = (\gamma_1, \dots, \gamma_p)'$ is the intercept parameter vector, and $\boldsymbol{\phi} = \text{diag}(\phi_1, \dots, \phi_p)$ is the diagonal matrix of persistence parameters. In equation 2.4, time-varying variances of returns are the diagonal entries of the diagonal matrix $\mathbf{V}_{y,t} = \text{diag}(\exp(h_{1t}), \dots, \exp(h_{pt}))$. The innovations $\boldsymbol{\varepsilon}_t = (\varepsilon_{1t}, \dots, \varepsilon_{pt})'$ and the disturbances $\boldsymbol{\eta}_t = (\eta_{1t}, \dots, \eta_{pt})'$ in equation 2.4 are related with each other through,

$$\begin{pmatrix} \boldsymbol{\varepsilon}_t \\ \boldsymbol{\eta}_t \end{pmatrix} \sim \text{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{\varepsilon\varepsilon} & \boldsymbol{\Sigma}_{\varepsilon\eta} \\ \boldsymbol{\Sigma}_{\eta\varepsilon} & \boldsymbol{\Sigma}_{\eta\eta} \end{pmatrix} \right) \text{ and } \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{\varepsilon\varepsilon} & \boldsymbol{\Sigma}_{\varepsilon\eta} \\ \boldsymbol{\Sigma}_{\eta\varepsilon} & \boldsymbol{\Sigma}_{\eta\eta} \end{pmatrix}, \quad (2.5)$$

where, the covariance matrix, $\boldsymbol{\Sigma}$ defines the relationship between asset returns and log-volatilities. Here, depending on the structure of the covariance matrix, $\boldsymbol{\Sigma}$, the model in equation 2.4 and equation 2.5 can address various stylized facts:

- If the off-diagonal elements of $\boldsymbol{\Sigma}_{\varepsilon\varepsilon}$ are nonzero then there is correlation between asset returns.
- If the off-diagonal elements of $\boldsymbol{\Sigma}_{\eta\eta}$ are nonzero then there is volatility spillover.
- If the diagonal elements of $\boldsymbol{\Sigma}_{\eta\varepsilon}$ (and $\boldsymbol{\Sigma}_{\varepsilon\eta}$) are nonzero then there is leverage effect.
- If the off-diagonal element of $\boldsymbol{\Sigma}_{\eta\varepsilon}$ (and $\boldsymbol{\Sigma}_{\varepsilon\eta}$) is nonzero then there is cross-leverage effect.

The general MSV model described in equation 2.4 and equation 2.5 will be referred as the MSV-G model throughout this study. And the special case of the MSV-G model with

$$\begin{pmatrix} \boldsymbol{\varepsilon}_t \\ \boldsymbol{\eta}_t \end{pmatrix} \sim \text{N} \left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} \boldsymbol{\Sigma}_{\varepsilon\varepsilon} & 0 \\ 0 & \mathbf{V}_{\eta\eta} \end{pmatrix} \right) \text{ and } \boldsymbol{\Sigma} = \begin{pmatrix} \boldsymbol{\Sigma}_{\varepsilon\varepsilon} & 0 \\ 0 & \mathbf{V}_{\eta\eta} \end{pmatrix}, \quad (2.6)$$

where, the asset returns are correlated with no leverage effects (i.e. $\Sigma_{\eta\varepsilon} = \Sigma_{\varepsilon\eta} = 0$) and no volatility spillovers (i.e. $\Sigma_{\eta\eta} = \mathbf{V}_{\eta\eta}$ is diagonal), will be referred as MSV-B representing the basic multivariate case.

The MSV-G model becomes quite complicated in terms of number of parameters as the dimension increases. To offer more parsimonious model structures, a class of MSV models based on factor analysis were proposed in the literature. The additive factor model was first introduced in (Harvey, Ruiz, and Shephard, 1994). Another factor model can be found in (Jacquier et al., 1995). The basic idea in those MSV models is originated from factor decomposition of covariance structures in multivariate analysis, where returns are decomposed into additive or multiplicative components. The additive K factor MSV model can be written as

$$\begin{aligned} \mathbf{y}_t &= \mathbf{D}\mathbf{f}_t + \boldsymbol{\varsigma}_t & \boldsymbol{\varsigma}_t &\sim \mathbf{N}(0, \mathbf{V}_\varepsilon), \\ f_{i,t} &= e^{h_{i,t}/2} \varepsilon_{i,t} & \varepsilon_{i,t} &\sim \mathbf{N}(0,1), \\ h_{i,t+1} &= \alpha_i + \phi_i h_{i,t} + \eta_{i,t} & \eta_{i,t} &\sim \mathbf{N}(0, \sigma_\eta^2), \end{aligned} \quad (2.7)$$

where, f_t is $K \times 1$ vector of factors ($K < p$) and \mathbf{D} is a $p \times K$ matrix of factor loadings. In this model, $\mathbf{V}_\varepsilon = \text{diag}(\sigma_1^2, \dots, \sigma_p^2)$ and the variance of y_t is given by

$$\mathbf{V} = \mathbf{D}\Sigma_f\mathbf{D}' + \mathbf{V}_\varepsilon, \quad (2.8)$$

which is always positive definite by construction. While being parsimonious models which is an important advantage, the main drawback of the factor models is the difficulty in interpretability because of the implicit structure.

One of the consequences of the factor model given in equation 2.7 and equation 2.8 is that the conditional correlations of asset returns are actually time varying as well as the variance (Asai et al., 2006). Based on that fact, a class of MSV models capturing the time-varying correlations without factor structure were proposed and studied. These studies let either the covariance (correlation) matrix, $\Sigma_{\varepsilon\varepsilon}$, in equation 2.5 or the covariance of asset returns, Σ_{yy} , vary in time, often in a dynamic mechanism that ensure the positive definiteness and symmetry properties of the covariance matrix.

Thus, the dynamic mechanisms used to handle the covariance matrices are the main focus of these studies. Tsay (2002) and Lopes et al. (2011) are examples that use

Cholesky decomposition of the covariance matrix as such a mechanism by letting the covariance matrix of asset returns, Σ_{yy} , dynamically change with the relation,

$$\Sigma_{yy,t} = \mathbf{L}_t \mathbf{D}_t \mathbf{L}_t', \quad (2.9)$$

where, \mathbf{L}_t is a lower triangular matrix and \mathbf{D}_t is a diagonal matrix. Here the elements of both \mathbf{L}_t and \mathbf{D}_t are obtained with separate autoregressive processes like the log-volatilities in the MSV-B model. This approach directly models the covariance matrix of asset returns and there is no separation between correlations and variances, hence the log-volatilities are not explicitly modeled in autoregressive processes.

To achieve and keep the positive definiteness and symmetry, Asai and McAleer (2009) and Ishihara et al. (2014) incorporated matrix exponential, which is defined by

$$\exp(\mathbf{A}) \equiv \sum_{s=0}^{\infty} \frac{1}{s!} \mathbf{A}^s, \quad (2.10)$$

using the fact that for any real symmetric matrix \mathbf{A} , $\exp(\mathbf{A})$ is also a symmetric positive definite matrix. As in the Cholesky approach, this approach also directly handles the covariance matrix and does not model log-volatilities with explicit autoregressive processes, instead rotations found in principal component analysis (PCA) is used to obtain log-volatilities.

Another specification for dynamic structure is given by Gouriéroux et al. (2009) accommodating Wishart autoregressive process which is an AR process constructed on covariance matrices, thus satisfies the symmetry and positivity requirements.

All the approaches for dynamic structures in the literature usually restrict the dynamic structure with covariance or correlation of asset returns. Because all of the mechanisms ensuring positive definiteness and symmetry in the previous studies either implicitly model the variance or do not separate the dynamics of all correlations and variances, those approaches do not have the flexibility to address more complicated dynamic structures such as leverage effects and volatility spillovers. Thus, the model specifications and dynamic mechanisms proposed in the previous studies are somewhat restrictive in terms of flexibility and versatility considering the available options on dynamic components to be included in the

model. Based on this perspective, in section 2.1.2 a new MSV model specification is proposed to overcome these drawbacks.

Models with distributions having thicker tails than the Gaussian distribution for the observation disturbances, ε_t , in both univariate and multivariate settings are other noteworthy contributions in the previous studies. The Student's t-distribution were used to address leptokurtosis that arise in some financial series with examples in (Galant et al., 1997), (Sandman and Koopman, 1998), (Ishihara and Omori, 2012) and (Ishihara et al., 2014).

2.1.2 A novel multivariate stochastic volatility model (MSV-D)

In this section, a new MSV model specification is proposed based on the considerations about the available model specifications in the literature mentioned in section 2.1.1. The proposed model specification is a general specification that can accommodate the following stylized facts:

- Correlations between asset returns,
- Leverage effects (i.e. correlation between a particular asset return and its volatility),
- Cross-leverage effects (i.e. correlations between a particular asset and other assets' volatilities),
- Volatility spillovers (i.e. correlations between log-volatilities),

with both constant and dynamic settings (or their mixtures) for each stylized fact, which offers substantial flexibility, versatility and freedom in modeling preferences without the restrictions inherent in the available models in the literature.

One of the main differences of the proposed model from the models discussed in section 2.1.1 is the separation of variance and correlation components in the modeling approach. This separation allows explicit modeling of time-varying variance as in the MSV-G model while dynamic structures can still be incorporated unlike the dynamic models in the literature discussed in section 2.1.1

Let $\mathbf{y}_t = (y_{1t}, \dots, y_{pt})'$ be p -dimensional vector of stock returns, then the proposed MSV model starts with

$$\begin{pmatrix} \mathbf{y}_t \\ \boldsymbol{\eta}_t \end{pmatrix} \sim \mathbf{N}(0, \boldsymbol{\Sigma}_t), \quad \boldsymbol{\Sigma}_t = \mathbf{V}_t^{1/2} \mathbf{P}_t \mathbf{V}_t^{1/2}, \quad t=1, \dots, T, \quad (2.11)$$

where, the time-varying variance matrix is

$$\mathbf{V}_t = \begin{pmatrix} \mathbf{V}_{y,t} & \mathbf{0} \\ \mathbf{0} & \mathbf{V}_\eta \end{pmatrix} = \begin{pmatrix} e^{h_{1,t}} & \cdot & 0 & 0 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & e^{h_{p,t}} & 0 & \cdot & 0 \\ 0 & \cdot & 0 & \sigma_{\eta,1}^2 & \cdot & 0 \\ \cdot & \cdot & \cdot & \cdot & \cdot & \cdot \\ 0 & \cdot & 0 & 0 & \cdot & \sigma_{\eta,p}^2 \end{pmatrix}, \quad (2.12)$$

the time-varying correlation matrix is

$$\mathbf{P}_t = \begin{pmatrix} P_{yy,t} & P_{y\eta,t} \\ P_{\eta y,t} & P_{\eta\eta,t} \end{pmatrix}, \quad (2.13)$$

and the time-varying covariance matrix is

$$\boldsymbol{\Sigma}_t = \begin{pmatrix} \boldsymbol{\Sigma}_{yy,t} & \boldsymbol{\Sigma}_{y\eta,t} \\ \boldsymbol{\Sigma}_{\eta y,t} & \boldsymbol{\Sigma}_{\eta\eta,t} \end{pmatrix}. \quad (2.14)$$

The log-volatilities, $\mathbf{h}_t = (h_{1,t}, \dots, h_{p,t})'$, are driven by the following AR(1) process:

$$\mathbf{h}_{t+1} = \boldsymbol{\gamma} + \boldsymbol{\phi} \mathbf{h}_t + \boldsymbol{\eta}_t, \quad \boldsymbol{\eta}_t \sim \mathbf{N}(0, \boldsymbol{\Sigma}_{\eta\eta,t}) \quad (2.15)$$

for $t=1, \dots, T$ where, $\boldsymbol{\gamma}_t = (\gamma_1, \dots, \gamma_p)'$ is the intercept parameter vector, $\boldsymbol{\eta}_t = (\eta_1, \dots, \eta_p)'$ is the vector of disturbances on the log-volatilities having zero mean and covariance matrix, $\boldsymbol{\Sigma}_{\eta\eta,t}$, and $\boldsymbol{\phi} = \text{diag}(\phi_1, \dots, \phi_p)$ is the diagonal matrix of persistence parameters. In equation 2.15, the process mean, $\boldsymbol{\mu}_h$ is given by

$$\boldsymbol{\mu}_h = (\mathbf{I} - \boldsymbol{\phi})^{-1} \boldsymbol{\gamma}, \quad (2.16)$$

and the variance matrix of the process, \mathbf{V}_h , satisfying the stationarity condition,

$$\mathbf{V}_h = \boldsymbol{\phi} \mathbf{V}_h \boldsymbol{\phi} + \mathbf{V}_\eta, \quad (2.17)$$

is given by

$$\text{Vec}(\mathbf{V}_h) = (\mathbf{I} - \boldsymbol{\phi} \otimes \boldsymbol{\phi})^{-1} \text{Vec}(\mathbf{V}_\eta). \quad (2.18)$$

To ensure the positive semi-definiteness and symmetry of the correlation matrices through time periods the dynamic correlation matrix, \mathbf{P}_t , is parameterized as follows. Positivity is achieved by

$$\mathbf{P}_t = \mathbf{B}_t \mathbf{B}_t' \quad (2.19)$$

where the $(2p \times 2p)$ matrix \mathbf{B}_t can be obtained by Cholesky decomposition and is in the form,

$$\mathbf{B}_t = \begin{pmatrix} b_{1,1,t} & b_{1,2,t} & \cdot & b_{1,2p,t} \\ b_{2,1,t} & b_{2,2,t} & \cdot & 0 \\ \cdot & \cdot & 0 & 0 \\ b_{2p,1,t} & 0 & 0 & 0 \end{pmatrix}, \quad (2.20)$$

with entries, $b_{i,j,t}$, obtained by the relation,

$$b_{i,j,t} = \cos(\alpha_{i,j,t}) \prod_{k=1}^{j-1} \sin(\alpha_{i,k,t}), \quad 0 \leq \alpha_{i,j,t} \leq \pi, \quad (2.21)$$

where the angles $\alpha_{i,j,t}$, are the entries of the $(2p \times 2p)$ matrix, \mathbf{A}_t , given by

$$\mathbf{A}_t = \begin{pmatrix} \alpha_{1,1,t} & \cdot & \alpha_{1,2p-1,t} & 0 \\ \cdot & \cdot & 0 & 0 \\ \alpha_{2p-1,1,t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \quad (2.22)$$

A suitable transformation which maps the angles, $\alpha_{i,j,t}$, which take value in the interval $[0, \pi]$, to the interval $[-\infty, \infty]$ is the logit function given by

$$q_{i,j,t} = \log\left(\frac{\alpha_{i,j,t}}{\pi - \alpha_{i,j,t}}\right), \quad (2.23)$$

where $q_{i,j,t}$ are the entries of the $(2p \times 2p)$ matrix \mathbf{R}_t in the form,

$$\mathbf{R}_t = \begin{pmatrix} q_{1,1,t} & \cdot & q_{1,2p-1,t} & 0 \\ \cdot & \cdot & 0 & 0 \\ q_{2p-1,1,t} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}. \quad (2.24)$$

If the nonzero entries of the matrix \mathbf{R}_t are stacked column-wise in to a vector, and letting $r = p \times (2p - 1)$ then the vector $\mathbf{q}_t = (q_{1t}, \dots, q_{rt})'$ is obtained which has $r = p \times (2p - 1)$ elements.

Another AR(1) process, driving the dynamic correlations through \mathbf{q}_t can then be stated as

$$\mathbf{q}_{t+1} = \boldsymbol{\delta} + \boldsymbol{\theta}\mathbf{q}_t + \boldsymbol{\omega}_t, \quad \boldsymbol{\omega}_t \sim N(0, \mathbf{V}_\omega) \quad (2.25)$$

where, $\mathbf{V}_\omega = \text{diag}(\sigma_{\omega,1}^2, \dots, \sigma_{\omega,r}^2)$ is the diagonal variance matrix of the process error vector, $\boldsymbol{\omega}_t = (\omega_{1,t}, \dots, \omega_{r,t})'$, $\boldsymbol{\delta} = (\delta_1, \dots, \delta_r)'$ are the intercept parameter vector and $\boldsymbol{\theta} = \text{diag}(\theta_1, \dots, \theta_r)$ is the diagonal persistence parameter matrix. In equation 2.25 the process mean is given by

$$\boldsymbol{\mu}_q = (\mathbf{I} - \boldsymbol{\theta})^{-1} \boldsymbol{\delta}, \quad (2.26)$$

and the variance matrix of the process, \mathbf{V}_q satisfying the stationarity condition,

$$\mathbf{V}_q = \boldsymbol{\theta}\mathbf{V}_q\boldsymbol{\theta} + \mathbf{V}_\omega, \quad (2.27)$$

is given by

$$\text{Vec}(\mathbf{V}_q) = (\mathbf{I} - \boldsymbol{\theta} \otimes \boldsymbol{\theta})^{-1} \text{Vec}(\mathbf{V}_\omega). \quad (2.28)$$

In this model, equation 2.15 drives the time varying volatilities of asset returns while equation 2.25 is separately driving the time varying correlations between asset returns and log-volatilities. The transformation starting from equation 2.19 to equation 2.24 maps the correlation matrix, \mathbf{P}_t , to the vector, \mathbf{q}_t , and furthermore this mapping is one-to-one and reversible. A schematic illustration of the transformation is given in Figure B.1 in appendix B. The parameterization of correlation matrix from equation 2.19 to 2.22 is a modified version of the transformation given by Robenato and Jäckel (2011) and Kercheval (2008) in a general perspective. Any correlation matrix \mathbf{P}_t can be transformed into a vector \mathbf{q}_t through this transformation and moreover any real valued vector \mathbf{q}_t of dimension $p \times (2p - 1)$ can be mapped to a $(2p \times 2p)$ unique correlation matrix, \mathbf{P}_t , by reversing the transformation described in equation 2.19 to equation 2.24 and incorporating the inverse logit,

$$\alpha_{i,j,t} = \frac{\pi}{e^{-q_{i,j,t}} + 1}, \quad (2.29)$$

for obtaining the angle mapping of a given vector \mathbf{q}_t .

In the model, positivity is achieved by equation 2.19 and the relation in equation 2.21 ensures entries in the diagonal of the resulting correlation matrix to be 1 and off-diagonals to be in the $(-1, 1)$ interval.

The proposed model described above is quite general and flexible since it is possible to address several stylized facts while restricting some of them. Fixing some of the entries of vector \mathbf{q}_t as constants instead of associating with an AR(1) process, allows easy removal of dynamic components and replacement with static counterparts of these components and setting some of the elements of vector \mathbf{q}_t to zero allows easy removal of particular stylized facts mentioned from the model. For example, fixing the vector, \mathbf{q}_t , by setting,

$$\mathbf{q}_t = \boldsymbol{\delta} = (\delta_1, \dots, \delta_r)', \quad (2.30)$$

reduces the model to the MSV-G model. And similarly if appropriate elements of \mathbf{q}_t , in equation 2.30 are set to zero then the MSV-B model can be obtained.

Parsimony can be kept under control by removing certain properties and replacing dynamic structures with static counterparts through the structure of vector \mathbf{q}_t as mentioned above. Another approach to keep the parsimony is putting restrictions on the parameters of the AR(1) process given in equation 2.25 which essentially drives the dynamic correlations. It is possible for example to set a single scalar value for the persistence parameter vector $\boldsymbol{\theta} = \text{diag}(\theta_1, \dots, \theta_r)$ by restricting its entries to be equal to each other, which significantly decreases the number of parameters. The proposed MSV model in this section will be referred as MSV-D in the next sections of this study.

2.2 Estimation Algorithms for Stochastic Volatility Models

2.2.1 Stochastic volatility models as nonlinear state space models

Any given SV model such as the MSV-B, MSV-G or the proposed MSV-D models are essentially nonlinear state space models since they are time varying parameter models. For a given state space model, the main estimation problem regarding the

states (e.g. latent log-volatilities, \mathbf{h}_t and correlation states, \mathbf{q}_t in the MSV-D model) is finding the conditional expectation of these states using the information (e.g. asset returns, \mathbf{y}_t) set up to time s , $\mathbf{Y}_s = \{\mathbf{y}_1, \dots, \mathbf{y}_s\}$. The estimation problem associated with the log-volatilities, \mathbf{h}_t , and correlation states, \mathbf{q}_t , for the MSV-D model can be formally stated as

$$(\mathbf{h}_{t|s}, \mathbf{q}_{t|s})' = E\left((\mathbf{h}_t, \mathbf{q}_t)' \mid \mathbf{Y}_s, \boldsymbol{\Omega}\right), \quad (2.31)$$

where $\boldsymbol{\Omega} = \{\boldsymbol{\delta}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega\}$ is the model parameter set.

Depending on the timing of the information set, \mathbf{Y}_s , the estimation problems associated with the states given the parameters are as follows:

- If $t = s$ then the estimation problem is called filtering,
- If $t > s$ then the estimation problem is called prediction,
- If $t < s$ then the estimation problem is called smoothing.

In addition to these three estimation problems another estimation problem is the parameter estimation problem. Thus, there are essentially four fundamental problems of estimation for any SV model.

In this section, density based estimation algorithms for the above estimation problems of general MSV-D model are developed. The algorithms are developed based on the common state space modeling and analysis approach for the univariate cases that can be found in (Kitagawa, 1987) and (Tanizaki, 1997).

2.2.1.1 Densities implied by the MSV-D model

For notational simplicity let $\boldsymbol{\Omega} = \{\boldsymbol{\Omega}_h, \boldsymbol{\Omega}_q\}$ be the set of model parameters, where $\boldsymbol{\Omega}_h = \{\boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta\}$ and $\boldsymbol{\Omega}_q = \{\boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega\}$ are the separate parameter sets associated with the log-volatilities \mathbf{h}_t and correlation states, \mathbf{q}_t respectively.

One step transition density of the log-volatilities, \mathbf{h}_t , from the state space model of MSV-D is given by

$$\begin{aligned}
p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) &= \\
(2\pi)^{-\frac{p}{2}} |\boldsymbol{\Sigma}_{\eta\eta,t}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{h}_{t+1} - \boldsymbol{\Phi}\mathbf{h}_t - \boldsymbol{\gamma})' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\mathbf{h}_{t+1} - \boldsymbol{\Phi}\mathbf{h}_t - \boldsymbol{\gamma})\right), & \quad (2.32) \\
p_h(\mathbf{h}_1) &= (2\pi)^{-\frac{p}{2}} |\mathbf{V}_h|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{h}_1 - \boldsymbol{\mu}_h)' \mathbf{V}_h^{-1} (\mathbf{h}_1 - \boldsymbol{\mu}_h)\right),
\end{aligned}$$

for $t=1, \dots, T-1$.

Similarly one step transition density of correlation states \mathbf{q}_t is given by

$$\begin{aligned}
p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \boldsymbol{\Omega}_q) &= \\
(2\pi)^{-\frac{r}{2}} |\mathbf{V}_\omega|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{q}_{t+1} - \boldsymbol{\Theta}\mathbf{h}_t - \boldsymbol{\delta})' \mathbf{V}_\omega^{-1} (\mathbf{q}_{t+1} - \boldsymbol{\Theta}\mathbf{h}_t - \boldsymbol{\delta})\right), & \quad (2.33) \\
p_q(\mathbf{q}_1) &= \\
(2\pi)^{-\frac{r}{2}} |\mathbf{V}_q|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{q}_1 - \boldsymbol{\mu}_q)' \mathbf{V}_q^{-1} (\mathbf{q}_1 - \boldsymbol{\mu}_q)\right),
\end{aligned}$$

for $t=1, \dots, T-1$.

One step conditional density of \mathbf{y}_t implied by the state space representation of the MSV-D model is given by

$$\begin{aligned}
p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) &= \\
(2\pi)^{-\frac{p}{2}} |\boldsymbol{\Sigma}_{y|\eta,t}|^{-1/2} \exp\left(-\frac{1}{2}(\mathbf{y}_t - \boldsymbol{\mu}_{y|\eta,t})' \boldsymbol{\Sigma}_{y|\eta,t}^{-1} (\mathbf{y}_t - \boldsymbol{\mu}_{y|\eta,t})\right), & \quad (2.34)
\end{aligned}$$

where,

$$\boldsymbol{\Sigma}_{y|\eta,t} = \begin{cases} \boldsymbol{\Sigma}_{yy,t} - \boldsymbol{\Sigma}_{y\eta} \boldsymbol{\Sigma}_{\eta\eta}^{-1} \boldsymbol{\Sigma}_{\eta y} & , \text{ if } t < T \\ \boldsymbol{\Sigma}_{yy,t} & , \text{ if } t = T, \end{cases} \quad (2.35)$$

and

$$\boldsymbol{\mu}_{y|\eta,t} = \begin{cases} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\mathbf{h}_{t+1} - \boldsymbol{\Phi}\mathbf{h}_t - \boldsymbol{\gamma}) & , \text{ if } t < T \\ 0 & , \text{ if } t = T. \end{cases} \quad (2.36)$$

for $t=1, \dots, T-1$.

One step joint density of log-volatilities \mathbf{h}_t and correlation states, \mathbf{q}_t can be obtained from equation 2.32 and equation 2.33 as

$$\begin{aligned}
p(\mathbf{h}_{t+1}, \mathbf{q}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}) &= \\
p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \boldsymbol{\Omega}_q). & \quad (2.37)
\end{aligned}$$

Let $\mathbf{Y}_s = \{\mathbf{y}_1, \dots, \mathbf{y}_s\}$ be the observation up to time s , $\mathbf{H}_s = \{\mathbf{h}_1, \dots, \mathbf{h}_s\}$ be the set of log-volatility state vectors up to time s , and $\mathbf{Q}_s = \{\mathbf{q}_1, \dots, \mathbf{q}_s\}$ be the set of correlation state vectors up to time s , then, using the one step transition density given in equation 2.32, the conditional density of log-volatilities over a fixed periods of time is given by

$$p_H(\mathbf{H}_t | \mathbf{Q}_t, \Omega_h) = p_h(\mathbf{h}_1) \prod_{s=1}^{t-1} p_h(\mathbf{h}_{s+1} | \mathbf{h}_s, \mathbf{q}_s, \Omega_h). \quad (2.38)$$

Similarly, based on the one step transition density given in equation 2.33, conditional density of the correlation states over a fixed periods of time is given by

$$p_Q(\mathbf{Q}_t | \Omega_q) = p_q(\mathbf{q}_1) \prod_{s=1}^{t-1} p_q(\mathbf{q}_{s+1} | \mathbf{q}_s, \Omega_q). \quad (2.39)$$

Finally, based on the one period density given in equation 2.34, conditional density of the observations over a fixed periods of time is given by

$$p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h) = p_y(\mathbf{y}_t | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) \prod_{s=1}^{t-1} p_y(\mathbf{y}_s | \mathbf{h}_{s+1}, \mathbf{h}_s, \mathbf{q}_s, \Omega_h). \quad (2.40)$$

Several conditional joint densities of interest can be built upon the densities provided so far. One of them is the joint density of log-volatilities \mathbf{H}_t and correlation states, \mathbf{Q}_t which can be obtained from equation 2.38 and equation 2.39 as

$$p(\mathbf{H}_t, \mathbf{Q}_t | \Omega) = p_Q(\mathbf{Q}_t | \Omega_q) p_H(\mathbf{H}_t | \mathbf{Q}_t, \Omega_h). \quad (2.41)$$

Another important density used in estimation algorithms is the joint density of log-volatilities, \mathbf{H}_t and correlation states, \mathbf{Q}_t which can be obtained as

$$p(\mathbf{H}_t, \mathbf{Q}_t, \mathbf{Y}_t | \Omega) = p_Q(\mathbf{Q}_t | \Omega_q) p_H(\mathbf{H}_t | \mathbf{Q}_t, \Omega_h) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h). \quad (2.42)$$

2.2.1.2 Filtering, prediction and smoothing

Using the one step densities in equation 2.32 to equation 2.37 which are obtained from the state space representation of the MSV-D model and letting $p_h(\mathbf{h}_1 | \mathbf{h}_0, \mathbf{q}_0, \gamma, \phi, \mathbf{V}_\eta) = p_h(\mathbf{h}_1)$ and $p_q(\mathbf{q}_1 | \mathbf{q}_0, \delta, \theta, \mathbf{V}_\omega) = p_q(\mathbf{q}_1)$ density based filtering algorithm for the log-volatilities of MSV-D model can be constructed as a recursive algorithm as,

$$\begin{aligned}
p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_{t-1}, \Omega) &= \\
&\int \int p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{q}_{t-1}, \Omega_h) p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \Omega_q) p(\mathbf{h}_{t-1}, \mathbf{q}_{t-1} | \mathbf{Y}_{t-1}, \Omega) d\mathbf{h}_{t-1} d\mathbf{q}_{t-1}, \\
p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_t, \Omega) &= \\
&\frac{\int p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_{t-1}, \Omega) d\mathbf{h}_{t+1}}{\int \int \int p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_{t-1}, \Omega) d\mathbf{h}_{t+1} d\mathbf{h}_t d\mathbf{q}_t},
\end{aligned} \tag{2.43}$$

for $t = 1, \dots, T$. A non-recursive equivalent form of the filtering algorithm in equation 2.43 can be stated as

$$\begin{aligned}
p(\mathbf{H}_t, \mathbf{Q}_t | \mathbf{Y}_t, \Omega) &= \\
&\frac{p_Q(\mathbf{Q}_t | \Omega_q) p_H(\mathbf{H}_t | \mathbf{Q}_t, \Omega_h) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h)}{\int \int p_Q(\mathbf{Q}_t | \Omega_q) p_H(\mathbf{H}_t | \mathbf{Q}_t, \Omega_h) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h) d\mathbf{H}_t d\mathbf{Q}_t} \\
p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_t, \Omega) &= \int p(\mathbf{H}_t, \mathbf{Q}_t | \mathbf{Y}_t, \Omega) p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) d\mathbf{H}_{t-1} d\mathbf{Q}_{t-1} d\mathbf{h}_{t+1}.
\end{aligned} \tag{2.44}$$

Starting with the filtering density obtained in equation 2.43 for time t , density based L -step ahead prediction algorithm is obtained recursively by

$$\begin{aligned}
p(\mathbf{h}_{t+L}, \mathbf{q}_{t+L} | \mathbf{Y}_t, \Omega) &= \\
&\int \int p_h(\mathbf{h}_{t+L} | \mathbf{h}_{t+L-1}, \mathbf{q}_{t+L-1}, \Omega_h) p_q(\mathbf{q}_{t+L} | \mathbf{q}_{t+L-1}, \Omega_q) \\
&\quad p(\mathbf{h}_{t+L-1}, \mathbf{q}_{t+L-1} | \mathbf{Y}_t, \Omega) d\mathbf{h}_{t+L-1} d\mathbf{q}_{t+L-1}.
\end{aligned} \tag{2.45}$$

And equivalently prediction algorithm in equation 2.45 can be expressed in a non-recursive version as

$$\begin{aligned}
p(\mathbf{H}_{t+L}, \mathbf{Q}_{t+L} | \mathbf{Y}_t, \Omega) &= \\
&\frac{p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h) p_Q(\mathbf{Q}_{t+L} | \Omega_q) p_H(\mathbf{H}_{t+L} | \mathbf{Q}_{t+L}, \Omega_h)}{\int \int p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \Omega_h) p_Q(\mathbf{Q}_{t+L} | \Omega_q) p_H(\mathbf{H}_{t+L} | \mathbf{Q}_{t+L}, \Omega_h) d\mathbf{H}_{t+L} d\mathbf{Q}_{t+L}}, \\
p(\mathbf{h}_{t+L}, \mathbf{q}_{t+L} | \mathbf{Y}_t, \Omega) &= \int \int p(\mathbf{H}_{t+L}, \mathbf{Q}_{t+L} | \mathbf{Y}_t, \Omega) d\mathbf{H}_{t+L-1} d\mathbf{Q}_{t+L-1}.
\end{aligned} \tag{2.46}$$

Based on the filtering density at time t , obtained in equation 2.43, recursive algorithm for the smoothing density can be constructed as

$$\begin{aligned}
p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_T, \Omega) &= \\
&p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_t, \Omega) \times \\
&\int \frac{p(\mathbf{h}_{t+1}, \mathbf{q}_{t+1} | \mathbf{Y}_T, \Omega) p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \Omega_q)}{p(\mathbf{h}_{t+1}, \mathbf{q}_{t+1} | \mathbf{Y}_t, \Omega)} d\mathbf{h}_{t+1} d\mathbf{q}_{t+1},
\end{aligned} \tag{2.47}$$

for $t = T - 1, T - 2, \dots, 1$ which is a backward recursion. Let $\mathbf{H}_t = \{\mathbf{h}_1, \dots, \mathbf{h}_{t-1}, \mathbf{h}_{t+1}, \dots, \mathbf{h}_T\}$ and $\mathbf{Q}_t = \{\mathbf{q}_1, \dots, \mathbf{q}_{t-1}, \mathbf{q}_{t+1}, \dots, \mathbf{q}_T\}$ be the set of all state vectors from $t = 1, \dots, T$ excluding the state vector at time t , then the non-recursive version of the smoothing algorithm can be expressed as

$$p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_T, \Omega) = \frac{p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \Omega_h) p_Q(\mathbf{Q}_T | \Omega_q) p_H(\mathbf{H}_T | \mathbf{Q}_T, \Omega_h) d\mathbf{H}_T d\mathbf{Q}_T}{\int p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \Omega_h) p_Q(\mathbf{Q}_T | \Omega_q) p_H(\mathbf{H}_T | \mathbf{Q}_T, \Omega_h) d\mathbf{H}_T d\mathbf{Q}_T}. \quad (2.48)$$

The algorithms for the filtering, prediction and smoothing developed for MSV-D model are general and cover the models such as the MSV-B and MSV-G and their variations as well. For example, the recursive filtering algorithm in equation 2.43 reduces to

$$p(\mathbf{h}_t | \mathbf{Y}_{t-1}, \Omega) = \int p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \Omega_h) p(\mathbf{h}_{t-1} | \mathbf{Y}_{t-1}, \Omega_h) d\mathbf{h}_{t-1},$$

$$p(\mathbf{h}_t | \mathbf{Y}_t, \Omega) = \frac{p_y(\mathbf{y}_t | \mathbf{h}_t, \Omega_h) p(\mathbf{h}_t | \mathbf{Y}_{t-1}, \Omega_h)}{\int p_y(\mathbf{y}_t | \mathbf{h}_t, \mathbf{q}_t, \Omega_h) p(\mathbf{h}_t | \mathbf{Y}_{t-1}, \Omega_h) d\mathbf{h}_t}. \quad (2.49)$$

for the MSV-B model where the dynamic components and associated terms with the correlation states, \mathbf{q}_t , are dropped with the parameter set $\Omega_q = \{\delta, \theta, \mathbf{V}_\omega\}$, and the only parameter set Ω_h has become $\Omega_h = \{\gamma, \phi, \mathbf{V}_\eta, \delta\}$ where δ is the constant parameter for the correlations between asset returns in place of dynamic correlation states \mathbf{q}_t . In equation 2.49, another simplification is due to the lack of leverage effect in MSV-B which removes the temporal dependence between the asset returns \mathbf{y}_t and log-volatilities \mathbf{h}_{t+1} , thus the densities representing this dependence and associated differentials in the integrals are dropped as well. Similarly, for the MSV-G model algorithms can be obtained by only dropping the dynamic components as described above for the MSV-B. So density based estimation algorithms of most of the MSV models exhibiting any constant or dynamic correlation, leverage effects and volatility spillovers are special cases of the estimation algorithms developed in this section. Factor models or models implicitly modeling the log-volatilities are exceptions.

2.2.1.3 Mean, variance and likelihood

Once the density of interest (i.e. filtering, prediction or smoothing) is obtained with one of the algorithms described in the previous section, estimation of statistical properties such as the mean and the variance of a function $f(\mathbf{h}_u, \mathbf{q}_u)$ can be obtained with the expectation,

$$E(f(\mathbf{h}_v, \mathbf{q}_v) | \mathbf{Y}_s, \mathbf{\Omega}) = \int \int f(\mathbf{h}_v, \mathbf{q}_v) p(\mathbf{h}_v, \mathbf{q}_v | \mathbf{Y}_s) d\mathbf{h}_v d\mathbf{q}_v, \quad (2.50)$$

where, $(v, s) = (t, t)$, $(t + L, t)$, (T, T) for smoothing, filtering, and prediction respectively. For example, to obtain the vector of mean filtering estimates at time t , $f(\mathbf{h}_v, \mathbf{q}_v) = (\mathbf{h}_t, \mathbf{q}_t)'$ is used.

The log-likelihood function which plays an important role in parameter estimation for the MSV-D model can be obtained sequentially by

$$\begin{aligned} \log(p(\mathbf{Y}_T | \mathbf{\Omega})) = \\ \log\left(\prod_{t=1}^T \left(\int \int \int \frac{p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \mathbf{\Omega}_h) p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \mathbf{\Omega}_h)}{p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_{t-1}, \mathbf{\Omega})} d\mathbf{h}_{t+1} d\mathbf{h}_t d\mathbf{q}_t \right)\right), \end{aligned} \quad (2.51)$$

which is in fact readily available from the denominator of the second equation in the filtering algorithm given in equation 2.43 or equivalently,

$$\begin{aligned} \log(p(\mathbf{Y}_T | \mathbf{\Omega})) = \\ \log\left(\int \int p_Q(\mathbf{Q}_t | \mathbf{\Omega}_q) p_H(\mathbf{H}_t | \mathbf{Q}_t, \mathbf{\Omega}_h) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \mathbf{\Omega}_h) d\mathbf{H}_t d\mathbf{Q}_t\right), \end{aligned} \quad (2.52)$$

which is the denominator of the second equation in the filtering algorithm given in equation 2.44. Thus, the log-likelihood is a byproduct of the filtering algorithm for the current parameter set.

As is the case for estimation algorithms of section 2.2.1.2, the log-likelihood algorithms are also general.

2.2.2 Overview of the estimation methods

As it can be seen from the developed estimation algorithms in section 2.2.1, MSV models require handling of many high dimensional integrals arising in all kinds of estimation problems. To obtain estimates for the states (i.e. log-volatilities or correlation states in MSV-D), these states must be integrated out from the integral of

expressions composed of conditional densities or it is necessary to find ways to sample directly from the densities of interest. In addition to the high dimensionality, nonlinear nature of SV models makes it inefficient to use linear filters which offer closed-form solutions and quick algorithms. The high dimensional integrals and nonlinearity exists even in the univariate case and multivariate cases come with extra challenges.

Some of the first studies on estimation of SV models incorporated simplistic approximation methods based on the methods of moments and different versions of moment matching techniques with examples and references in (Ghysels et al., 1996).

A group of studies incorporated algorithms, providing fast approximations for log-volatility estimations based on the well-known Kalman filter and its extensions. An approach is modifying the state space model of the MSV so that the model becomes linear with distorted distributional structure. Applying the classical Kalman filter or the extended Kalman filter (EKF) on this transformed model can be used to obtain approximated estimates as discussed in (Harvey et al., 1994), (Tanizaki, 1997). Although being linear resulting in unsatisfactory approximations, Kalman filter based estimates provide good basis or starting points for more advanced methods.

For the general MSV-G model (and for MSV-D) the linearization is obtained by taking the square and then logarithms of the both side of the observation equation given in equation 2.4,

$$\log(\mathbf{y}_t^2) = \mathbf{h}_t + \log(\boldsymbol{\varepsilon}_t^2), \quad (2.53)$$

and obtaining the transformed observation equation as

$$\mathbf{z}_t = \mathbf{h}_t + \boldsymbol{\xi}_t, \quad (2.54)$$

where $\mathbf{z}_t = \log(\mathbf{y}_t^2) + 1.2703$ is the transformed observations vector, and $\boldsymbol{\xi}_t = \log(\boldsymbol{\varepsilon}_t^2)$ is the transformed error term of the observation equation. The transformation completes with the assumption that $\boldsymbol{\xi}_t$ is normally distributed with zero mean and covariance matrix, $\boldsymbol{\Sigma}_{\boldsymbol{\xi}\boldsymbol{\xi}} = 4.9339 \times \boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}}$, then the required transformation on the covariance matrices in equation 2.5 can be formally stated as

$$\boldsymbol{\Sigma}_{\boldsymbol{\xi}\boldsymbol{\xi}} = \mathbf{S}\boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}\boldsymbol{\varepsilon}}\mathbf{S}, \quad \boldsymbol{\Sigma}_{\boldsymbol{\eta}\boldsymbol{\xi}} = \boldsymbol{\Sigma}_{\boldsymbol{\eta}\boldsymbol{\varepsilon}}\mathbf{S} \quad \text{and} \quad \boldsymbol{\Sigma}_{\boldsymbol{\xi}\boldsymbol{\eta}} = \mathbf{S}\boldsymbol{\Sigma}_{\boldsymbol{\varepsilon}\boldsymbol{\eta}}. \quad (2.55)$$

where $\mathbf{S} = \text{diag}(2.22126)$ is the diagonal matrix of standard deviations of the transformed random variable $\log(\boldsymbol{\varepsilon}_t^2)$. Here, the mean and variance of $\log(\boldsymbol{\varepsilon}_t^2)$ are known to be -1.27 and $\pi^2/2 = 4.93$ as stated in (Harvey et al., 1994) and can also easily be obtained numerically.

With the above transformation and setting, filtering estimates of states, $\mathbf{h}_{t|t}$ and their covariance $\boldsymbol{\Sigma}_{h,t|t}$ for the MSV-G and MSV-D models can be obtained with the following iterative algorithm:

$$\begin{aligned}
\mathbf{D}_t &= \boldsymbol{\Sigma}_{h,t|t-1} + \boldsymbol{\Sigma}_{\xi\xi,t} \\
\mathbf{h}_{t|t} &= \mathbf{h}_{t|t-1} + \boldsymbol{\Sigma}_{h,t|t-1} \mathbf{D}_t^{-1} (\mathbf{z}_t - \mathbf{h}_{t|t-1}) \\
\boldsymbol{\Sigma}_{h,t|t} &= \boldsymbol{\Sigma}_{h,t|t-1} - \boldsymbol{\Sigma}_{h,t|t-1} \mathbf{D}_t^{-1} \boldsymbol{\Sigma}_{h,t|t-1} \\
\mathbf{K}_t &= (\boldsymbol{\phi} \boldsymbol{\Sigma}_{h,t|t-1} \boldsymbol{\phi}') \mathbf{D}_t^{-1} \\
\mathbf{h}_{t+1|t} &= \boldsymbol{\gamma} + \boldsymbol{\phi} \mathbf{h}_{t|t-1} + \mathbf{K}_t (\mathbf{z}_t - \mathbf{h}_{t|t-1}) \\
\boldsymbol{\Sigma}_{h,t+1|t} &= \boldsymbol{\phi} \boldsymbol{\Sigma}_{h,t|t-1} \boldsymbol{\phi}' + \boldsymbol{\Sigma}_{\eta\eta,t} + \mathbf{K}_t \mathbf{D}_t \mathbf{K}_t'.
\end{aligned} \tag{2.56}$$

L -step predictions based on the Kalman filter algorithm can be obtained by

$$\begin{aligned}
\mathbf{h}_{t+L|t} &= \boldsymbol{\gamma} + \boldsymbol{\phi} \mathbf{h}_{t+L-1|t} \\
\boldsymbol{\Sigma}_{h,t+L|t} &= \boldsymbol{\phi} \boldsymbol{\Sigma}_{h,t+L-1|t} \boldsymbol{\phi}' + \boldsymbol{\Sigma}_{\eta\eta,t},
\end{aligned} \tag{2.57}$$

after the filtering algorithm is executed. Smoothing estimates based on the Kalman filter algorithm is given by

$$\begin{aligned}
\mathbf{D}_t &= \boldsymbol{\Sigma}_{h,t|t-1} + \boldsymbol{\Sigma}_{\xi\xi,t} \\
\mathbf{K}_t &= (\boldsymbol{\phi} \boldsymbol{\Sigma}_{h,t|t-1} \boldsymbol{\phi}' + \boldsymbol{\Sigma}_{\eta\xi,t}) \mathbf{D}_t^{-1} \\
\mathbf{L}_t &= \boldsymbol{\phi} - \mathbf{K}_t \\
\mathbf{u}_{t-1} &= \mathbf{D}_t^{-1} + \mathbf{L}_t' \mathbf{U}_t \mathbf{L}_t \\
\mathbf{U}_{t-1} &= \mathbf{D}_t^{-1} + \boldsymbol{\Sigma}_{h,t|t-1} \mathbf{u}_{t-1} \\
\mathbf{h}_{t|T} &= \mathbf{h}_{t|t-1} + \boldsymbol{\Sigma}_{h,t|t-1} \mathbf{u}_{t-1} \\
\boldsymbol{\Sigma}_{h,t|T} &= \boldsymbol{\Sigma}_{h,t|t-1} - \boldsymbol{\Sigma}_{h,t|t-1} \mathbf{U}_{t-1} \boldsymbol{\Sigma}_{h,t|t-1}.
\end{aligned} \tag{2.58}$$

In the above Kalman filter based algorithms, if the model is static (i.e. MSV-G) then the time subscripts of $\boldsymbol{\Sigma}_{\xi\xi,t}$, $\boldsymbol{\Sigma}_{\eta\xi,t}$, and $\boldsymbol{\Sigma}_{\eta\eta,t}$ are omitted since they stay constant over time.

The log-likelihood function based on the Kalman filter algorithm can be obtained by,

$$\log p(\mathbf{Y}_T) = -\frac{T}{2} \log(2\pi) - \frac{1}{2} \sum_{t=1}^T \log |\mathbf{D}_t| - \frac{1}{2} \sum_{t=1}^T (\mathbf{z}_t - \mathbf{h}_{t|t-1}) \quad (2.59)$$

which can be used in parameter estimation. On the Kalman filter based algorithms given above, extended Kalman filter (EKF) modifications observation and can also be added for better approximations. See further discussions on the usage of Kalman filter based algorithms in (Harvey et al., 1994), (Harvey and Shephard, 1996), and (Tanizaki, 1997) for the SV models.

Although being fast and simple the methods described so far generally suffered from poor performance (Shephard and Andersen 2009), (Watanabe 1999). The mentioned challenges in the estimation problems of MSV models and advances in the computational capabilities quickly incited the usage of computationally intensive Monte Carlo simulation based methods to improve the estimation performance. Several Monte Carlo based algorithms incorporating resampling, particle filters, rejection sampling and importance sampling algorithms have been proposed both in nonlinear state space modeling and SV literature. Monte Carlo simulation based methods were proved to be better methods than the mentioned simplistic approximations or linearizations, however they come with their own disadvantages and limitations regarding error control, convergence, computational complexity, and curse of dimensionality for particular types of algorithms. Examples and detailed discussions of Monte Carlo simulation based methods can be found in (Watanabe, 1999), (Tanizaki, 1997), (Carlin et al., 1992), and (Sandman and Koopman, 1998).

A major breakthrough in SV literature was the introduction of Markov Chain Monte Carlo (MCMC) methods to the econometrics and SV fields by the studies of Tanner and Wong (1987), Tierney (1994), Chib and Greenberg (1995, 1996), and Jacquier et al., 1994). MCMC methods including the influential Metropolis-Hastings and Gibbs sampling algorithms quickly became central to the SV modeling and estimation studies, and an appreciable amount of literature on the applications of different variations of MCMC methods on several types of SV models, particularly for the MSV models was built up. Theoretically, MCMC methods are immune to the curse of dimensionality by construction, and they can easily be implemented in a Bayesian setting where the parameter estimation can also be handled without a maximization routine for the likelihood, hence without an explicit evaluation of the likelihood function as described in (Jacquier et al. 1994). The appealing characteristics of

MCMC methods made them a natural first choice in MSV estimation studies. However, MCMC algorithms are not flawless. They still require intense computational resources for complicated iterative sampling schemes for estimation. They are not exact as being Monte Carlo simulation based methods, and additional issues on error control and convergence are inherent particularly for MCMC methods. Poor mixing chains, correlated samples, identification of convergence, diagnosis, selection of suitable proposal densities are some of the named challenges of MCMC methods.

Multi-dimensional integrals arising in estimation of SV models can be handled by classical numerical integration methods. Being an exact method with a deterministic nature, convergence properties of classical numerical integration methods are superior to simulation methods. However, when the problem dimension increases these methods become computationally infeasible since the number of dimensions increases the complexity of these type of algorithms exponentially. Unsurprisingly, studies on the application of the numerical integration methods to nonlinear state space models and particularly MSV models are quite rare compared to the approximation based methods and Monte Carlo simulation based methods including the MCMC. One of the studies incorporated numerical integration for the univariate case is (Kitagawa, 1987) and a brief discussion on numerical integration can be found in (Tanizaki, 1997). Most of the studies in the MSV estimation literature mention but exclude the numerical integration methods.

Sparse grid integration (SGI) method is a smartly reshaped version of conventional numerical integration method to handle multidimensional integrals by constructing multi-dimensional integration formulas in a way that the dimensionality effect is decreased to a certain extent which allows practical implementation in high dimensional cases in contrast to the conventional numeric integration methods. Sparse grid integration approach starts with work of Smolyak (1963) and detailed treatment of the methods based on the idea is available in (Heiss & Winschel, 2006) and (Bungarts and Griebel, 2004). The sparse grid integration approach was applied to some economics and financial problems (E.g., discrete choice analysis in by Bungarts & Griebel (2004), collateral mortgage optimization problem by Gerstner and Griebel (1998), derivative and option pricing in Gerstner (2007) and asset liability in life insurance in (Holtz, 2010). However, estimation algorithms based on

the sparse grid integration methods for SV models have been neither studied nor mentioned in the literature. In this context, one of the objectives of this study is to fill this gap by applying sparse grid integration method to the estimation algorithms of MSV models.

2.2.3 Estimation with Markov Chain Monte Carlo (MCMC) methods

In this section, implementations of the MCMC methods including the well known Metropolis-Hastings and Gibbs sampling algorithms to the estimation problems of MSV-D model will be presented after a brief background on MCMC methods.

2.2.3.1 Preliminaries on MCMC methods

The idea behind the MCMC methods is to produce variates from a given multivariate density by repeatedly sampling a Markov Chain whose invariant distribution is the target density of interest. Although being a Monte Carlo method, MCMC method is completely different than the classical Monte Carlo techniques because the variates are not generated randomly instead they follow a Markov Chain. To approximate the integral,

$$\int h(x) f(x) dx \quad (2.60)$$

where f is a density function, classical Monte Carlo methods seek ways to obtain direct independent samples from the density f whereas MCMC methods obtain dependent samples using transition kernels through an ergodic Markov chain with stationary distribution f .

The idea of incorporating a Markov chain to sample the target distribution is first proposed by Metropolis et al. (1953) and generalized by Hastings (1970) so the method is known as Metropolis-Hastings algorithm and originates from statistical physics. Later on several variations and extensions are adopted in many fields including the econometrics and SV literature.

Metropolis-Hastings algorithm is one of the main building blocks of MCMC methods. The algorithm starts with the target density f and selection of a conditional density $q(y|x)$ which is called the proposal density. Given the sample X_i , at iteration i , algorithm proceeds as follows:

1. Generate candidate $Y_i \sim q(y|X_i)$,

2. Take,

$$X_{i+1} = \begin{cases} Y_i & \text{with probability } p(X_i, Y_i) \\ X_i & \text{with probability } 1 - p(X_i, Y_i), \end{cases}$$

where, $p(x, y) = \min\{w_{MH}, 1\}$ and w_{MH} is the Metropolis-Hastings ratio (weight) calculated by

$$w_{MH} = \frac{f(y)q(x|y)}{f(x)q(y|x)}. \quad (2.61)$$

In this algorithm, first step generates a candidate sample from the proposal density $q(y|x)$ based on the current sample. The candidate sample is accepted as the new sample if the criteria calculated using the Metropolis-Hastings ratio is met, otherwise current sample is kept as the new sample. Using this algorithm several hard-to-sample densities can be sampled. The proposal density, $q(y|x)$ plays the critical role in the algorithm by determining the structure of the chain and by adjusting the perturbations performed on the current sample. The convergence of algorithm, strictly depends on the choice of the proposal density and its parameters. There is a vast literature on the selection of proposal densities. A good reference on the on Metropolis-Hastings algorithm is (Chib and Greenberg, 1995) and (Chib, 2001).

The Gibbs sampling algorithm is another milestone in the evolution of MCMC methods. Gibbs sampling provides a step by step approach for sampling from multivariate densities using conditional densities of lower dimensions (usually one dimension) than the target density. Starting with the sample i which is a, p -dimensional random variable, $\mathbf{X}^{(i)} = (X_1^{(i)}, \dots, X_p^{(i)})$, each sweep of Gibbs sampling is performed by the sampling step

$$X_j^{(i+1)} \sim f_j(x_j | X_1^{(i)}, X_2^{(i)}, \dots, X_{j-1}^{(i)}, X_{j+1}^{(i)}, \dots, X_p^{(i)}) \quad (2.62)$$

for all $j=1, \dots, p$. One sweep of the sampler is completed in p steps if conditional densities f_j are univariate and after a complete sweep, sample $i+1$, $\mathbf{X}^{(i+1)} = (X_1^{(i+1)}, \dots, X_p^{(i+1)})$, is obtained. In the Gibbs sampling algorithm described above, the full conditional densities f_j are the only densities used for simulation. Thus, even in high dimensional problems such as the MSV models, all of the simulation may be univariate.

2.2.3.2 MCMC based estimation algorithms for the MSV-D

There are two main approaches for applying the MCMC methods to the estimation algorithms developed and discussed in section 2.2.1 for the MSV models.

First approach incorporates Gibbs sampling and Metropolis-Hastings algorithm for obtaining the smoothing densities of the log-volatilities \mathbf{H}_t and the correlation states, \mathbf{Q}_t , first, then uses these densities for filtering, prediction and parameter estimation. In this approach filtering is performed by repeatedly executing smoothing for each time step, prediction is performed by other Monte Carlo methods such as resampling and parameter estimation is performed by a Expectation Maximization (EM) algorithm based on the smoothing densities. This approach is referred as MCMC with EM Algorithm

Second approach uses MCMC methods for obtaining both the smoothing densities of states and parameters at the same time with a Bayesian approach augmenting the parameter space to the state space which eliminates the need for a separate maximization step for parameter estimation and explicit likelihood evaluation. In this approach filtering and prediction is performed in similar to the first approach. This approach is referred as Bayesian MCMC in the study.

In this section algorithms based on the MCMC methods are developed for the MSV-D model for both approaches summarized above.

MCMC with EM algorithm for the MSV-D

This approach incorporates MCMC methods to obtain smoothing estimates of the log-volatilities, \mathbf{h}_t and the correlation states, \mathbf{q}_t by directly sampling from the conditional density $p(\mathbf{H}_T, \mathbf{Q}_T | \mathbf{Y}_T, \boldsymbol{\Omega})$ by sampling $p(\mathbf{Q}_T | \mathbf{H}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$ and $p(\mathbf{H}_T | \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega}_h)$. In this approach, given the current parameter set $\boldsymbol{\Omega} = \{\boldsymbol{\Omega}_h, \boldsymbol{\Omega}_q\}$, one sweep of a Gibbs sampler is performed by the following two main steps:

1. Sample from $p(\mathbf{Q}_T | \mathbf{H}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$,
2. Sample from $p(\mathbf{H}_T | \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega}_h)$.

At each step of the above Gibbs sampler, sampling from $p(\mathbf{Q}_T | \mathbf{H}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$, and $p(\mathbf{H}_T | \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega}_h)$ is performed by other Gibbs samplers sampling from $p(\mathbf{q}_t | \mathbf{H}_t, \mathbf{Q}_{\setminus t}, \mathbf{Y}_t, \boldsymbol{\Omega})$ and $p(\mathbf{h}_t | \mathbf{H}_{\setminus t}, \mathbf{Q}_t, \mathbf{Y}_t, \boldsymbol{\Omega})$ for $t = 1, \dots, T$. And at each time step,

Metropolis-Hastings algorithm is used. N sweeps of the two step Gibbs sampler completes the algorithm and M samples are discarded as burn-in samples and smoothing densities are obtained.

In step 1, to sample from $p(\mathbf{q}_t | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$, the kernel to be sampled at each time period t can be obtained by using the densities constructed in section 2.2.1 and using equation 2.42 as follows

$$\begin{aligned}
p(\mathbf{q}_t | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega}) &= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}_h)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}_h) d\mathbf{q}_t} \\
&= \frac{p_H(\mathbf{H}_T | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q) p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\Omega}_h)}{\int p_H(\mathbf{H}_T | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q) p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\Omega}_h) d\mathbf{q}_t} \quad (2.63) \\
&\propto \begin{cases} p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h), & t < T \\ p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h), & t = T. \end{cases}
\end{aligned}$$

The kernel density obtained in equation 2.63 is not in a simple form that allows direct sampling so Metropolis-Hastings algorithm is applied where the Metropolis-Hasting ratio is given by

$$\begin{aligned}
w_{MH} &= \begin{cases} \frac{p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t^*, \boldsymbol{\Omega}_h) p_q(\mathbf{q}_t^* | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_q(\mathbf{q}_{t+1} | \mathbf{q}_t^*, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t^*, \boldsymbol{\Omega}_h) p_*(\mathbf{q}_t)}{p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_*(\mathbf{q}_t^*)}, & t < T, \\ \frac{p_q(\mathbf{q}_t^* | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h) p_*(\mathbf{q}_t)}{p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h) p_*(\mathbf{q}_t^*)}, & t = T. \end{cases} \quad (2.64)
\end{aligned}$$

In equation 2.64, $p_*(\mathbf{q}_t)$ is the proposal density and \mathbf{q}_t^* is the candidate generated by the proposal density. According to the Metropolis-Hastings algorithm, the candidate is accepted with probability $\min(w_{MH}, 1)$, otherwise previous sample is kept.

One of the options for the proposal density is the density obtained from the state equation of correlation states $p_q(\mathbf{q}_{t+1} | \mathbf{q}_t, \boldsymbol{\Omega}_q)$ given in equation 2.33 with appropriate time indices given by

$$p_*(\mathbf{q}_t) = p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\Omega}_q), \quad (2.65)$$

where a candidate \mathbf{q}_t^* is generated based on \mathbf{q}_{t-1} . Another option is the density obtained from AR(1) missing data problem approach where,

$$\begin{aligned} p_*(\mathbf{q}_t) &\sim N(\boldsymbol{\mu}_p, \mathbf{V}_p) \quad \text{with} \\ \boldsymbol{\mu}_p &= (\mathbf{V}_\omega^{-1} + \boldsymbol{\theta}\mathbf{V}_\omega^{-1}\boldsymbol{\theta})^{-1} (\mathbf{V}_\omega^{-1}(\boldsymbol{\theta}\mathbf{q}_{t-1} + \boldsymbol{\delta}) + \boldsymbol{\theta}\mathbf{V}_\omega^{-1}(\mathbf{q}_{t+1} - \boldsymbol{\delta})), \\ \mathbf{V}_p &= (\mathbf{V}_\omega^{-1} + \boldsymbol{\theta}\mathbf{V}_\omega^{-1}\boldsymbol{\theta})^{-1}. \end{aligned} \quad (2.66)$$

In these densities a scalar c as a coefficient for variance/covariance matrices can be used as a tuning parameter. Several other proposal densities can also be constructed such as random-walk proposal. In the numerical applications of this study, the proposal density obtained with AR(1) missing data problem approach is used.

In step 2, to sample from $p(\mathbf{h}_t | \mathbf{H}_{\setminus T}, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$, the kernel to be sampled at each time period, t , can be obtained by using the densities constructed in section 2.2.1 and using equation 2.42 as follows:

$$\begin{aligned} p(\mathbf{h}_t | \mathbf{H}_{\setminus T}, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega}) &= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}_h)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}_h) d\mathbf{q}_t} \\ &= \frac{p_H(\mathbf{H}_T | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q) p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\Omega}_h)}{\int p_H(\mathbf{H}_T | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q) p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\Omega}_h) d\mathbf{h}_t} \end{aligned} \quad (2.67)$$

$$\propto \begin{cases} p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{q}_{t-1}, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t, \mathbf{h}_{t-1}, \mathbf{q}_{t-1}, \boldsymbol{\Omega}_h), & t < T \\ p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{q}_{t-1}, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t, \mathbf{h}_{t-1}, \mathbf{q}_{t-1}, \boldsymbol{\Omega}_h), & t = T \end{cases}$$

The kernel density obtained in equation 2.67 is also sampled using Metropolis-Hastings algorithm with the Metropolis-Hastings ratio is given by

$$\begin{aligned} w_{MH} &= \\ &= \begin{cases} \frac{p_h(\mathbf{h}_{t+1} | \mathbf{h}_t^*, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_h(\mathbf{h}_t^* | \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t^*, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t^*, \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_*(\mathbf{h}_t)}{p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t, \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_*(\mathbf{h}_t^*)}, & t < T \\ \frac{p_h(\mathbf{h}_t^* | \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t^*, \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_*(\mathbf{h}_t)}{p_h(\mathbf{h}_t | \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_T | \mathbf{h}_T, \mathbf{q}_T, \boldsymbol{\Omega}_h) p_y(\mathbf{y}_{t-1} | \mathbf{h}_t, \mathbf{h}_{t-1}, \mathbf{q}_t, \boldsymbol{\Omega}_h) p_*(\mathbf{h}_t^*)}, & t = T, \end{cases} \end{aligned} \quad (2.68)$$

where $p_*(\mathbf{h}_t)$ is the proposal density and \mathbf{h}_t^* is the candidate generated by the proposal density. According to the Metropolis-Hastings algorithm, the candidate is accepted with probability $\min(w_{MH}, 1)$, otherwise previous sample is kept.

One of the good options for the proposal density $p_*(\mathbf{h}_t)$ is the density obtained from the Kalman filter smoothing algorithm given in equation 2.58 as

$$p_*(\mathbf{h}_t) \sim N(\boldsymbol{\mu}_{h,t|T}, c\boldsymbol{\Sigma}_{h,t|T}) \quad (2.69)$$

where $\boldsymbol{\mu}_{h,t|T}$ is the mean and $\boldsymbol{\Sigma}_{h,t|T}$ is the covariance matrix obtained from the Kalman smoother algorithm given in equation 2.58 and c is a scalar tuning parameter. Other proposal densities can also be used such as the density from log-volatility equation or density from AR(1) missing data problem. In the applications of this study, the density from the Kalman filter is used.

Having the joint conditional smoothing density $p(\mathbf{H}_T, \mathbf{Q}_T | \mathbf{Y}_T, \boldsymbol{\Omega})$ with the above MCMC algorithm, the parameter estimation is performed by maximizing the expected log-likelihood,

$$\begin{aligned} & E_{\mathbf{H}, \mathbf{Q}, \mathbf{Y}, \boldsymbol{\Omega}} \left[\log \left(p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}) \right) \right] = \\ & = \int \log \left(p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T | \boldsymbol{\Omega}) \right) p(\mathbf{H}_T, \mathbf{Q}_T | \mathbf{Y}_T, \boldsymbol{\Omega}) d\mathbf{H}_T d\mathbf{Q}_T \\ & = \int \log \left(p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q) p_H(\mathbf{H}_T | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Y(\mathbf{Y}_T | \mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\Omega}_h) \right) \\ & \quad p(\mathbf{H}_T, \mathbf{Q}_T | \mathbf{Y}_T, \boldsymbol{\Omega}) d\mathbf{H}_T d\mathbf{Q}_T \quad (2.70) \\ & = \frac{1}{N-M} \sum_{i=M+1}^N \log \left(p_Q(\mathbf{Q}_T^{(i)} | \boldsymbol{\Omega}_q) p_H(\mathbf{H}_T^{(i)} | \mathbf{Q}_T, \boldsymbol{\Omega}_h) p_Y(\mathbf{Y}_T | \mathbf{H}_T^{(i)}, \mathbf{Q}_T^{(i)}, \boldsymbol{\Omega}_h) \right) \end{aligned}$$

with respect to the parameter set $\boldsymbol{\Omega}$ in an EM algorithm where the smoothing density $p(\mathbf{H}_T, \mathbf{Q}_T | \mathbf{Y}_T, \boldsymbol{\Omega})$ is repeatedly executed with the current parameter set obtained by the maximization of the expectation in equation 2.70 until a particular convergence criteria is met. As an optimization routine several alternatives are available, one of the general purpose nonlinear optimization method can be used. Well known Newton's method, derivative free search algorithms such as Nelder-Mead, or quasi-Newton methods such as Broyden-Fletcher-Goldfarb-Shanno (BFGS) are some of the examples. Using derivative free optimization methods can significantly improve computational performance however methods that provide the exact or an approximated Hessian at the likelihood readily provides the Fisher information matrix which directly gives the standard errors of parameter estimates. In the numerical applications in this study Newton's method and Nelder-Mead algorithms are used.

In this approach, filtering densities and estimates are obtained by simply using the exact same procedure described above for smoothing with only change in the time indices, for each time step $t = 1, \dots, T$ the smoothing procedure is repeated and filtering estimates are obtained. Obviously, filtering with MCMC method is computationally much more demanding than the smoothing.

Prediction estimates can be obtained recursively using resampling (with N samples) for approximating the prediction algorithm given equation 2.45 as

$$\begin{aligned} p(\mathbf{h}_{t+L}, \mathbf{q}_{t+L} | \mathbf{Y}_t, \boldsymbol{\Omega}) &\approx \\ &\approx \sum_{j=1}^N p_h(\mathbf{h}_{t+L} | \mathbf{h}_{t+L-1}^{(j)}, \mathbf{q}_{t+L-1}^{(j)}, \boldsymbol{\Omega}_h) p_q(\mathbf{q}_{t+L} | \mathbf{q}_{t+L-1}^{(j)}, \boldsymbol{\Omega}_q) p(\mathbf{h}_{t+L-1}^{(j)}, \mathbf{q}_{t+L-1}^{(j)} | \mathbf{Y}_t, \boldsymbol{\Omega}), \end{aligned} \quad (2.71)$$

based on the filtering density $p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_t, \boldsymbol{\Omega})$.

Bayesian MCMC for the MSV-D

Bayesian approach for the estimation of MSV models is probably the most studied topic in SV literature with examples in (Jacquier et al. 1994), (Gourieroux et al. 2009), (Asai and McAleer, 2009), (Ishihara, 2012), (Ishihara et al., 2014).

In the Bayesian approach the parameters of the MSV-D model, $\boldsymbol{\Omega} = \{\boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega\}$ are considered to be random variables with prior distributions and their space is augmented to the state space and MCMC methods are used to sample from the joint distribution of states and parameters given the observations, $p(\mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega | \mathbf{Y}_T)$. In this approach there is no need for a separate log-likelihood maximization routine since parameter estimates are also obtained through the implemented sampling scheme. This also means that there is no need for explicit calculation of the log-likelihood function or its expectation. However, appropriate prior distributions for the parameters and efficient sampling mechanisms from their posterior distributions must be developed in this approach. Thus models with different specification usually have different implementations. In this section a customized Bayesian MCMC algorithm is developed for the MSV-D.

The MCMC based algorithm to sample from $p(\mathbf{H}_T, \mathbf{Q}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega | \mathbf{Y}_T)$ is performed with the following Gibbs sampler at each step sampling from a different posterior density:

1. Sample from $p(\boldsymbol{\delta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\theta}, \mathbf{V}_\omega)$,

2. Sample from $p(\boldsymbol{\theta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \mathbf{V}_\omega)$,
3. Sample from $p(\mathbf{V}_\omega | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta})$,
4. Sample from $p(\mathbf{Q}_T | \mathbf{H}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$,
5. Sample from $p(\gamma | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$,
6. Sample from $p(\boldsymbol{\varphi} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$,
7. Sample from $p(\mathbf{V}_\eta | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$,
8. Sample from $p(\mathbf{H}_T | \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$.

Here, the densities at steps 1, 2, 3, 5, 6 and 7 are posterior densities of the parameters, and the densities at steps 4 and 8 are posterior densities of the states. Here the eight step Gibbs sampling sweep is repeated N times and joint density of the states and parameters are obtained at once. At each step, the conditional posterior density of interest is multi-dimensional and some of them do not allow direct sampling so Metropolis-Hastings algorithm is used in such cases.

At step 1 of the above Gibbs sampler, the posterior is,

$$\begin{aligned}
p(\boldsymbol{\delta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\theta}, \mathbf{V}_\omega) &= \\
&= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\boldsymbol{\delta}} \\
&\quad \left(p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta) \right) \\
&= \frac{p(\gamma) p(\boldsymbol{\varphi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega)}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\varphi}, \mathbf{V}_\eta) \\
&\quad p(\gamma) p(\boldsymbol{\varphi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\boldsymbol{\delta}} \quad (2.72) \\
&= \frac{p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\delta})}{\int p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\delta}) d\boldsymbol{\delta}} \\
&\propto p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\delta}).
\end{aligned}$$

If a diffuse prior for $\boldsymbol{\delta}$ (i.e. $p(\boldsymbol{\delta}) \propto 1$) is assumed then the posterior obtained in equation 2.72 becomes

$$\begin{aligned}
& p(\boldsymbol{\delta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
& \propto p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
& = p_q(\mathbf{q}_1) \prod_{t=1}^{T-1} p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
& = (2\pi)^{-rT/2} |\mathbf{V}_q|^{-\frac{1}{2}} |\mathbf{V}_\omega|^{-\frac{T-1}{2}} \exp \left(-\frac{1}{2} (\mathbf{q}_1 - \boldsymbol{\mu}_q)' \mathbf{V}_q^{-1} (\mathbf{q}_1 - \boldsymbol{\mu}_q) - \sum_{t=1}^{T-1} \frac{1}{2} (\mathbf{q}_{t+1} - \boldsymbol{\theta} \mathbf{q}_t - \boldsymbol{\delta})' \mathbf{V}_\omega^{-1} (\mathbf{q}_{t+1} - \boldsymbol{\theta} \mathbf{q}_t - \boldsymbol{\delta}) \right) \\
& \propto f_N(\boldsymbol{\delta}; \boldsymbol{\mu}_\delta, \boldsymbol{\Sigma}_\delta).
\end{aligned} \tag{2.73}$$

which is a normal distribution with the mean and covariance given by

$$\begin{aligned}
\boldsymbol{\mu}_\delta &= \frac{1}{T-1} \sum_{t=1}^{T-1} (\mathbf{q}_{t+1} - \boldsymbol{\theta} \mathbf{q}_t), \\
\boldsymbol{\Sigma}_\delta &= \frac{1}{T-1} \mathbf{V}_\omega.
\end{aligned} \tag{2.74}$$

Thus, sampling from $p(\boldsymbol{\delta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\theta}, \mathbf{V}_\omega)$ can be directly performed from a normal distribution with the parameters given in equation 2.74.

At step 2 of the above Gibbs sampler, the posterior is,

$$\begin{aligned}
& p(\boldsymbol{\theta} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \mathbf{V}_\omega) = \\
& = \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\boldsymbol{\theta}} \\
& \quad \left(p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right) \\
& = \frac{p(\boldsymbol{\gamma}) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega)}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\gamma}) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\boldsymbol{\theta}} \tag{2.75} \\
& = \frac{p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\theta})}{\int p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\theta}) d\boldsymbol{\theta}} \\
& \propto p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\boldsymbol{\theta}).
\end{aligned}$$

A beta prior for each θ_i can be used for $i = 1, \dots, r$. Then, the prior distribution of $\boldsymbol{\theta}$ becomes,

$$p(\boldsymbol{\theta}) = \prod_{i=1}^r f_B\left(\frac{\theta_i + 1}{2}; \alpha_i, \beta_i\right), \quad (2.76)$$

With this prior density, the posterior in equation 2.75 is not in a simple form allowing direct sampling, thus it can be sampled by Metropolis-Hastings algorithm with the ratio,

$$w_{MH} = \frac{p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \mathbf{V}_\omega, \boldsymbol{\theta}^*) p(\boldsymbol{\theta}^*) p_*(\boldsymbol{\theta})}{p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \mathbf{V}_\omega, \boldsymbol{\theta}) p(\boldsymbol{\theta}) p_*(\boldsymbol{\theta}^*)}, \quad (2.77)$$

where an appropriate proposal density, $p_*(\boldsymbol{\theta})$, can be obtained from $p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \mathbf{V}_\omega, \boldsymbol{\theta})$, by letting $\mathbf{D}_{q,t} = \text{diag}(\mathbf{q}_{1t}, \dots, \mathbf{q}_{rt})$ and $\boldsymbol{\theta}_v$ be the vector composed of the diagonal entries of the diagonal matrix $\boldsymbol{\theta}$, as follows:

$$\begin{aligned} & p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \mathbf{V}_\omega, \boldsymbol{\theta}) \\ &= p_q(\mathbf{q}_1) \prod_{t=1}^{T-1} p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\ &= (2\pi)^{-rT/2} |\mathbf{V}_q|^{-\frac{1}{2}} |\mathbf{V}_\omega|^{-\frac{T-1}{2}} \\ & \exp \left(-\frac{1}{2} (\mathbf{q}_1 - \boldsymbol{\mu}_q)' \mathbf{V}_q^{-1} (\mathbf{q}_1 - \boldsymbol{\mu}_q) - \right. \\ & \left. \sum_{t=1}^{T-1} \frac{1}{2} (\mathbf{D}_{q,t}^{-1} \mathbf{q}_{t+1} - \boldsymbol{\theta}_v - \mathbf{D}_{q,t}^{-1} \boldsymbol{\delta})' \mathbf{D}_{q,t} \mathbf{V}_\omega^{-1} \mathbf{D}_{q,t} (\mathbf{D}_{q,t}^{-1} \mathbf{q}_{t+1} - \boldsymbol{\theta}_v - \mathbf{D}_{q,t}^{-1} \boldsymbol{\delta}) \right) \\ & \propto f_N(\boldsymbol{\theta}_v; \boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta). \end{aligned} \quad (2.78)$$

The density obtained in equation 2.78 is a normal distribution with covariance and mean given by

$$\begin{aligned} \boldsymbol{\Sigma}_\theta &= (\mathbf{V}_\omega^{-1} \odot \mathbf{A})^{-1}, \\ \boldsymbol{\mu}_\theta &= (\mathbf{V}_\omega^{-1} \odot \mathbf{A})^{-1} \mathbf{b}, \\ \mathbf{A} &= \sum_{t=1}^T \mathbf{q}_t \mathbf{q}_t', \quad \mathbf{b} = \text{diag} \left(\sum_{t=1}^{T-1} \mathbf{q}_t (\mathbf{q}_{t+1} - \boldsymbol{\delta})' \mathbf{V}_\omega^{-1} \right), \end{aligned} \quad (2.79)$$

where \odot is the Hadamard product. Then a suitable proposal density $p_*(\boldsymbol{\theta})$ is given by the following truncated normal density with a truncation interval $[0, 1]$:

$$p_*(\boldsymbol{\theta}) = f_{TN}(\boldsymbol{\theta}_v; \boldsymbol{\mu}_\theta, \boldsymbol{\Sigma}_\theta, 0, 1). \quad (2.80)$$

At step 3 of the above Gibbs sampler, the posterior is,

$$\begin{aligned}
& p(\mathbf{V}_\omega | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}) \\
&= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\mathbf{V}_\omega} \\
&= \frac{(p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega))}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\mathbf{V}_\omega} \quad (2.81) \\
&= \frac{p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\mathbf{V}_\omega)}{\int p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\mathbf{V}_\omega) d\mathbf{V}_\omega} \\
&\propto p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \mathbf{V}_\omega, \boldsymbol{\theta}) p(\mathbf{V}_\omega).
\end{aligned}$$

and if an inverse gamma prior for each $\sigma_{\omega,i}$ is assumed, then the prior distribution of \mathbf{V}_ω becomes,

$$p(\mathbf{V}_\omega) = \prod_{i=1}^r f_{IG}(\sigma_{\omega,i}^2; \alpha_{0i}, \beta_{0i}). \quad (2.82)$$

With this prior, posterior density in equation 2.81 becomes

$$\begin{aligned}
& p(\mathbf{V}_\omega | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}) \\
&\propto p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p(\mathbf{V}_\omega) \\
&= p_q(\mathbf{q}_1) \prod_{t=1}^{T-1} p_q(\mathbf{q}_t | \mathbf{q}_{t-1}, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
&= (2\pi)^{-\frac{rT}{2}} |\mathbf{V}_q|^{\frac{1}{2}} |\mathbf{V}_\omega|^{-\frac{T-1}{2}} \exp \left(-\frac{1}{2} (\mathbf{q}_1 - \boldsymbol{\mu}_q)' \mathbf{V}_q^{-1} (\mathbf{q}_1 - \boldsymbol{\mu}_q) - \sum_{t=1}^{T-1} \frac{1}{2} (\mathbf{q}_{t+1} - \boldsymbol{\theta} \mathbf{q}_t - \boldsymbol{\delta})' \mathbf{V}_\omega^{-1} (\mathbf{q}_{t+1} - \boldsymbol{\theta} \mathbf{q}_t - \boldsymbol{\delta}) \right) \\
&\quad \prod_{i=1}^r \frac{\beta_{0i}^{\alpha_{0i}}}{\Gamma(\alpha_{0i})} \sigma_{\omega,i}^{-2(\alpha_{0i}+1)} \exp \left(-\frac{\beta_{0i}}{\sigma_{\omega,i}^2} \right) \\
&\propto \prod_{i=1}^r \sigma_{\omega,i}^{-2\left(\alpha_{0i} + \frac{T-1}{2} + 1\right)} \exp \left(-\frac{\beta_{0i} + \sum_{t=1}^{T-1} (q_{i,t+1} - \theta_i q_{i,t} - \delta_i)^2}{\sigma_{\omega,i}^2} \right) \quad (2.83) \\
&\propto \prod_{i=1}^r f_{IG}(\sigma_{\omega,i}^2; \alpha_{li}, \beta_{li}).
\end{aligned}$$

The density obtained in equation 2.83 is also an inverse gamma density with parameters

$$\begin{aligned}\alpha_{li} &= -\left(\alpha_{0i} + \frac{T-1}{2} + 1\right), \quad i=1, \dots, r, \\ \beta_{li} &= \beta_i + \frac{1}{2} \sum_{t=1}^{T-1} (q_{i,t+1} - \theta_i q_{i,t} - \delta_i)^2, \quad i=1, \dots, r.\end{aligned}\tag{2.84}$$

Thus, sampling from $p(\mathbf{V}_\omega | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta})$ can be directly performed from an inverse gamma distribution with the parameters given equation 2.84.

At step 4, sampling from $p(\mathbf{Q}_T | \mathbf{H}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$ is performed by sampling from the kernel density $p(\mathbf{q}_t | \mathbf{H}_T, \mathbf{Q}_{\setminus t}, \mathbf{Y}_T, \boldsymbol{\Omega})$ as described in the previous section using the Metropolis-Hastings algorithm with the Metropolis-Hastings ratio given in equation 2.64 and the proposal density given in equation 2.66.

At step 5 of the above Gibbs sampler, the posterior is,

$$\begin{aligned}p(\gamma | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) &= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\gamma} \\ &= \frac{(p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega))}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\gamma} \\ &= \frac{p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma)}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) d\gamma} \\ &\propto p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma).\end{aligned}\tag{2.85}$$

If a diffuse prior for γ (i.e. $p(\gamma) \propto 1$) is assumed then the posterior obtained in equation 2.85 becomes

$$\begin{aligned}
& p(\boldsymbol{\gamma} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
& \propto p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\gamma}) \\
& = p_h(\mathbf{h}_1) \left(\prod_{t=1}^{T-1} p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right) \left(\prod_{t=1}^T p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right) \\
& \propto \exp \left(-\frac{1}{2} \sum_{t=1}^{T-1} \left((\boldsymbol{\gamma} - (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t))' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\boldsymbol{\gamma} - (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t)) + \right. \right. \\
& \quad \left. \left. (\boldsymbol{\gamma} - (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t - \boldsymbol{\mu}_{y|\eta}))' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\boldsymbol{\gamma} - (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t - \boldsymbol{\mu}_{y|\eta})) \right) \right) \\
& \propto f_N(\boldsymbol{\gamma}; \boldsymbol{\mu}_\gamma, \boldsymbol{\Sigma}_\gamma).
\end{aligned} \tag{2.86}$$

which is a normal distribution with covariance and mean given by

$$\begin{aligned}
\boldsymbol{\Sigma}_\gamma &= \sum_{t=1}^T \left(\boldsymbol{\Sigma}_{\eta\eta,t}^{-1} + \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \right)^{-1} \\
\boldsymbol{\mu}_\gamma &= \boldsymbol{\Sigma}_\gamma \sum_{t=1}^T \left(\boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t) + \right. \\
& \quad \left. \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t - \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \mathbf{y}_t) \right).
\end{aligned} \tag{2.87}$$

Thus, sampling from $p(\boldsymbol{\gamma} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$ can be directly performed from a normal distribution with the parameters given in equation 2.87.

At step 6 of the above Gibbs sampler, the posterior is,

$$\begin{aligned}
& p(\boldsymbol{\phi} | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\
& = \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\boldsymbol{\phi}} \\
& \quad \left(p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right. \\
& \quad \left. \frac{p(\boldsymbol{\gamma}) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega)}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right. \\
& \quad \left. p(\boldsymbol{\gamma}) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\boldsymbol{\phi} \right) \\
& = \frac{p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\phi})}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\phi}) d\boldsymbol{\phi}} \\
& \propto p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\phi})
\end{aligned} \tag{2.88}$$

If a beta prior for each ϕ_i , $i = 1, \dots, p$, the prior distribution of $\boldsymbol{\phi}$ becomes,

$$p(\boldsymbol{\varphi}) = \prod_{i=1}^p f_B\left(\frac{\varphi_i + 1}{2}; \alpha_i, \beta_i\right). \quad (2.89)$$

With this prior distribution, the posterior in equation 2.88 can be sampled by Metropolis-Hastings algorithm with the ratio,

$$w_{MH} = \frac{p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}^*, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}^*, \mathbf{V}_\eta) p(\boldsymbol{\varphi}^*) p_*(\boldsymbol{\varphi})}{p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) p(\boldsymbol{\varphi}) p_*(\boldsymbol{\varphi}^*)} \quad (2.90)$$

By letting $\mathbf{D}_{h,t} = \text{diag}(\mathbf{h}_{1t}, \dots, \mathbf{h}_{pt})$ and $\boldsymbol{\varphi}_v$ be the vector composed of the diagonal entries of the diagonal matrix $\boldsymbol{\varphi}$, an appropriate proposal density, $p_*(\boldsymbol{\varphi})$, can be obtained from the conditional $p(\mathbf{Y}_t, \mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta)$ as follows:

$$\begin{aligned} & p(\mathbf{Y}_t, \mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) \\ &= p_H(\mathbf{H}_t | \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) \\ &= p_h(\mathbf{h}_1) \left(\prod_{t=1}^{T-1} p_h(\mathbf{h}_{t+1} | \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) \right) \left(\prod_{t=1}^T p_y(\mathbf{y}_t | \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta) \right) \\ &\propto \exp\left(-\frac{1}{2} \sum_{t=1}^{T-1} (\boldsymbol{\varphi}_v - \mathbf{D}_{h,t}^{-1}(\mathbf{h}_{t+1} - \boldsymbol{\gamma}))' \mathbf{D}_{h,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \mathbf{D}_{h,t} (\boldsymbol{\varphi}_v - \mathbf{D}_{h,t}^{-1}(\mathbf{h}_{t+1} - \boldsymbol{\gamma})) \right. \\ &\quad \left. + (\boldsymbol{\varphi}_v - \mathbf{D}_{h,t}^{-1}(\mathbf{h}_{t+1} - \boldsymbol{\gamma} - \boldsymbol{\Sigma}_{\eta\eta,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \mathbf{y}_t))' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{\eta y,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \right. \\ &\quad \left. (\boldsymbol{\varphi}_v - \mathbf{D}_{h,t}^{-1}(\mathbf{h}_{t+1} - \boldsymbol{\gamma} - \boldsymbol{\Sigma}_{\eta\eta,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \mathbf{y}_t)) \right) \\ &\propto f_N(\boldsymbol{\varphi}_v; \boldsymbol{\mu}_\varphi, \boldsymbol{\Sigma}_\varphi). \end{aligned} \quad (2.91)$$

The density obtained in equation 2.91 is a normal density with the covariance and mean given by

$$\boldsymbol{\Sigma}_\varphi = \left(\sum_{t=1}^T (\boldsymbol{\Sigma}_{\eta\eta,t}^{-1} + \boldsymbol{\Sigma}_{\eta\eta,t} + \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{\eta y,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1}) \odot \mathbf{A}_t \right)^{-1}, \quad \boldsymbol{\mu}_\varphi = \boldsymbol{\Sigma}_\varphi \sum_{t=1}^T \mathbf{b}_t, \quad (2.92)$$

where,

$$\begin{aligned} \mathbf{A}_t &= \mathbf{h}_t \mathbf{h}_t', \\ \mathbf{B}_t &= \mathbf{h}_t (\mathbf{h}_{t+1} - \boldsymbol{\gamma})' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} + \\ &\quad \mathbf{h}_t (\mathbf{h}_{t+1} - \boldsymbol{\gamma} - \boldsymbol{\Sigma}_{\eta\eta,t} \boldsymbol{\Sigma}_{y\eta,t}^{-1} \mathbf{y}_t)' \boldsymbol{\Sigma}_{\eta\eta,t}^{-1} \boldsymbol{\Sigma}_{\eta y,t} \boldsymbol{\Sigma}_{\eta y,t}^{-1} \boldsymbol{\Sigma}_{y\eta,t} \boldsymbol{\Sigma}_{\eta\eta,t}^{-1}, \\ \mathbf{b}_t &= \text{diagonal}(\mathbf{B}_t). \end{aligned} \quad (2.93)$$

Then, a suitable proposal density, $p_*(\boldsymbol{\phi})$ is given by the following truncated normal density with a truncation interval $[0, 1]$:

$$p_*(\boldsymbol{\phi}) = f_{TN}(\boldsymbol{\phi}_v; \boldsymbol{\mu}_\phi, \boldsymbol{\Sigma}_\phi, 0, 1), \quad (2.94)$$

At step 7 of the above Gibbs sampler, the posterior is,

$$\begin{aligned} & p(\mathbf{V}_\eta | \mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) \\ &= \frac{p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)}{\int p(\mathbf{H}_T, \mathbf{Q}_T, \mathbf{Y}_T, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) d\mathbf{V}_\eta} \\ &= \frac{(p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega))}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Q(\mathbf{Q}_T | \boldsymbol{\Omega}_q, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\gamma) p(\boldsymbol{\phi}) p(\mathbf{V}_\eta) p(\boldsymbol{\delta}) p(\boldsymbol{\theta}) p(\mathbf{V}_\omega) d\mathbf{V}_\eta} \\ &= \frac{p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\mathbf{V}_\eta)}{\int p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\mathbf{V}_\eta) d\mathbf{V}_\eta} \\ &\propto p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\mathbf{V}_\eta). \end{aligned} \quad (2.95)$$

If an inverse gamma prior for each $\sigma_{\eta,i}$ is assumed, then the prior distribution of \mathbf{V}_η becomes

$$p(\mathbf{V}_\eta) = \prod_{i=1}^p f_{IG}(\sigma_{\eta,i}^2; \alpha_{0i}, \beta_{0i}) \quad (2.96)$$

With this prior distribution, the posterior in equation 2.95 can be sampled by Metropolis-Hastings algorithm with the ratio,

$$w_{MH} = \frac{p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta^*) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta^*) p(\mathbf{V}_\eta^*) p_*(\mathbf{V}_\eta)}{p_H(\mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t | \mathbf{H}_t, \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta) p(\boldsymbol{\phi}) p_*(\mathbf{V}_\eta^*)} \quad (2.97)$$

Letting $\mathbf{D}_{\eta,t} = \text{diag}(\eta_{1,t}, \dots, \eta_{p,t})$ and $\mathbf{s}_\eta = (1/\sigma_{\eta,1}, \dots, 1/\sigma_{\eta,p})'$ then a suitable proposal density, $p_*(\mathbf{V}_\eta)$ can be constructed from the conditional $p(\mathbf{Y}_t, \mathbf{H}_t | \mathbf{Q}_t, \gamma, \boldsymbol{\phi}, \mathbf{V}_\eta)$ as follows:

$$\begin{aligned}
& p(\mathbf{Y}_t, \mathbf{H}_t \mid \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \\
&= p_H(\mathbf{H}_t \mid \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) p_Y(\mathbf{Y}_t \mid \mathbf{H}_t, \mathbf{Q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \\
&= p_h(\mathbf{h}_1) \left(\prod_{t=1}^{T-1} p_h(\mathbf{h}_{t+1} \mid \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right) \left(\prod_{t=1}^T p_y(\mathbf{y}_t \mid \mathbf{h}_{t+1}, \mathbf{h}_t, \mathbf{q}_t, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta) \right) \\
&\propto \prod_{i=1}^p \sigma_{\eta,i}^{-2\left(\alpha_{0i} + \frac{T-1}{2} + 1\right)} \\
&\quad \exp\left(-\sum_{i=1}^p \frac{\beta_{0i}}{\sigma_{\eta,i}^2} - \frac{1}{2} \mathbf{s}'_\eta \left(\sum_{t=1}^{T-1} \mathbf{D}_{\eta,t} \left(\mathbf{P}_{\eta\eta,t}^{-1} + \mathbf{P}_{\eta\eta,t}^{-1} \mathbf{P}_{\eta y,t} \mathbf{V}_{y,t}^{1/2} \boldsymbol{\Sigma}_{y|\eta,t}^{-1} \mathbf{V}_{y,t}^{1/2} \mathbf{P}_{y\eta,t} \mathbf{P}_{\eta\eta,t}^{-1} \right) \mathbf{D}_{\eta,t} \right) \mathbf{s}_\eta \right) \\
&\quad \exp\left(\sum_{t=1}^{T-1} \mathbf{s}'_\eta \mathbf{D}_{\eta,t} \mathbf{P}_{\eta\eta,t}^{-1} \mathbf{P}_{\eta y,t} \mathbf{V}_{y,t}^{1/2} \boldsymbol{\Sigma}_{y|\eta,t}^{-1} \mathbf{y}_t + \mathbf{y}'_t \boldsymbol{\Sigma}_{y|\eta,t}^{-1} \mathbf{V}_{y,t}^{1/2} \mathbf{P}_{y\eta,t} \mathbf{P}_{\eta\eta,t}^{-1} \mathbf{D}_{\eta,t} \mathbf{s}_\eta \right) \\
&\quad \exp\left(\sum_{t=1}^{T-1} \mathbf{y}'_t \boldsymbol{\Sigma}_{y|\eta,t}^{-1} \mathbf{y}_t \right) \\
&= \prod_{i=1}^p \sigma_{\eta,i}^{-2\left(\alpha_{0i} + \frac{T-1}{2} + 1\right)} \exp\left(-\sum_{i=1}^p \frac{\mathbf{A}_{ii}}{\sigma_{\eta,i}^2}\right) f(\sigma_{\eta,1}, \dots, \sigma_{\eta,p}) \\
&= \left(\prod_{i=1}^p f_{IG}(\sigma_{\eta,i}^2; \alpha_{1i}, \beta_{1i}) \right) f(\sigma_{\eta,1}, \dots, \sigma_{\eta,p}).
\end{aligned} \tag{2.98}$$

where,

$$\begin{aligned}
\mathbf{A} &= \sum_{t=1}^{T-1} \left(\left(\mathbf{P}_{\eta\eta,t}^{-1} + \mathbf{P}_{\eta\eta,t}^{-1} \mathbf{P}_{\eta y,t} \mathbf{V}_{y,t}^{1/2} \boldsymbol{\Sigma}_{y|\eta,t}^{-1} \mathbf{V}_{y,t}^{1/2} \mathbf{P}_{y\eta,t} \mathbf{P}_{\eta\eta,t}^{-1} \right) \odot \mathbf{B}_t \right), \\
\mathbf{B}_t &= (\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t - \boldsymbol{\gamma})(\mathbf{h}_{t+1} - \boldsymbol{\phi} \mathbf{h}_t - \boldsymbol{\gamma})'.
\end{aligned} \tag{2.99}$$

Then, the first part of the posterior in equation 2.98 which is the multiplication of inverse gamma densities can be used as a proposal density $p_*(\mathbf{V}_\eta)$ where the parameters are given by

$$\begin{aligned}
\alpha_{1i} &= -\left(\alpha_{0i} + \frac{T-1}{2} + 1 \right) \\
\beta_{1i} &= \beta_{0i} + \frac{1}{2} \sum_{t=1}^{T-1} \mathbf{A}_{ii}
\end{aligned} \tag{2.100}$$

At step 8 of the Gibbs sampler, sampling from $p(\mathbf{H}_T \mid \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega)$ is performed by sampling from the kernel density $p(\mathbf{h}_t \mid \mathbf{H}_{\setminus T}, \mathbf{Q}_T, \mathbf{Y}_T, \boldsymbol{\Omega})$ as described in the previous section using the Metropolis-Hastings algorithm with the Metropolis-Hastings ratio given in equation 2.68 and the proposal density given in equation 2.69.

In both MCMC based estimation approaches for the MSV-D model, if dynamic components and associated terms with the correlation states, \mathbf{q}_t , are dropped along with the parameter set $\mathbf{\Omega}_q = \{\boldsymbol{\delta}, \boldsymbol{\theta}, \mathbf{V}_\omega\}$, then the only parameter set $\mathbf{\Omega}_h$ becomes $\mathbf{\Omega}_h = \{\boldsymbol{\gamma}, \boldsymbol{\phi}, \mathbf{V}_\eta, \boldsymbol{\delta}\}$ where $\boldsymbol{\delta}$ is the constant parameter for the correlations between asset returns in place of dynamic correlation states \mathbf{q}_t . Then the simplified versions of the algorithms can be used for the MSV-G or MSV-B model. For example the two steps Gibbs sampler for the MCMC with EM algorithm, reduces to a single step and eight-steps Gibbs sampler for the Bayesian MCMC, reduces to five steps and posteriors stays almost same with simplifications.

2.2.4 Estimation with sparse grid integration (SGI) method

In this section, development of the estimation algorithms based on the sparse grid integration (SGI) method for the MSV-D model will be presented after a brief background on the method.

2.2.4.1 Preliminaries on the SGI method

Similar to classical numerical integration methods, sparse grid integration methods are also based on integration formulas which are simply represented by a set of function evaluation points and corresponding weights. These points and weights are then used to evaluate the integral of a given function with,

$$I^{(d)} f = \int_{\Omega} f(\mathbf{x}) d\mathbf{x} \approx \sum_{i=1}^{N_l} w_{il} f(\mathbf{x}_{il}), \quad (2.101)$$

where, d is the dimension, l is the level, x_{il} are the p -dimensional vectors representing points, w_{il} are the weights and N_l is the number of points in the integration formula represented by

$$Q_l^{(d)} f = \sum_{i=1}^{N_l} w_{il} f(\mathbf{x}_{il}). \quad (2.102)$$

In equation 2.101 and equation 2.102, level $l = 1, 2, ..$ determines the number of points, N_l , in the formula. The relationship between the level l and the number of points, N_l , depends on the type of the formula. For the well known univariate iterated trapezoid rule for open interval given by

$$Q_l^{(1)} f = \frac{1}{N_l + 1} \left(\frac{3}{2} f \left(\frac{1}{N_l + 1} \right) + \sum_{i=2}^{N_l-1} f \left(\frac{i}{N_l + 1} \right) + \frac{3}{2} f \left(\frac{N_l}{N_l + 1} \right) \right). \quad (2.103)$$

The relationship between the number of points N_l and the level l can be cast as

$$N_l = 2^l - 1. \quad (2.104)$$

The trapezoid rule given in equation 2.103 is a nested rule where the points in a given level are kept in the upper levels with addition of extra points at each increment in level.

In the conventional numerical integration approach multivariate integration formulas such as the one in equation 2.102 are constructed from univariate rules with the tensor product,

$$\left(Q_{i_1}^{(1)} \otimes \dots \otimes Q_{i_d}^{(1)} \right) f = \sum_{i_1=1}^{N_{i_1}} \dots \sum_{i_d=1}^{N_{i_d}} w_{i_1} \dots w_{i_d} f \left(x_{i_1}, \dots, x_{i_d} \right). \quad (2.105)$$

Thus, the integrand is evaluated at the points of a product grid where the resulting multidimensional weights are the products of the corresponding one dimensional weights. Classical product quadrature methods with this approach achieve an accuracy of

$$\varepsilon(N) = \mathcal{O}(N^{-r/d}) \quad (2.106)$$

for the computation of multivariate integrals with N evaluations of the integrand at each dimension of the grid boundary for functions with bounded mixed derivatives up to order r . In conventional numerical integration approach as d increase the convergence deteriorates rapidly and computational burden increases exponentially which is the curse of dimensionality in this approach.

Instead of the tensor product grid in equation 2.105, multivariate sparse grid integration rules can be obtained by construction of a regular sparse grid with the telescoping sum,

$$Q_l^{(d)} = \sum_{|\mathbf{k}|_r \leq l+d-1} \left(\Delta_{k_1}^{(1)} \otimes \dots \otimes \Delta_{k_d}^{(1)} \right), \quad (2.107)$$

where $Q_l^{(d)}$ is an integration formula of dimension d and level l , $\mathbf{k} = (k_1, \dots, k_d)'$ is a d -dimensional vector and $|\cdot|_r$ is the r -norm operator. In equation 2.107, $\Delta_{k_i}^{(1)}$ are the

difference formulas which are also integration formulas obtained from the univariate integration formulas $Q_l^{(1)}$ of different levels with

$$\Delta_{k_i}^{(1)} = (Q_{k_i}^{(1)} - Q_{k_i-1}^{(1)}). \quad (2.108)$$

The grid construction approach in equation 2.107 uses the products whose sum of indices are smaller than a constant $l + d - 1$ out of all possible tensor products. Classical construction of multivariate integration formula given in equation 2.105 would be obtained if $|k|_1 < l + d - 1$, is replaced by $|k|_\infty < l$ in equation 2.107, then the complete tensor grid used in classical approach would be obtained. The construction in equation 2.107 is known as Smolyak's construction (Smolyak, 1963).

The regular sparse grid obtained using the construction in equation 2.107 has an accuracy of

$$\varepsilon(N) = O\left(N^{-r} (\log N)^{(d-1)(r-1)}\right) \quad (2.109)$$

with N points in one dimension of the grid at the boundary for functions with bounded mixed partial derivatives of order up to r .

The regular sparse grid constructed in equation 2.107 involves $O(N(\log N)^{(d-1)})$ degrees of freedom whereas the full tensor product grid involves $O(N^d)$ degrees of freedom. As a result, although not completely, SGI method significantly helps for relaxing the limitation imposed by the number of dimensions in conventional numerical integration approach. Table 2.1 shows the required number of grid points for different number of dimensions and levels of regular sparse grid and full tensor product grid constructed from the trapezoid rule given in equation 2.103 where the dependence on dimension can be seen clearly.

2.2.4.2 SGI based estimation algorithms for the MSV-D

At time t , if the state spaces of \mathbf{h}_t and \mathbf{q}_t are augmented then the total dimension of the resulting state space becomes $d = 2p^2$. Let $Q_l^{(d)}$ be the d -dimensional and $Q_l^{(p)}$ be the p -dimensional sparse grid quadrature rules constructed from the choice of a univariate quadrature rule, $Q_l^{(1)}$ of level l . Although it is not necessary, it is assumed that the level l stays constant through time periods $t = 1, \dots, T$ for notational

simplicity. Let $G_f = \{1, 2, \dots, N_F\}$ and $G_h = \{1, 2, \dots, N_H\}$ be the sets of sparse grid point indices of the formulas $Q_l^{(d)}$ and $Q_l^{(p)}$ respectively. Then, each point $i \in G_f$ in the first grid is characterized by a d -dimensional vector $\mathbf{c}_f^{(i)} = (c_{f,1}^{(i)}, \dots, c_{f,d}^{(i)})'$ and each point $i \in G_h$ in the second grid is characterized by a p -dimensional vector $\mathbf{c}_h^{(i)} = (c_{h,1}^{(i)}, \dots, c_{h,p}^{(i)})'$ representing the raw grid coordinates of the points and corresponding weights $w_F^{(i)}$, $w_H^{(i)}$.

Table 2.1 : Multi-dimensional grid sizes based on the trapezoid rule.

Level	Dimension	Complete Tensor Product Grid	Regular Sparse Grid
2	1	3	3
2	5	243	5
2	10	59,049	21
2	20	3.48×10^9	41
3	1	7	7
3	5	16807	71
3	10	282,475,249	241
3	20	7.97×10^{16}	881
4	1	15	15
4	5	795,375	351
4	10	5.76×10^{11}	2,001
4	20	3.32×10^{23}	13,201
5	1	31	31
5	5	28,629,151	1,471
5	10	8.19×10^{14}	13,441
5	20	6.71×10^{29}	154,881

The constructed sparse grid can be applied to the state space model of MSV-D by providing the suitable integration intervals for each dimension of the state vectors $\mathbf{h}_t = (h_{1t}, \dots, h_{pt})'$ and $\mathbf{q}_t = (q_{1t}, \dots, q_{rt})'$ for $t = 1, \dots, T$. Once the integration intervals provided, raw grid coordinates, $\mathbf{c}_f^{(i)}$ and $\mathbf{c}_h^{(i)}$, of points can be converted to actual point coordinates as

$$\begin{aligned} \mathbf{c}_f^{(i)} = (c_{f,1}^{(i)}, \dots, c_{f,d}^{(i)})' &\rightarrow (\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}) = (h_{1t}^{(i)}, \dots, h_{pt}^{(i)}, q_{1t}^{(i)}, \dots, q_{p(2p-1),t}^{(i)})', \quad i \in G_F \\ \mathbf{c}_h^{(i)} = (c_{h,1}^{(i)}, \dots, c_{h,d}^{(i)})' &\rightarrow \mathbf{h}_t^{(i)} = (h_{1t}^{(i)}, \dots, h_{pt}^{(i)})', \quad i \in G_H. \end{aligned} \quad (2.110)$$

with corresponding weights $w_F^{(i)}$, $w_H^{(i)}$.

Here, two regular sparse grids are constructed, sparse grid of the augmented \mathbf{h}_t and \mathbf{q}_t states is the first and second is the sparse grid of \mathbf{h}_t which is a sub-grid of the first one with adjusted weights and different indices.

Equipped with the sparse grid points, corresponding weights and sets of their indices for the two sparse grids, the integrals in the filtering algorithm given in equation 2.43 in section 2.2.1.2 can be handled numerically as,

$$\begin{aligned}
& p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_{t-1}, \Omega) \approx \\
& \sum_{j \in G_F} p_h(\mathbf{h}_t^{(i)} | \mathbf{h}_{t-1}^{(j)}, \mathbf{q}_{t-1}^{(j)}, \Omega_h) p_q(\mathbf{q}_t^{(i)} | \mathbf{q}_{t-1}^{(j)}, \Omega_q) p(\mathbf{h}_{t-1}^{(j)}, \mathbf{q}_{t-1}^{(j)} | \mathbf{Y}_{t-1}, \Omega) w_f^{(j)}, \quad i \in G_F \\
& p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_t, \Omega) = \\
& \frac{\sum_{k \in G_H} p_y(\mathbf{y}_t | \mathbf{h}_{t+1}^{(k)}, \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \Omega_h) p_h(\mathbf{h}_{t+1}^{(k)} | \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \Omega_h) p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_{t-1}, \Omega) w_h^{(k)}}{\sum_{i \in G_F} \sum_{k \in G_H} p_y(\mathbf{y}_t | \mathbf{h}_{t+1}^{(k)}, \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \Omega_h) p_h(\mathbf{h}_{t+1}^{(k)} | \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \Omega_h) p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_{t-1}, \Omega) w_h^{(k)} w_f^{(i)}}.
\end{aligned} \tag{2.111}$$

Similarly, density based L -step ahead prediction algorithm given in equation 2.45 in section 2.2.1.2 can be obtained using the sparse grid integration as

$$\begin{aligned}
& p(\mathbf{h}_{t+L}^{(i)}, \mathbf{q}_{t+L}^{(i)} | \mathbf{Y}_t, \Omega) \approx \\
& \sum_{j \in G_F} p_h(\mathbf{h}_{t+L}^{(i)} | \mathbf{h}_{t+L-1}^{(j)}, \mathbf{q}_{t+L-1}^{(j)}, \Omega_h) p_q(\mathbf{q}_{t+L}^{(i)} | \mathbf{q}_{t+L-1}^{(j)}, \Omega_q) \\
& p(\mathbf{h}_{t+L-1}^{(j)}, \mathbf{q}_{t+L-1}^{(j)} | \mathbf{Y}_t, \Omega) w_F^{(j)}, \quad i \in G_F,
\end{aligned} \tag{2.112}$$

based on the filtering density obtained in equation 2.111.

The recursive smoothing density algorithm given in equation 2.47 in section 2.2.1.2 can be numerically approximated as

$$\begin{aligned}
& p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_T, \Omega) = \\
& p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_t, \Omega) \cdot \\
& \sum_{i \in G_F} \frac{p(\mathbf{h}_{t+1}^{(j)}, \mathbf{q}_{t+1}^{(j)} | \mathbf{Y}_T, \Omega) p_h(\mathbf{h}_{t+1}^{(j)} | \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \Omega_h) p_q(\mathbf{q}_{t+1}^{(j)} | \mathbf{q}_t^{(i)}, \Omega_q)}{p(\mathbf{h}_{t+1}^{(j)}, \mathbf{q}_{t+1}^{(j)} | \mathbf{Y}_t, \Omega)} w_f^{(i)}, \quad j \in G_F,
\end{aligned} \tag{2.113}$$

for $t = T-1, T-2, \dots, 1$.

The expectation of a function $f(\mathbf{h}_u, \mathbf{q}_u)$ given in equation 2.50 can be numerically approximated by

$$\mathbb{E}\left(f(\mathbf{h}_v, \mathbf{q}_v) \mid \mathbf{Y}_s, \boldsymbol{\Omega}\right) = \sum_{i \in G_F} f(\mathbf{h}_v^{(i)}, \mathbf{q}_v^{(i)}) p(\mathbf{h}_v^{(i)}, \mathbf{q}_v^{(i)} \mid \mathbf{Y}_s) w_F^{(i)}, \quad (2.114)$$

and finally, for the parameter estimation, the log-likelihood function in equation 2.51 can be approximated as

$$\log(p(\mathbf{Y}_T \mid \boldsymbol{\Omega})) = \sum_{t=1}^T \sum_{k \in G_H} \sum_{i \in G_F} \log \left(\frac{p_y(\mathbf{y}_t \mid \mathbf{h}_{t+1}^{(k)}, \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \boldsymbol{\Omega}_h) p_h(\mathbf{h}_{t+1}^{(k)} \mid \mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)}, \boldsymbol{\Omega}_h)}{p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} \mid \mathbf{Y}_{t-1}, \boldsymbol{\Omega}) w_f^{(i)} w_h^{(k)}} \right), \quad (2.115)$$

which is readily available from the denominator of the filtering algorithm given in equation 2.111.

The algorithms based on the SGI method are general and in the cases of static MSV-G and simpler MSV-B models, the algorithms presented here get simpler too. For example the filtering algorithm for the MSV-B model simplifies to

$$p(\mathbf{h}_t^{(i)} \mid \mathbf{Y}_{t-1}, \boldsymbol{\Omega}) \approx \sum_{j \in G_H} p_h(\mathbf{h}_t^{(i)} \mid \mathbf{h}_{t-1}^{(j)}, \boldsymbol{\Omega}_h) p(\mathbf{h}_{t-1}^{(j)} \mid \mathbf{Y}_{t-1}, \boldsymbol{\Omega}) w_f^{(j)}, \quad i \in G_H, \\ p(\mathbf{h}_t^{(i)} \mid \mathbf{Y}_t, \boldsymbol{\Omega}) \approx \frac{p_y(\mathbf{y}_t \mid \mathbf{h}_t^{(i)}, \boldsymbol{\Omega}_h) p(\mathbf{h}_t^{(i)} \mid \mathbf{Y}_{t-1}, \boldsymbol{\Omega})}{\sum_{i \in G_F} p_y(\mathbf{y}_t \mid \mathbf{h}_t^{(i)}, \boldsymbol{\Omega}_h) p(\mathbf{h}_t^{(i)} \mid \mathbf{Y}_{t-1}, \boldsymbol{\Omega}) w_h^{(i)}}, \quad (2.116)$$

with $\boldsymbol{\Omega}_h = \{\boldsymbol{\gamma}, \boldsymbol{\varphi}, \mathbf{V}_\eta, \boldsymbol{\delta}\}$, where only the sparse grid for the states \mathbf{h}_t is used since the dynamic elements \mathbf{q}_t are dropped and $\boldsymbol{\delta}$ is included in the parameter set for correlations.

A method for identifying suitable integration intervals for the states, \mathbf{h}_t , is incorporating the estimates from the Kalman filter and setting integration intervals as

$$\left(\mathbf{h}_{t|s} - z \mathbf{d}_{t|s}, \mathbf{h} + z \mathbf{d}_{t|s} \right) \quad (2.117)$$

where $\mathbf{h}_{t|s}$ is the state estimates and $\mathbf{d}_{t|s}$ is the vector composed of square roots of the diagonal elements of the covariance estimate, $\boldsymbol{\Sigma}_{h,t|s}$ (i.e., standard deviation estimates of the states \mathbf{h}_t) obtained with the Kalman filter algorithms given in section 2.2.2 with equations 2.56, equation 2.57 and equation 2.58. In equation 2.117 z is a scalar tuning parameter greater than 1. Identifying the integration intervals for the states \mathbf{q}_t can be challenging. One of the options is using a generic interval such as $[-2\pi, 2\pi]$ covering most of the interval in accordance with the transformation in equation 2.23 or a smarter approach would be fitting a static version of the model and then using

the parameter estimates for the constant correlations as the mean of the integration interval for the dynamic case and identifying a symmetric interval, where maximum for the upper bound is set to 2π and minimum for the lower bound is -2π , leading to integration intervals that are narrowed down for better precision.





3. COMPUTATIONAL IMPLEMENTATION

In this section important topics on the practical implementation of the estimation algorithms, their computational aspects and parallelization approaches, particularly implementation with the graphics processing unit (GPU), which is one of the research objectives of this study, are discussed.

3.1 Computational Aspects of Estimation Algorithms

In a typical practical application of volatility estimation main purpose is forecasting the future volatilities for decision making given the past information. Long and mid-term predictions are usually composed of several periods whereas the short term forecasting usually refers the next single period or a couple of next periods. The models and estimation algorithms presented in this study are suitable for short term predictions because models do not include components which are common in mid-term to long term forecasting methods (such as external regressors, leading indicators etc.). Although it is trivial to include such components the focus of the study is short term forecasts which is often considered as the next period.

For the main practical objective of producing forecasts, all the prediction algorithms rely on other estimation algorithms namely, filtering, smoothing and parameter estimation in quite different ways for the MCMC and SGI methods because of the computational and algorithmic differences in these approaches. It is important to consider the ordering and prioritization of estimation algorithms and their dependencies for correct evaluation of performance in a practical implementation. Figure 3.1 illustrates the dependence and ordering of different estimation algorithms for practical implementations of the MCMC and SGI based approaches for estimation. MCMC based estimation algorithms mainly rely on the smoothing algorithm whereas the SGI based algorithms rely on filtering algorithm. Filtering algorithm for the MCMC approach and smoothing algorithm for the SGI approach are not required for prediction and parameter estimation algorithms. In a typical implementation, MCMC based approaches require frequent (each period) execution

of prediction and smoothing algorithms whereas SGI based approach requires execution of filtering and prediction algorithms.

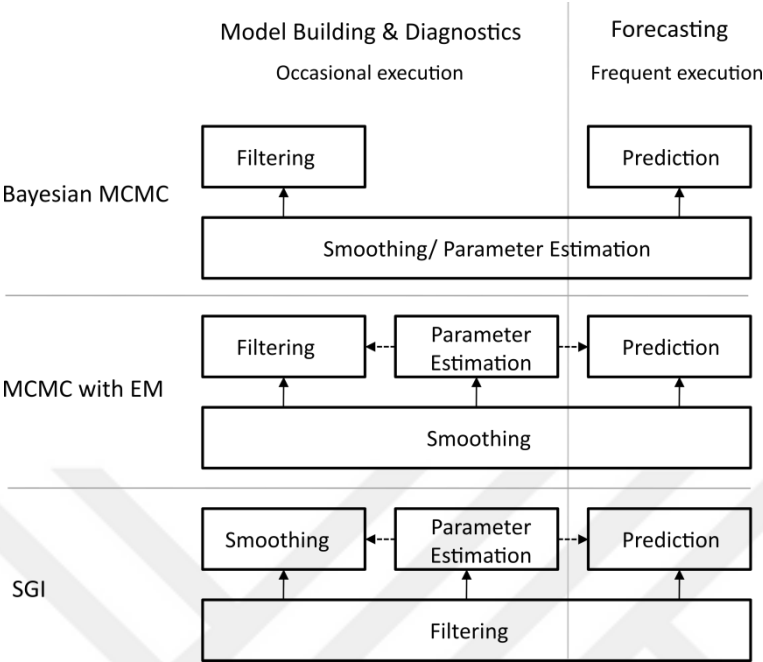


Figure 3.1 : Estimation algorithm dependencies.

Another important difference between the MCMC based estimation algorithms and SGI based algorithms is the batch versus sequential structure which is also related with the algorithm dependency differences discussed above. As illustrated in Figure 3.2, sequential algorithms use the estimation from the previous step and the current information set to make estimations whereas the batch algorithms needs all the information in the past and does not use any estimations from the history.

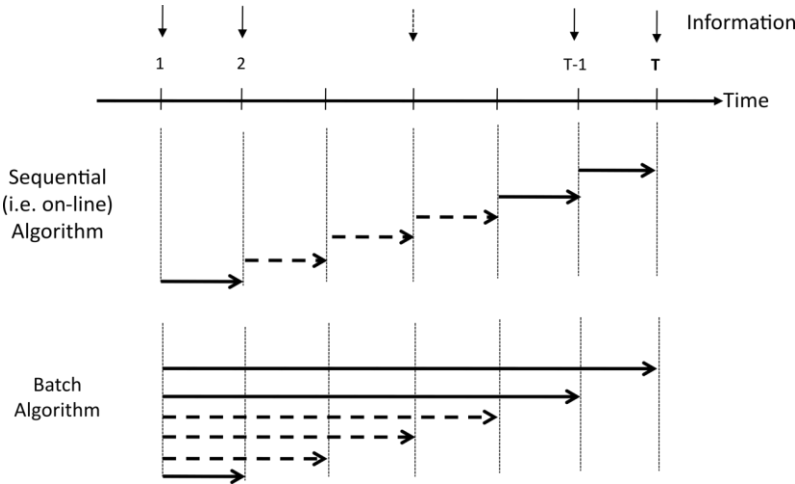


Figure 3.2 : Sequential (i.e. on-line) vs. batch algorithm.

The algorithms based on the MCMC methods all have a batch structure inherited from the underlying logic and structure of the MCMC methods which put the smoothing algorithm at the basis. The estimation algorithms based on sparse grid integration method are sequential methods since they all rely on filtering algorithm. The advantage of sequential algorithms is that the computational burden can be split across time periods which can be a critical advantage in practical applications.

For both MCMC with EM and Bayesian MCMC approaches presented in section 2.2.3.2 the smoothing algorithm is the main algorithm used by all other estimation algorithms. The illustration of the smoothing algorithms for the MCMC with EM and the Bayesian MCMC approaches are given in Figure 3.3 and Figure 3.4 respectively. The basic structure of the smoothing algorithms in these approaches is composed of two main loops where the outer loop constructs samples based on the previous sample by running a inner loop computing the state estimates over time periods with addition of parameter estimates to the inner loop for the Bayesian case.

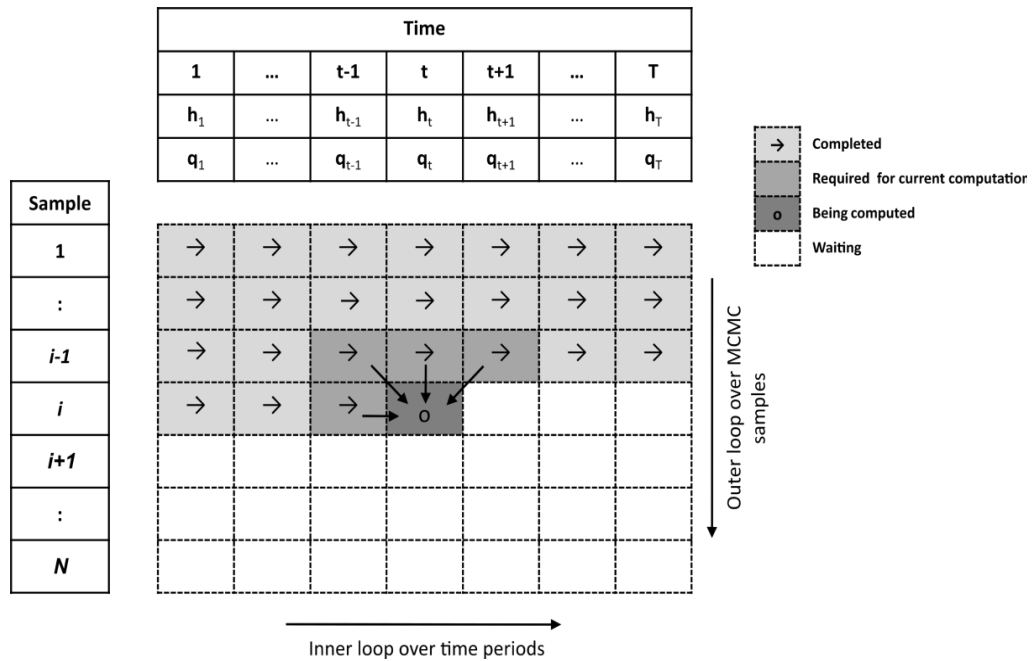


Figure 3.3 : Smoothing algorithm of MCMC with EM approach.

Let P be the number of parameters then it can be seen from Figure 3.3 and Figure 3.4 that $T \times N$ computations for the MCMC with EM approach and $(T + P) \times N$ computations for the Bayesian MCMC approach are required, where each computation involves random number generation, density function evaluation and

matrix operations including the inversion and determinant for matrices in size of maximum $p(2p-1) \times p(2p-1)$.

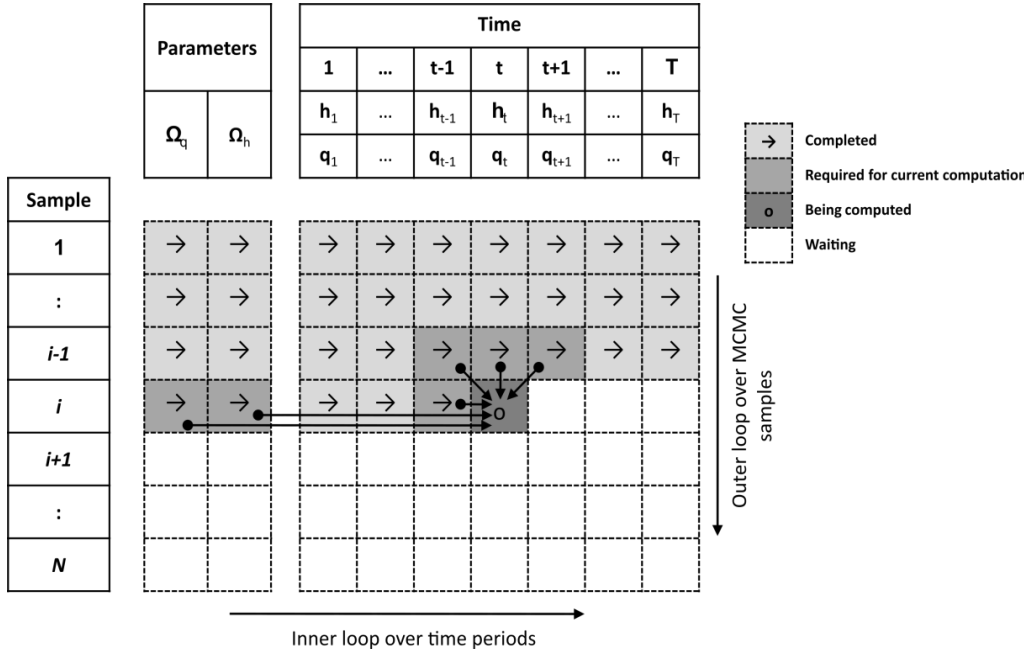


Figure 3.4 : Smoothing algorithm of Bayesian MCMC approach.

The filtering algorithms for the MCMC based approaches are computationally more intensive than the smoothing algorithms since filtering basically executes the smoothing algorithm repeatedly over time periods. Then the computational requirement for the both MCMC based methods for filtering becomes $N \times T \times (T-1)/2$. The prediction algorithm for the MCMC based methods given in equation 2.71 is a resampling algorithm has an additional computational burden of $L \times N$. For the parameter estimation Bayesian MCMC approach does not introduce additional computational burden on top of the smoothing algorithm requiring $(T+P) \times N$ computations whereas MCMC with EM algorithm introduces the additional computational requirement on the inherited computational requirement of smoothing resulting a $z \times T \times N$ computations where z represents the complexity of the EM algorithm which depends on the state space dimension thus the number of model parameters and the choice of the optimization routine.

As depicted in Figure 3.1, the filtering algorithm given in equation 2.111 is the main algorithm required by all other estimation algorithms. In Figure 3.5, the filtering algorithm for the SGI based approach is illustrated.

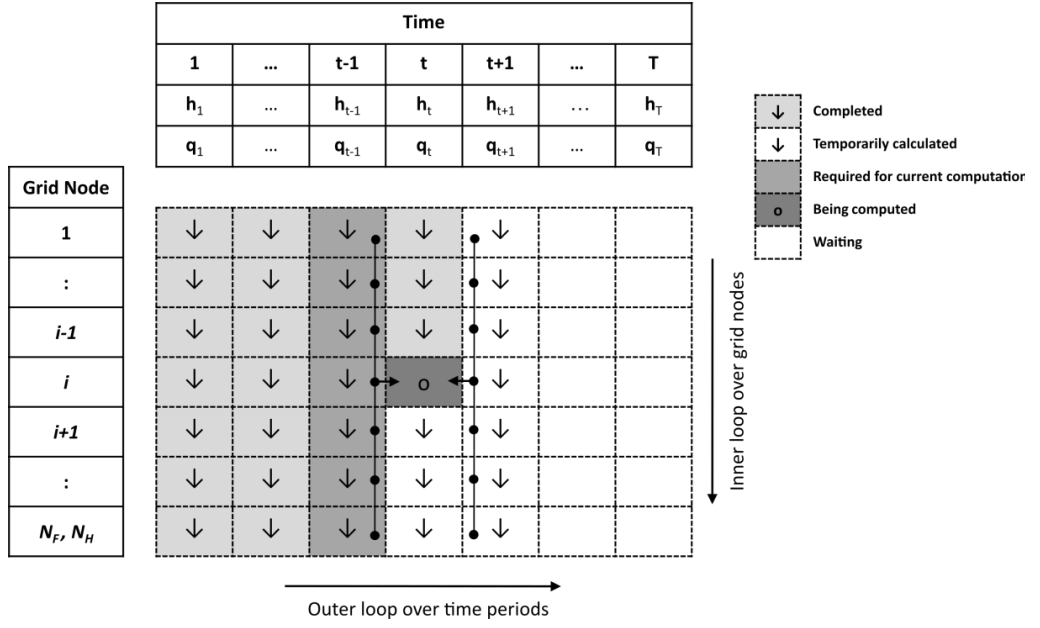


Figure 3.5 : Filtering algorithm of SGI approach.

The filtering algorithm based on the SGI methods is composed of two main loops where the outer loop constructs the estimates for time periods using two inner loops enumerating the sparse grids of previous and next time periods. In Figure 3.5, it can be more clearly seen that, this filtering algorithm is sequential in the time domain in contrast with the MCMC based algorithms.

The filtering algorithm given in equation 2.111 and illustrated in Figure 3.5 requires $T \times (N_F^2 + N_F^2 \times N_H)$ computations where N_F and N_H are the number of points in the sparse grids G_F and G_H which have dependency on the level l of the underlying univariate integration formula and the dimension of the associated state space. See Table.2.1 for the number of sparse grid points with the trapezoid rule as the underlying formula. G_F is the sparse grid of the augmented state space of \mathbf{q}_t and \mathbf{h}_t which has maximum $2p^2$ dimensions in the complete dynamic MSV-D case and minimum p dimensions in the static MSV-D case (i.e. MSV-B and MSV-G models). G_H is the sparse grid for the state space of \mathbf{h}_t which has p dimensions. Thus, under the static specifications (i.e. MSV-G and MSV-B) G_F reduces to G_H and computational burden can be represented as $T \times (N_H^2 + N_H^3)$. There is an extra reduction in the computational requirement of the MSV-B model due to the lack of temporal dependency on the next time step then the resulting computational burden

becomes $T \times (N_H^2 + N_H^2)$ for the MSV-B model. Although, the curse of dimension is overcome to a certain degree, polynomial increase in the dimension of the augmented state-space of \mathbf{q}_t and \mathbf{h}_t makes it still hard for SGI to be used in models where the number of assets is high and complete dynamic structure is imposed, however SGI based filtering is still feasible for multidimensional models with static structures or restricted dynamic structures. The smoothing algorithm given in equation 2.113 has an additional requirement of $T \times N_F^2$ computations and the prediction algorithm in equation 2.112 requires $L \times N_F^2$ computations. In the SGI based approach, each computation involves density function evaluation and matrix operations including inversion and determinant for matrices in size of maximum $p(2p-1) \times p(2p-1)$ for the complete dynamic structure and minimum $4p^2$ for the static specifications such as the MSV-G and MSV-B. In the SGI approach, parameter estimation incorporates filtering algorithm depicted in Figure 3.4, thus it inherits the computational requirement of filtering algorithm and resulting computational burden becomes $z \times T \times (N_H^2 + N_H^3)$ where z depends on the state space dimension, thus the number of model parameters, and the choice of the optimization routine.

It is noteworthy that the computational burden of the algorithms provided above can be split among time periods and one-step computational burden of the estimation algorithms reduces by $1/T$ of the requirements presented above for the SGI approach since SGI based algorithms are sequential which is not the case for MCMC based methods as discussed before.

In the implementation of SGI based estimation algorithms a complication is the accumulation of numerical errors as a result of the recursive nature of the algorithms. To overcome this complication, a correction satisfying the requirement

$$\int p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_s, \mathbf{\Omega}) d\mathbf{h}_t d\mathbf{q}_t \approx \sum_{i \in G_F} p(\mathbf{h}_t^{(i)}, \mathbf{q}_t^{(i)} | \mathbf{Y}_s, \mathbf{\Omega}) w_F^{(i)} = 1 \quad (3.1)$$

from the basic property of probability density functions should be implemented with an extra computational cost of recomputing $p(\mathbf{h}_t, \mathbf{q}_t | \mathbf{Y}_s, \mathbf{\Omega})$.

A final note about the implementation is the task of construction of the regular sparse grid used in the SGI based methods. The construction of regular sparse grids which involves calculation of the weights and raw coordinates of points for an arbitrary

level l and arbitrary dimension d for a given univariate integration formula requires a separate algorithm which does not have fixed loops (see appendix C.2 for the C functions used in the construction of regular sparse grids based on the trapezoid rule). For high dimensions and high values of integration rule levels, construction of the sparse grid has its own computational burden however the regular sparse grids for all time periods can be constructed in advance and kept in memory so computational requirement for the construction of the sparse grid does not necessarily increase the computational requirement of the overall estimation algorithm if the total size of the resulting grids are not problematic for memory usage.

3.2 Parallelization Approaches for the MCMC Based Algorithms

The smoothing algorithm for the MCMC with EM approach illustrated in Figure 3.3 has dependencies both in the spatial and temporal domain in a batch structure at each sample and time step. A parallelization can be achieved by decomposing the inner loop (i.e. the time domain) for concurrency as illustrated in Figure 3.6.

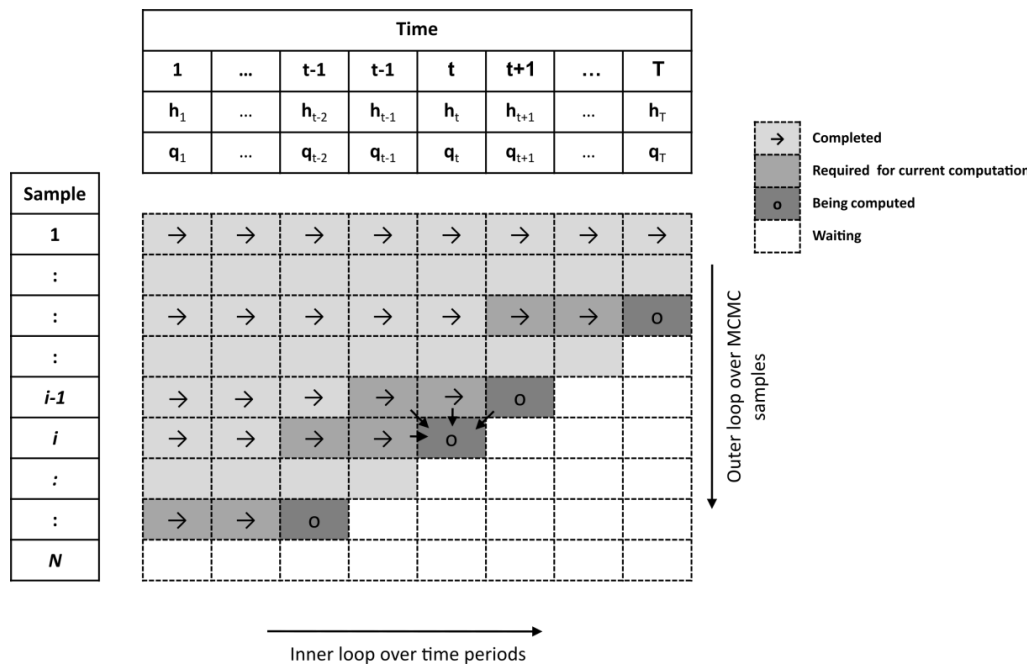


Figure 3.6 : Parallel MCMC based smoothing algorithm.

In this approach, if the computations for each time step of each sample is considered as a process then the maximum number of total processes that can be executed in parallel is given by,

$$np = \begin{cases} T/2, & \text{if } T \text{ is even,} \\ (T+1)/2, & \text{if } T \text{ is odd.} \end{cases} \quad (3.2)$$

after the first three time steps of the first sample and before the last three time steps of the last sample. All the estimation algorithms using the smoothing algorithm (see Figure 3.1) can benefit from the accelerated smoothing algorithm described here for the MCMC with EM approach.

The parallelization approach for the MCMC with EM approach illustrated in Figure 3.6 can not be applied to the smoothing algorithm of the Bayesian MCMC approach illustrated in Figure 3.4 because of the additional dependency on the parameters which are out of the temporal domain preventing the decomposition illustrated in Figure 3.6. MCMC based estimation methods such as the Bayesian MCMC where the parameter space and state space are augmented are known to be hard to parallel algorithms (Rosenthal, 2000). An option is using parallel independent chains, each same as the serial version illustrated in Figure 3.4 with different random number sequences and then combining the samples. The main drawback of this approach is the burn-in samples (i.e first M samples) which are discarded out of the N samples. There is not an exact theoretical number but burn-in samples are usually the 20%-30% percent of the all samples in most of the studies. The parallelization with the parallel chains requires that either generating M burn-in samples for each chain or generating the M burn-in sample in one process and then splitting up the remaining samples in parallel where in either case there is a limit on the maximum theoretical speed-up which is N/M .

Filtering algorithms based on the MCMC methods uses the smoothing algorithm for the MCMC with EM approach repeatedly for $t = 1, \dots, T$, thus the parallelization approach described for smoothing automatically inherited by the filtering algorithm however an additional acceleration can be achieved by decomposing the runs of smoothing algorithm for each t , since they are independent.

3.3 Parallelization Approach for the SGI Based Estimation Algorithms

In the filtering algorithm based on the SGI method illustrated in Figure 3.5 the computations at each grid node for a given time step has dependency only to the previous step in the time domain. An highly efficient parallelization can be achieved

by decomposing the sparse grid nodes of each time step to processes. The parallel filtering algorithm based on SGI method is illustrated in Figure 3.7. In this approach, if computations at each grid node for a given time step is considered as a process, then maximum number of processes that be executed in parallel equals the sparse grid size N_F .

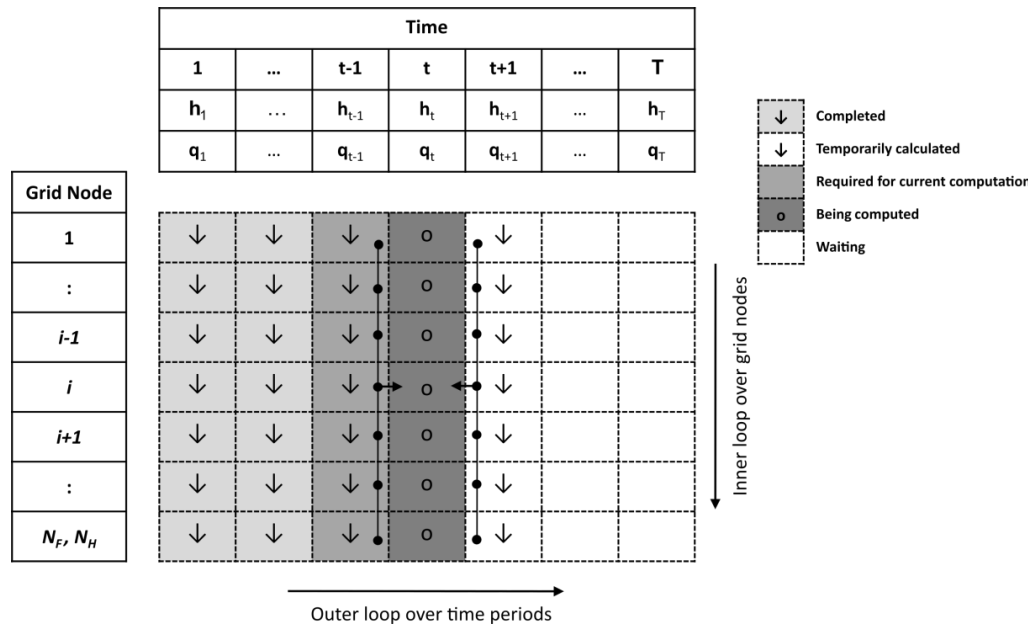


Figure 3.7 : Parallel SGI based filtering algorithm.

All other estimation algorithms based on the SGI method are sequential and does not possess dependencies within a time step so the described parallelization approach is valid for them as well.

Although the discussion on the parallel algorithms presented can be extended to the distributed memory and processors systems by addressing the obvious intense communication overhead required by the algorithms and their memory complexity, this study is limited with the shared memory multi processor systems such as GPUs or modern many-core computers. The parallel algorithms described above achieve the most efficient parallelism by exploiting the advantage of shared memory systems. And the memory requirements of the algorithms presented above are not beyond the resource limits of modern computers and GPUs thus is not a bottleneck in implementation.

3.4 Notes on the GPU Implementation

In this study, parallel MCMC smoothing algorithm and parallel SGI filtering algorithm described in section 3.3 and illustrated in Figure 3.6 and Figure 3.7 are implemented on GPU.

In a typical GPU implementation, parallel execution is achieved by device kernel functions which require number of blocks and total number of threads in each block in the function call along with other usual arguments such as pointers and data structures. The number of blocks n_{blocks} and block size n_{size} depend on each other and are limited by the specification of the device used. The number of blocks for the device kernel calls can be obtained by

$$n_{blocks} = \frac{n_{threads} + n_{size} - 1}{n_{size}} \quad (3.3)$$

where the number of threads $n_{threads}$ depends on the parallel algorithm. In implementation of the parallel MCMC smoothing algorithm in Figure 3.6 each computation at time t and sample i is considered as a single thread and the number of threads that can be executed in parallel is set equal to the number of processes that can be executed parallel, np , given in equation 3.2.

In a similar approach the parallel implementation of the parallel SGI filtering algorithm is performed by setting the number of threads that can be executed in parallel $n_{threads}$ to the number of processes that can be executed in parallel, $np = N_F$.

Although GPU architectures and designs are much more suitable and provide tools for exploiting lower level and more granular parallelism constructed on simpler but repeated computations, the approach considering a set of more complex computations composed of random number generation, density function evaluations and matrix operations as a single thread works quite well too in terms of performance with only the cost of considerable coding effort involving the development of all the device counterparts of the serial functions used in the computations. See appendix C.3 for the example codes on GPU functions, kernels and their usage.

In the implementation of MCMC algorithms, random number generation is an important consideration and use of robust custom libraries and implementations are

required. In GPU implementations, parallel handling capability of random number generation provided by the used library is important.

Some other details regarding the implementation can be found in section 4.1 where the software and hardware used in the study are summarized as part of the methodology.





4. METHODOLOGY

In this section the methodology followed in the study is presented. The methodology of the study considers the following main objectives:

- a. Illustrate and evaluate the proposed model (i.e. MSV-D model) and its developed estimation method based on Bayesian MCMC,
- b. Compare the performance of the the proposed estimation approach (i.e. SGI based approach) for MSV models with the MCMC based approach,
- c. Evaluate the implications of GPU acceleration support on the estimation algorithms for the MSV models,

With these objectives, the methodology consists of computational studies and analyses conducted on simulated and empirical data sets.

This chapter is organized as follows. Section 4.1 provides details of the software programs and hardware used in the computational studies. Section 4.2 describes the methodology used for illustration and evaluation of the proposed model, MSV-D. And in section 4.3 the methodology used to assess and compare the SGI based estimation algorithms with the MCMC based algorithms and to evaluate the implications of GPU support in estimation algorithms are presented together.

4.1 Software Programs and Hardware

The estimation algorithms used in computational studies are implemented using the C programming language for both the serial and parallel implementations. NVIDIA™ CUDA® 4.0 platform is used for the development and programming of the parallel algorithms for GPUs.

Table 4.1 provides the summary of the estimation algorithms and their implementations.

Table 4.1 : Implemented estimation algorithms.

Approach	Serial CPU	Parallel GPU
Kalman Filter	Smoothing Filtering Prediction	
MCMC	Smoothing Filtering Prediction Parameter Estimation	Smoothing
SIG	Smoothing Filtering Prediction Parameter Estimation	Filtering

As discussed in section 3, the smoothing algorithm for the MCMC approach and the filtering algorithm for the SIG approach are base algorithms which other algorithms depend on, thus parallel implementations of these two algorithms automatically make other algorithms parallel too.

In implementations of the algorithms, the random number generation in the serial programs is performed with a C implementation of LAPACK `_larnv` routines and in GPU programs, CURAND library is used.

In the parameter estimation algorithms of the MCMC with EM and SIG approaches, NLOpt C library is used for non linear optimization methods.

Computational studies and numerical applications are performed on a computer with Intel Core I7-920 processor, 16 GB memory and NVIDIA™ Tesla C1060 compute processor as the GPU.

For comparison purposes, in some of the parts of the computational studies GARCH models are used and they are fitted in R version 3.1.0 with `rugarch`, `rmgarch` and `ccgarch` packages.

4.2 Assessment of the MSV-D Model

To illustrate and evaluate the proposed model, namely the MSV-D model, and its Bayesian MCMC estimation algorithm, two simulated data sets and an empirical data set are used.

Two simulated data sets of five asset returns are generated with known parameters for $T = 5,000$ time periods. First data set is constructed with a static specification

which is equivalent to the MSV-G model whereas the second simulated data set has a complete dynamic structure based on the MSV-D model. It is noteworthy that, the static specification which is equivalent to the MSV-G model, is essentially a special case of the MSV-D model parameterized with the proposed approach.

First set of simulated data is obtained by setting the parameters of the AR(1) process of the log-volatilities, \mathbf{h}_t , as

$$\gamma_i = 0.25, \quad \varphi_i = 0.97, \quad \sigma_{\eta,i}^2 = 0.04, \quad i = 1, \dots, 5 \quad (4.1)$$

and the static correlation structure is determined by the correlation matrix,

$$\mathbf{P} = \begin{pmatrix} 1 & 0.6 & 0.6 & 0.6 & 0.6 & -0.4 & -0.3 & -0.3 & -0.3 & -0.3 \\ 0.6 & 1 & 0.6 & 0.6 & 0.6 & -0.3 & -0.4 & -0.3 & -0.3 & -0.3 \\ 0.6 & 0.6 & 1 & 0.6 & 0.6 & -0.3 & -0.3 & -0.4 & -0.3 & -0.3 \\ 0.6 & 0.6 & 0.6 & 1 & 0.6 & -0.3 & -0.3 & -0.3 & -0.4 & -0.3 \\ 0.6 & 0.6 & 0.6 & 0.6 & 1 & -0.3 & -0.3 & -0.3 & -0.3 & -0.4 \\ -0.4 & -0.3 & -0.3 & -0.3 & -0.3 & 1 & 0.6 & 0.6 & 0.6 & 0.6 \\ -0.3 & -0.4 & -0.3 & -0.3 & -0.3 & 0.6 & 1 & 0.6 & 0.6 & 0.6 \\ -0.3 & -0.3 & -0.4 & -0.3 & -0.3 & 0.6 & 0.6 & 1 & 0.6 & 0.6 \\ -0.3 & -0.3 & -0.3 & -0.4 & -0.3 & 0.6 & 0.6 & 0.6 & 1 & 0.6 \\ -0.3 & -0.3 & -0.3 & -0.3 & -0.4 & 0.6 & 0.6 & 0.6 & 0.6 & 1 \end{pmatrix} \quad (4.2)$$

In the second simulated data set, instead of the static correlation matrix \mathbf{P} , a dynamic structure is introduced by AR(1) processes of \mathbf{q}_t with the parameters,

$$\theta_i = 0.97, \quad \sigma_{\omega,i}^2 = 0.01 \quad (4.3)$$

and taking the matrix,

$$\mathbf{P} = \begin{pmatrix} 1 & 0.81 & 0.78 & 0.75 & 0.71 & -0.66 & -0.60 & -0.54 & -0.46 & -0.38 \\ 0.81 & 1 & 0.78 & 0.75 & 0.71 & -0.66 & -0.60 & -0.54 & -0.46 & -0.38 \\ 0.78 & 0.78 & 1 & 0.75 & 0.71 & -0.66 & -0.60 & -0.54 & -0.46 & -0.38 \\ 0.75 & 0.75 & 0.75 & 1 & 0.71 & -0.66 & -0.60 & -0.54 & -0.46 & -0.38 \\ 0.71 & 0.71 & 0.71 & 0.71 & 1 & -0.66 & -0.60 & -0.54 & -0.46 & -0.38 \\ -0.66 & -0.66 & -0.66 & -0.66 & -0.66 & 1 & 0.60 & 0.54 & 0.46 & 0.38 \\ -0.60 & -0.60 & -0.60 & -0.60 & -0.60 & 0.60 & 1 & 0.54 & 0.46 & 0.38 \\ -0.54 & -0.54 & -0.54 & -0.54 & -0.54 & 0.54 & 0.54 & 1 & 0.46 & 0.38 \\ -0.46 & -0.46 & -0.46 & -0.46 & -0.46 & 0.46 & 0.46 & 0.46 & 1 & 0.38 \\ -0.38 & -0.38 & -0.38 & -0.38 & -0.38 & 0.38 & 0.38 & 0.38 & 0.38 & 1 \end{pmatrix} \quad (4.4)$$

as the mean to obtain the process intercept δ using the transformations in equation 2.19 to equation 2.24 of MSV-D model specification.

On the two sets of simulated asset returns data, static and dynamic models are fitted using the Bayesian MCMC estimation approach developed in section 2.2.3.2. The prior density parameters for the Bayesian MCMC estimation approach for the static model are set as follows:

$$\begin{aligned}
p(\boldsymbol{\gamma}) &\propto 1 \\
p(\boldsymbol{\Phi}) &= \prod_{i=1}^p f_B\left(\frac{\varphi_i + 1}{2}; \alpha_i, \beta_i\right), \quad \alpha_i = 20, \beta_i = 1 \\
p(\mathbf{V}_\eta) &= \prod_{i=1}^p f_{IG}(\sigma_{\eta,i}^2; \alpha_{0i}, \beta_{0i}), \quad \alpha_{0i} = 2.5, \beta_{0i} = 0.025
\end{aligned} \tag{4.5}$$

where $f_B(\cdot)$ is the beta distribution, $f_{IG}(\cdot)$ is the inverse gamma distribution. In addition to the densities above the prior densities,

$$\begin{aligned}
p(\boldsymbol{\delta}) &\propto 1 \\
p(\boldsymbol{\theta}) &= \prod_{i=1}^r f_B\left(\frac{\theta_i + 1}{2}; \alpha_i, \beta_i\right), \quad \alpha_i = 20, \beta_i = 1 \\
p(\mathbf{V}_\omega) &= \prod_{i=1}^r f_{IG}(\sigma_{\omega,i}^2; \alpha_{0i}, \beta_{0i}), \quad \alpha_{0i} = 2.5, \beta_{0i} = 0.025
\end{aligned} \tag{4.6}$$

are considered for the dynamic model. For sampling the log-volatilities, \mathbf{h}_t , the density in equation 2.69 is used with the tuning scalar parameter $c = 6$.

With these settings, the results composed of the parameter and smoothing estimates and their standard errors are obtained which constitute a basis for comparison with the true values of parameters and log-volatilities providing statistical evidence on the how the proposed MSV-D model and its Bayesian MCMC estimation algorithm work in capturing the patterns.

Parameter estimates and 95% credible intervals are computed from the resulting samples representing the posterior distributions of parameters obtained from the executed Bayesian MCMC algorithm.

The comparison of the estimated and actual log-volatilities are performed based on the root mean squared error (RMSE) as the criteria. RMSE of the log-volatility series \mathbf{h}_t is computed as,

$$\text{RMSE}_{\mathbf{h}} = \sqrt{\frac{1}{T - t_S + 1} \sum_{t=1}^T (\mathbf{h}_{t|S} - \mathbf{h}_t)^2} \tag{4.7}$$

where $(s, t_s) = (T, 1)$ and $\mathbf{h}_{t|s}$ are the smoothing estimates obtained by the Bayesian MCMC estimation algorithm and \mathbf{h}_t is the actual log-volatility at time t in the simulated data. T is the number of time periods which is 5,000. Similarly, for the dynamic setting, RMSEs of a time varying parameter such as the dynamic correlation coefficient is calculated by replacing $\mathbf{h}_{t|T}$ and \mathbf{h}_t with the estimate and actual value of the parameter at time t in equation 4.7.

An inefficiency factor which is a diagnostic measure indicating how well the MCMC mixes is computed by calculating the ratio of the numerical variance of the posterior mean to the variance of the sample mean based on uncorrelated draws. The inefficiency factor shows how many times the number of uncorrelated samples must be drawn for reliable estimates. See details on inefficiency factor as a diagnostic measure in (Chib, 2001). The results of the applications of MSV-D models are shown in section 5.1.

An empirical analysis was also conducted to illustrate how the MSV-D model and its Bayesian MCMC estimation algorithm work on real data. The data include the price series of S&P500 index, IBM and Intel Corporation (INTC) stock price series from January 1, 1996 to August 31, 2015 including 4,951 observations excluding the days when the markets are closed due to holidays, weekends and other special days. The data are dividend adjusted.

Asset returns are defined as the log-differences of price series and the price series are converted to return series accordingly. The returns data are checked for the AR(1) effects and removed from the data. The data were tested for the heteroskedasticity using Ljung-Box test on the squared returns data series resulting with $p < 0.01$ for all asset return series, rejecting the null hypothesis of homoskedasticity and independence.

A complete dynamic specifications of MSV-D model was fit on the data using the Bayesian MCMC estimation approach described in section 2.2.3.2 with the priors given in equation 4.5 and equation 4.6.

The log-volatility and time varying correlation estimations are compared with estimations from an exponential version of the Dynamic Conditional Correlation GARCH (DCC-GARCH) model of Engle (2002). DCC-EGARCH (1,1) model is given by

$$\log(h_{i,t}) = \omega_i + \alpha_i z_{i,t-1} + \gamma_i \left(|z_{t-1}| - E[|z_{t-1}|] \right) + \beta_i \log(h_{i,t-1}), \quad \mathbf{z}_t \sim N(\mathbf{0}, \mathbf{P}_t) \quad (4.8)$$

where the dynamic correlation matrix \mathbf{P}_t is modeled by

$$\begin{aligned} \mathbf{P}_t &= (\mathbf{Q}_t \odot \mathbf{I})^{-1/2} \mathbf{Q}_t (\mathbf{Q}_t \odot \mathbf{I})^{-1/2} \\ \mathbf{Q}_t &= (1-a-b)\mathbf{Q} + a\mathbf{z}_{t-1}\mathbf{z}'_{t-1} + \beta\mathbf{Q}_{t-1}, \quad a+b < 1 \text{ and } a, b > 0 \end{aligned} \quad (4.9)$$

In equation 4.9, \mathbf{Q} is the sample covariance matrix of \mathbf{z}_t . The DCC-EGARCH model is fitted in R with the `rmgarch` package.

The results of the empirical application are shown in section 5.2.

4.3 Assessment of the Estimation Algorithms

In order to compare the proposed SGI based estimation algorithms with MCMC based estimation algorithms in terms of accuracy and computational requirements a simulation study was designed as follows.

The simulation study is based on repeated generation of artificial return series using the MSV-B model specification with known parameter values,

$$\gamma_i = 0.25, \quad \varphi_i = 0.95, \quad \sigma_{\eta,i}^2 = 0.04, \quad \rho_{\varepsilon\varepsilon,ij} = 0.6, \quad i = 1, \dots, p \quad (4.10)$$

with $\mathbf{q}_t = \boldsymbol{\delta}$ is obtained with the transformations from equation 2.19 to equation 2.24 of MSV-D model specification based on the correlation matrix $\mathbf{P}_{\varepsilon\varepsilon} = (\rho_{\varepsilon\varepsilon,ij})$.

Different artificial return series are generated for different number of assets (i.e. dimension of the state space of log-volatilities) $p = \{1, 2, 3\}$, for fixed $T = 1,000$ periods.

Then, on each simulated return series data, both the SGI and MCMC based estimation algorithms are used to calculate smoothing, filtering, one-step ahead prediction estimates of log-volatilities and estimates of parameters. In the application of SGI based estimation algorithms, integration levels $l = \{4, 5, 6\}$ and in the application of MCMC based estimation algorithms sample sizes $N = \{100,000, 200,000, 400,000\}$ are used.

The procedure of generating simulated data and estimation described above is repeated $R = 100$ times to capture the statistics on estimations comparing with the

true values of the parameters and log-volatilities. The accuracy of the parameter estimates is measured by the root mean squared error (RMSE) calculated by

$$\text{RMSE}_{\Omega} = \sqrt{\frac{1}{R} \sum_{i=1}^R \left(\hat{\Omega}_h^{(i)} - \Omega_h^{(i)} \right)^2} \quad (4.11)$$

where $R = 100$ is the number of repeats and $\hat{\Omega}_h^{(i)}$ is the particular parameter estimate and $\Omega_h^{(i)}$ is the true value of the particular parameter used to generate the data at repeat i .

The accuracy of the log-volatility estimates is measured by the root mean squared error (RMSE) calculated by

$$\text{RMSE}_h = \frac{1}{p} \sum_{j=1}^p \sqrt{\frac{1}{T - t_s + 1} \frac{1}{R} \sum_{t=t_s}^T \sum_{j=1}^R \left(h_{j,t_s}^{(i)} - h_{j,t}^{(i)} \right)^2} \quad (4.12)$$

where $T = 1,000$ is the number of periods, $R = 100$ is the number of repeats, p is dimension of the state-space, $h_{j,t_s}^{(i)}$ is the log-volatility estimate and $h_{j,t}^{(i)}$ is the true value of the log-volatility from the simulated data at repeat i . Depending on the type of the estimate, the indices become $(s, t_s) = (t, 1)$ for filtering, $(s, t_s) = (T, 1)$ for smoothing and $(s, t_s) = (2, t-1)$ for one-step ahead prediction.

Computational requirements and assessments regarding the GPU acceleration were obtained by measuring the execution times of the estimation algorithms for both serial CPU implementations and parallel GPU implementations and calculating the speed up defined as the ratio of the serial execution time to parallel execution time. Multiple measurements of executions are averaged and single measurements are reported in some serial cases that take too much time. The results of the simulation study are given in section 5.3.

4.4 Illustration of SGI Based Estimation Algorithms on Empirical Data

Proposed SGI based algorithms are applied on empirical data to illustrate how SGI based estimation algorithms perform on real data. The data includes foreign-exchange rate series of Euro(EUR)/Turkish Lira(TRL) and US Dollar(USD)/Turkish Lira(TRL) from March 1, 2001 to September 30, 2015. There are 3,669 trading days for each series. The returns are defined by the log-difference of each series. The

return series were checked for AR(1) effects and those effects were removed from the series.

The data is tested for the heteroskedasticity using Ljung-Box test on the squared returns data series resulting with $p < 0.01$ for all asset return series rejecting the null hypothesis of homoskedasticity and independence.

For illustration purposes three models are fitted to the data two of which are actually the same MSV-B model but estimated with SGI based and MCMC based estimation algorithms and a diagonal CCC-GARCH model of Bollerslev (1990) given by

$$h_{i,t} = \omega_i + \alpha_i \varepsilon_{i,t-1}^2 + \beta_i h_{i,t-1}, \quad \varepsilon_{i,t} = h_{i,t}^{1/2} z_{i,t}, \quad \mathbf{z}_t \sim N(\mathbf{0}, \mathbf{P}) \quad (4.13)$$

The CCC-GARCH model is fitted in R with the ccgarch package.

In fitting the MSV-B model using the SGI based estimation algorithms, trapezoid rule of level $l = 8$ is used as the basis univariate integration formula which results in a two dimensional sparse grid having 1,793 points for each time period. For the MCMC based algorithms, sample size of $N = 400,000$ is used and 25% of them are discarded as burn-in samples. For the covariance of the proposal distribution of log-volatilities in MCMC and for identifying the integration intervals in SGI scalar tuning parameter $c = 6$ and $z = 6$ are used in equation 2.69 and equation 2.117 respectively.

Model fit statistics standard error of estimates of parameters and RMSE of log-volatilities are used for evaluation. RMSE of the log-volatility series are calculated by equation 4.7 setting $(s, t_S) = (T, 1)$ for smoothing and $(s, t_S) = (2, t-1)$ for one-step predictions. The results of the empirical analysis are given in section 5.4.

5. RESULTS

In this section the results of the computational studies and numerical applications described in section 4 regarding the proposed MSV-D model and its Bayesian MCMC estimation method, proposed SGI based estimation algorithms and computational implications of GPU usage in estimation of MSV models are presented.

5.1 MSV-D Model on Simulated Returns Data

The proposed MSV-D model and its Bayesian MCMC estimation method were applied on two sets of simulated asset returns data as described in section 4.2 for illustration and evaluation purposes.

First simulated data set is based on the static specification equivalent to the MSV-G model and is a special case of the MSV-D model parameterized with the proposed approach.

Figure 5.1 shows the simulated return series and log-volatilities of five assets with the static specifications given by equation 4.1 and equation 4.2. In Figure 5.1, volatility clusterings, co-movements of asset returns and co-movement of log-volatilities are visible from the charts.

The simulated return series parts of the simulated data are used to fit a static model to the returns data using the Bayesian MCMC estimation algorithm developed for the MSV-D model in section 2.2.3.2 which produces both the parameter estimates and smoothing estimates of the log-volatilities at the same time.

In the estimation algorithm sample size of $N = 400,000$ is used and 25% of them are discarded as burn-in samples. The parameter estimates and log-volatility estimates obtained are then compared with the true values of parameters and log-volatilities which are used to simulate the data to see how the MSV-D model and its estimation algorithm perform.

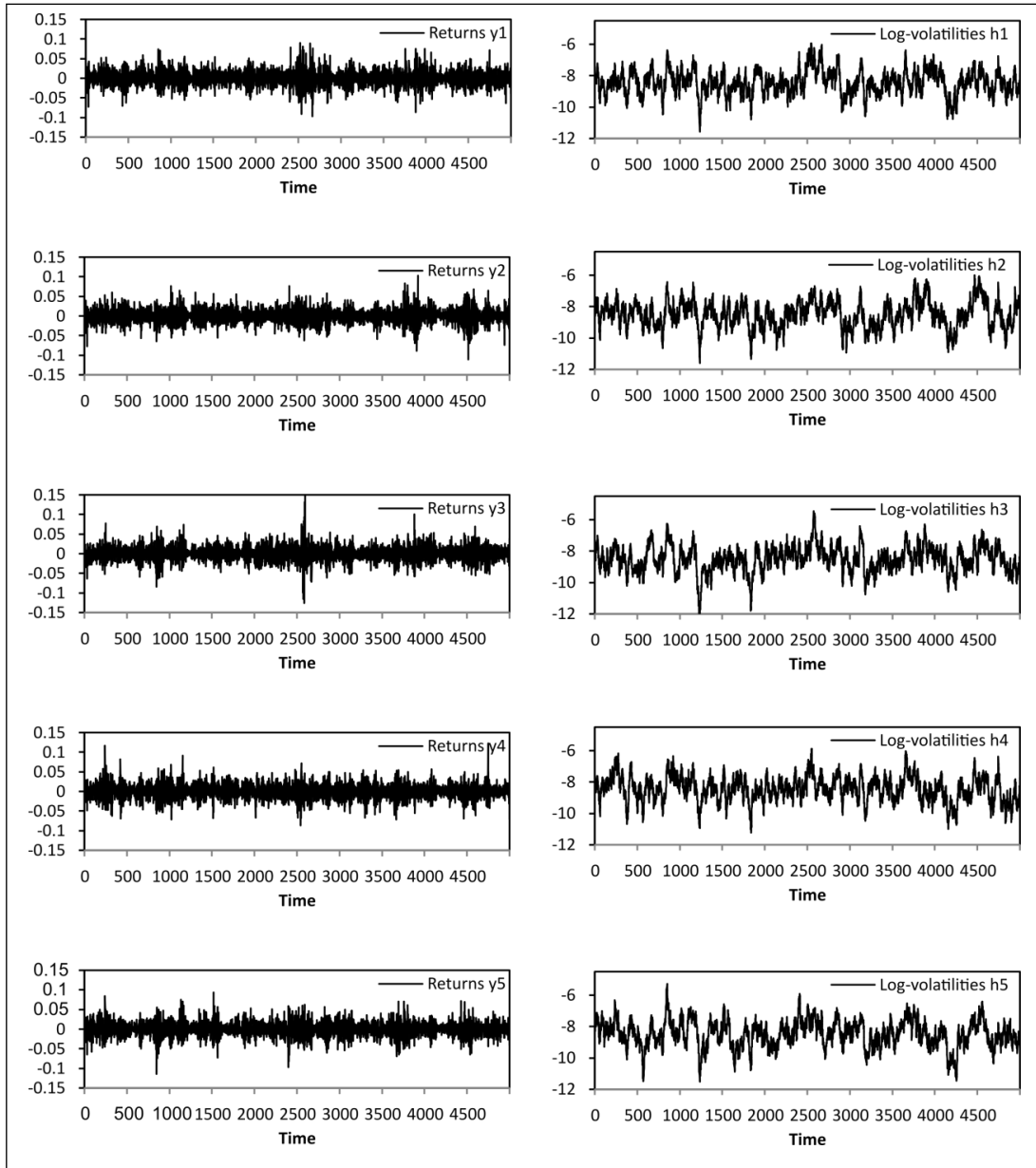


Figure 5.1 : Simulated series based on the static MSV-D model.

Parameter estimation results of the parameters of the log-volatility process, γ_i , ϕ_i , $\sigma_{\eta,i}$ are given in Table 5.1. In Table 5.1, it can be observed that in general posterior means of the parameter estimates are sufficiently close to the true parameter values with 95% intervals containing the true values.

The intervals in the estimates for the persistence parameters, ϕ_i are relatively narrower than the intercept, γ_i and standard deviations of the error term, $\sigma_{\eta,i}$. Higher inefficiency factors are observed in $\sigma_{\eta,i}$ when compared to other parameters followed by ϕ_i (Table 5.1).

Table 5.1 : Static MSV-D model parameter estimation results of $\gamma_i, \varphi_i, \sigma_{\eta,i}$.

	True	i	Mean	95% interval	Inefficiency
γ_i	0.25	1	-0.261	[-0.322,-0.201]	129
		2	-0.242	[-0.313,-0.171]	190
		3	-0.239	[-0.303,-0.175]	206
		4	-0.264	[-0.329,-0.199]	169
		5	-0.241	[-0.304,-0.178]	102
φ_i	0.97	1	0.961	[0.952, 0.971]	365
		2	0.976	[0.967, 0.985]	340
		3	0.966	[0.957, 0.975]	272
		4	0.963	[0.954, 0.972]	314
		5	0.974	[0.965, 0.983]	324
$\sigma_{\eta,i}$	0.2	1	0.208	[0.177, 0.239]	386
		2	0.194	[0.165, 0.223]	351
		3	0.203	[0.171, 0.235]	408
		4	0.188	[0.160, 0.216]	430
		5	0.191	[0.161, 0.221]	381

Parameter estimation results of the correlation matrix entries, $\rho_{\varepsilon,\varepsilon,ij}, \rho_{\varepsilon\eta,ii}, \rho_{\varepsilon\eta,ij}$ are given in Table 5.2 for the static MSV-D model. It can be observed from Table 5.2 that the MSV-D model and its estimation algorithm performs well in estimating the static correlation matrix, \mathbf{P} , with estimations close to the true values which are within 95% intervals.

It is seen from both Table 5.1 and Table 5.2 that the inefficiency factors, the indicator of how the chain mixes, are high when compared to the benchmarks in the literature especially for the parameters $\sigma_{\eta,i}, \rho_{\varepsilon\eta,ii}, \rho_{\varepsilon\eta,ij}, \rho_{\eta\eta,ij}$, with values well above 300. This indicates the requirement of large sample sizes and justifies the sample size choice of $N = 400,000$ which is quite high when compared to examples in the literature. Although the developed Bayesian MCMC estimation algorithm is not highly efficient, it performs well enough for the objectives of this study.

The efficiency of the MCMC algorithm can be altered by implementing alternative sampling techniques such as the multi-move sampler as discussed in (Ishihara & Omori, 2012) without the loss of generality of the developed MCMC algorithm.

Table 5.2 : Static MSV-D model parameter estimation results of $\rho_{\varepsilon\varepsilon,ij}$, $\rho_{\varepsilon\eta,ij}$, $\rho_{\eta\eta,ij}$.

	True	i/j	Estimate	95% interval	Inefficiency
$\rho_{\varepsilon\varepsilon,ij}$	0.6	12	0.581	[0.467, 0.695]	27
		13	0.609	[0.529, 0.689]	41
		14	0.587	[0.477, 0.697]	33
		15	0.580	[0.483, 0.677]	41
		23	0.593	[0.506, 0.680]	24
		31	0.591	[0.509, 0.673]	39
		32	0.612	[0.523, 0.701]	40
		34	0.596	[0.498, 0.694]	28
		35	0.605	[0.485, 0.725]	26
$\rho_{\varepsilon\eta,ii}$	-0.4	11	-0.431	[-0.541, -0.321]	395
		22	-0.412	[-0.552, -0.272]	414
		33	-0.378	[-0.502, -0.254]	425
		44	-0.381	[-0.498, -0.264]	306
		55	-0.385	[-0.514, -0.256]	368
$\rho_{\varepsilon\eta,ij}$	-0.3	12	-0.289	[-0.418, -0.160]	416
		13	-0.271	[-0.387, -0.155]	444
		14	-0.283	[-0.380, -0.186]	332
		15	-0.276	[-0.362, -0.190]	346
		21	-0.329	[-0.411, -0.247]	218
		23	-0.311	[-0.410, -0.212]	338
		24	-0.291	[-0.418, -0.164]	427
		25	-0.275	[-0.356, -0.194]	255
		23	-0.332	[-0.427, -0.237]	431
		31	-0.293	[-0.376, -0.210]	231
		32	-0.280	[-0.411, -0.150]	229
		34	-0.316	[-0.396, -0.236]	373
		35	-0.321	[-0.419, -0.222]	300
		41	-0.329	[-0.434, -0.224]	444
		42	-0.288	[-0.395, -0.181]	289
		43	-0.294	[-0.418, -0.170]	275
		45	-0.318	[-0.399, -0.237]	273
51	-0.325	[-0.432, -0.218]	422		
52	-0.279	[-0.396, -0.162]	276		
53	-0.286	[-0.404, -0.168]	398		
54	-0.267	[-0.364, -0.169]	415		
$\rho_{\eta\eta,ij}$	0.6	11	0.648	[0.545, 0.751]	625
		12	0.622	[0.487, 0.757]	704
		13	0.637	[0.522, 0.754]	451
		14	0.619	[0.487, 0.751]	698
		15	0.594	[0.501, 0.688]	562
		23	0.611	[0.484, 0.738]	717
		31	0.631	[0.536, 0.726]	653
		32	0.589	[0.492, 0.686]	748
		34	0.590	[0.468, 0.712]	469
		35	0.587	[0.495, 0.679]	487
45	0.627	[0.522, 0.732]	721		

The charts in Figure 5.2 shows the estimated and actual log-volatilities for comparison. Log-volatility estimation results for the static MSV-D model are summarized in Table 5.3.

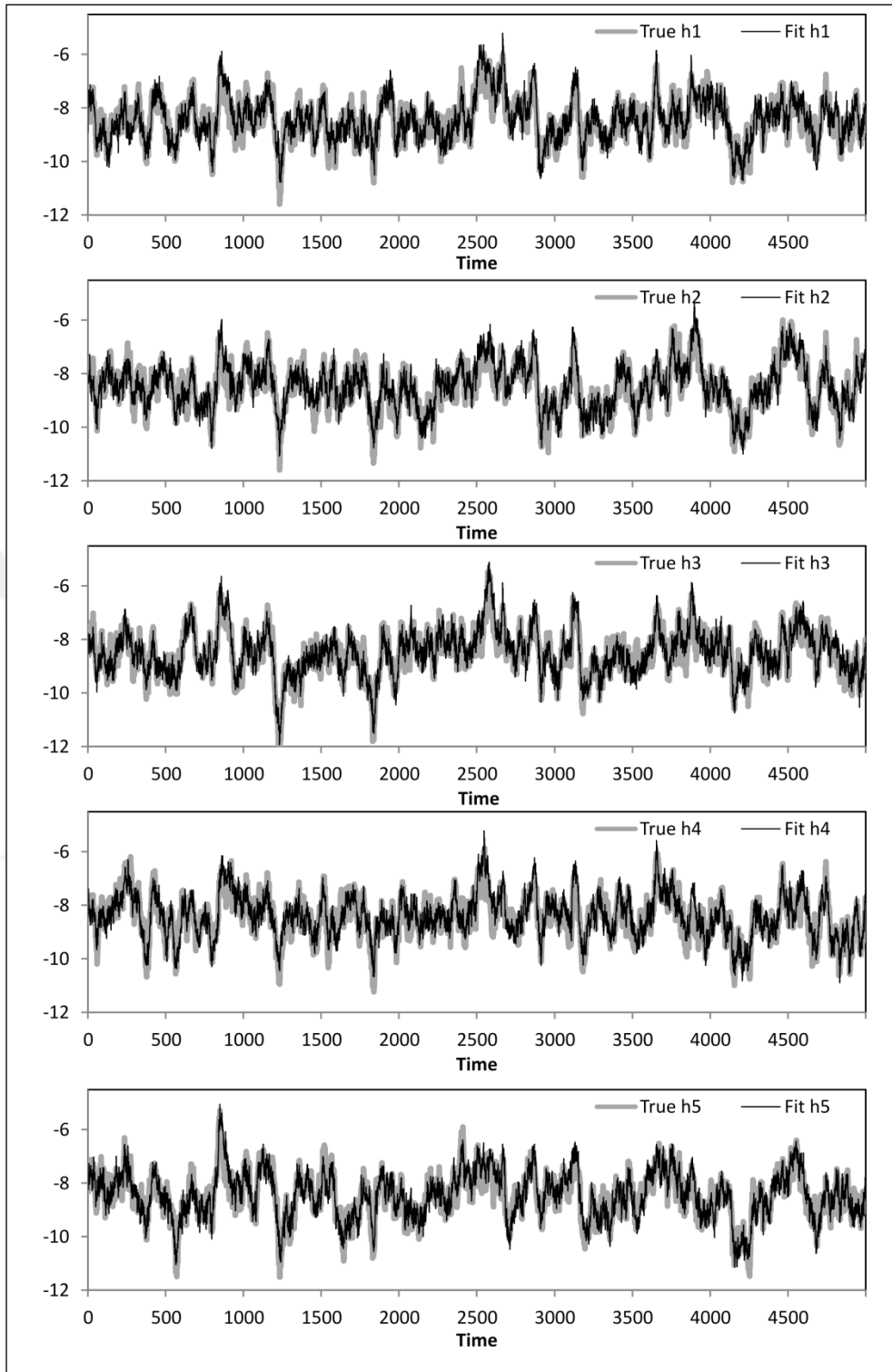


Figure 5.2 : Log-volatility fits for the static MSV-D model.

In the charts in Figure 5.2 the match between the patterns of the estimates and true values are visually seen. In Table 5.3, it can be seen that the estimated means for the log-volatilities and their true values are quite close with acceptable values of RMSE values for all five return series.

Table 5.3 : Static MSV-D model log-volatility estimation results.

Series (i)	True Mean	Estimated Mean	RMSE _h
1	-8.454	-8.424	0.311
2	-8.523	-8.476	0.305
3	-8.515	-8.485	0.302
4	-8.446	-8.393	0.315
5	-8.474	-8.448	0.314

It can be concluded that log-volatility estimates are sufficiently close to the true values and that the proposed MSV-D model, its parameterization and its custom Bayesian MCMC estimation algorithm perform sufficiently well for the static case.

Second simulated data set regarding the assessment of MSV-D model is based on the complete dynamic specification using the proposed MSV-D model.

Figure 5.3 shows the simulated data including the five asset returns and corresponding log-volatility series for the complete dynamic MSV-D model. In the charts of Figure 5.3, volatility clusterings and co-movements of the asset returns and volatilities are observable.

In the complete dynamic MSV-D specification, the correlations are also time varying and in Figure 5.4 some of the correlations between different components indicating various dynamic stylized facts are plotted.

In Figure 5.4, first chart is an example of correlations between the asset returns, second chart is an example of dynamic leverage effect, third chart is an example of dynamic cross-leverage effect and fourth chart is an example of dynamic volatility spillover which are all addressable with the flexible structure of the proposed MSV-D model.

Using the Bayesian MCMC estimation algorithm developed in section 2.2.3.2, estimates of parameters, log-volatilities and correlations are obtained.

Dynamic MSV-D estimation results for the log-volatility process parameters, γ_i , ϕ_i , $\sigma_{\eta,i}$ are given in Table 5.4. It can be observed that the true values are close to the parameter estimates with 95% intervals including the true values.

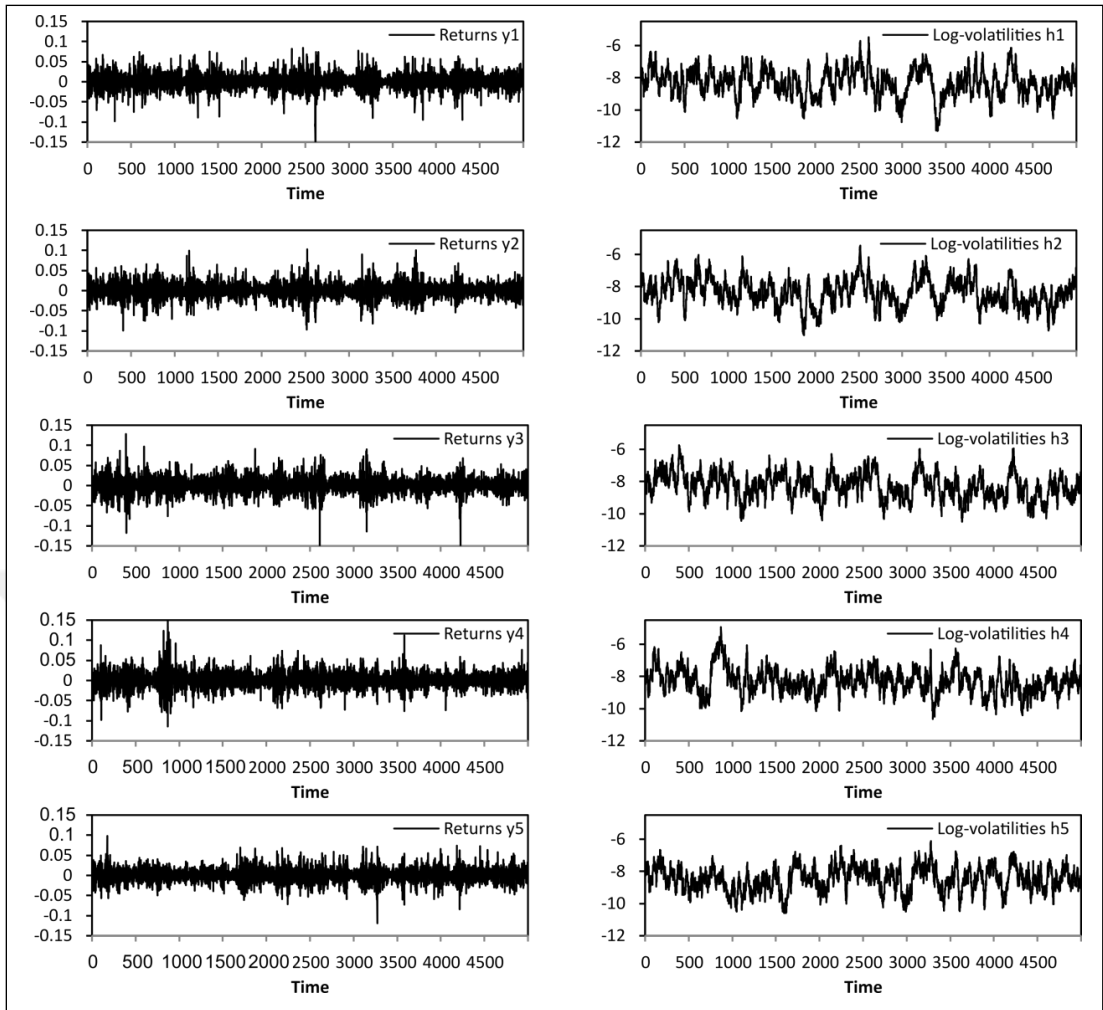


Figure 5.3 : Simulated series based on the dynamic MSV-D model.

Dynamic MSV-D estimation results for parameter δ_i which is the intercept parameter of the AR(1) process driving the dynamic correlations are given in Table 5.5. In Table 5.6, dynamic MSV-D estimation results for the parameters θ_i which is the persistence parameter of the AR(1) process driving the dynamic correlations are given. Table 5.7 shows the dynamic MSV-D estimation results for parameter $\sigma_{\omega,i}$. In Table 5.5, Table 5.6 and Table 5.7, results show that the parameter estimates are sufficiently close to the true values and true values fall into the 95% intervals. The parameter estimates of $\sigma_{\eta,i}$ given in Table 5.4 and $\sigma_{\omega,i}$, in Table 5.7 have relatively large estimation intervals and inefficiency factors are quite high for those parameters. The dynamic MSV-D specification has too many parameters to estimate and dimensionality is high and this result is in fact expected. Obviously larger sample sizes would be better for the dynamic setting.

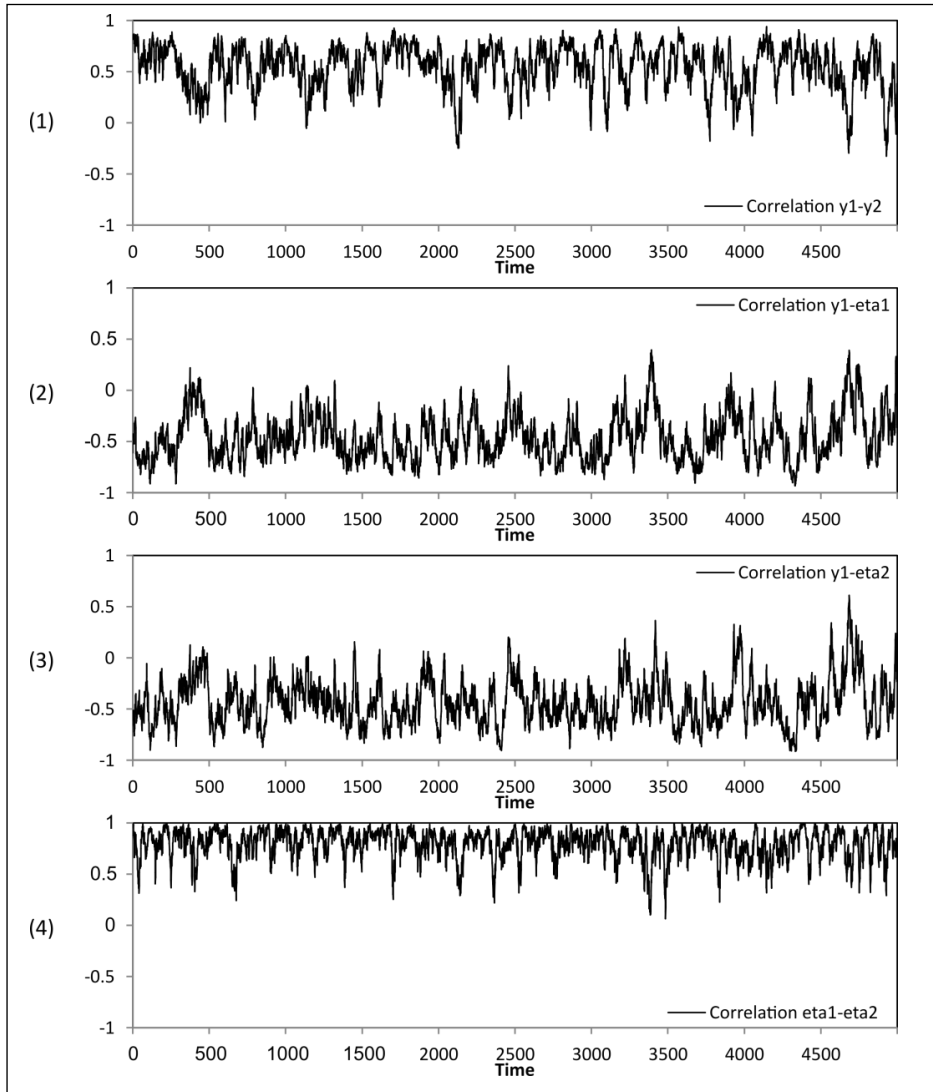


Figure 5.4 : Examples of simulated dynamic correlations based on the dynamic MSV-D model.

Dynamic MSV-D model parameter estimation results of the log-volatilities are given in Table 5.8 where the RMSE values are all at acceptable levels with close means of estimations and true values.

The charts in Figure 5.5 plot the log-volatility estimates and their actual values for all time periods and it can be seen that the log-volatility estimates successfully follow the patterns of the actual values.

Table 5.4 : Dynamic MSV-D model parameter estimation results of $\gamma_i, \varphi_i, \sigma_{\eta,i}$.

	True	i	Mean	95% interval	Inefficiency
γ_i	-0.25	1	-0.267	[-0.397, -0.137]	251
		2	-0.268	[-0.398, -0.138]	270
		3	-0.243	[-0.383, -0.103]	265
		4	-0.256	[-0.376, -0.136]	189
		5	-0.264	[-0.404, -0.124]	259
φ_i	0.97	1	0.979	[0.959, 0.998]	365
		2	0.971	[0.952, 0.991]	340
		3	0.959	[0.938, 0.979]	272
		4	0.961	[0.941, 0.980]	314
		5	0.979	[0.959, 0.998]	324
$\sigma_{\eta,i}$	0.2	1	0.191	[0.122, 0.260]	431
		2	0.215	[0.152, 0.278]	436
		3	0.214	[0.145, 0.283]	422
		4	0.194	[0.124, 0.264]	410
		5	0.197	[0.125, 0.269]	396

Table 5.5 : Dynamic MSV-D model parameter estimation results of δ_i .

True	i	Mean	95% interval	Ineff.	True	i	Mean	95% interval	Ineff.
0.015	1	0.019	[0.013, 0.024]	342	-0.015	24	-0.016	[-0.021, -0.010]	341
0.015	2	0.013	[0.007, 0.018]	338	0.015	25	0.009	[0.004, 0.016]	360
0.015	3	0.017	[0.011, 0.022]	393	0.015	26	0.010	[0.005, 0.017]	329
0.015	4	0.017	[0.012, 0.022]	362	0.015	27	0.016	[0.010, 0.022]	340
0.015	5	0.019	[0.013, 0.024]	331	0.015	28	0.008	[0.002, 0.016]	338
-0.015	6	-0.009	[-0.017, -0.005]	388	0.015	29	0.019	[0.013, 0.025]	332
-0.015	7	-0.017	[-0.022, -0.011]	379	-0.015	30	-0.016	[-0.021, -0.010]	322
-0.015	8	-0.020	[-0.026, -0.014]	339	0.015	31	0.014	[0.008, 0.019]	323
-0.015	9	-0.019	[-0.025, -0.013]	389	0.015	32	0.020	[0.014, 0.025]	327
0.015	10	0.019	[0.013, 0.025]	352	0.015	33	0.021	[0.014, 0.028]	359
0.015	11	0.017	[0.011, 0.022]	395	0.015	34	0.020	[0.014, 0.025]	336
0.015	12	0.021	[0.014, 0.027]	368	0.015	35	0.011	[0.005, 0.016]	351
0.015	13	0.012	[0.006, 0.018]	352	-0.015	36	-0.018	[-0.024, -0.012]	320
0.015	14	0.016	[0.011, 0.020]	322	-0.015	37	-0.019	[-0.024, -0.014]	367
-0.015	15	-0.019	[-0.024, -0.013]	337	-0.015	38	-0.015	[-0.020, -0.009]	329
-0.015	16	-0.021	[-0.028, -0.013]	356	-0.015	39	-0.008	[-0.013, -0.003]	364
-0.015	17	-0.021	[-0.026, -0.014]	401	-0.015	40	-0.019	[-0.024, -0.013]	351
0.015	18	0.019	[0.012, 0.024]	389	-0.015	41	-0.013	[-0.018, -0.007]	386
0.015	19	0.015	[0.009, 0.021]	358	-0.015	42	-0.017	[-0.022, -0.011]	349
0.015	20	0.012	[0.006, 0.018]	368	-0.015	43	-0.009	[-0.016, -0.003]	375
0.015	21	0.009	[0.005, 0.016]	373	-0.015	44	-0.017	[-0.022, -0.012]	403
0.015	22	0.009	[0.004, 0.016]	345	-0.015	45	-0.009	[-0.017, -0.001]	337
-0.015	23	-0.014	[-0.019, -0.008]	383					

Table 5.6 : Dynamic MSV-D model parameter estimation results of θ_i .

True	i	Mean	95% interval	Ineff.	True	i	Mean	95% interval	Ineff.
0.97	1	0.979	[0.962, 0.996]	376	0.97	24	0.975	[0.957, 0.993]	410
	2	0.983	[0.966, 0.999]	356		25	0.983	[0.965, 0.999]	410
	3	0.963	[0.945, 0.980]	394		26	0.960	[0.943, 0.976]	425
	4	0.969	[0.951, 0.986]	395		27	0.976	[0.958, 0.993]	411
	5	0.965	[0.947, 0.982]	389		28	0.970	[0.952, 0.987]	426
	6	0.978	[0.960, 0.995]	356		29	0.958	[0.940, 0.975]	363
	7	0.968	[0.950, 0.985]	423		30	0.965	[0.947, 0.982]	352
	8	0.965	[0.948, 0.981]	367		31	0.964	[0.947, 0.981]	417
	9	0.976	[0.959, 0.992]	416		32	0.963	[0.944, 0.981]	357
	10	0.959	[0.941, 0.976]	356		33	0.980	[0.962, 0.997]	393
	11	0.964	[0.946, 0.981]	397		34	0.967	[0.949, 0.984]	362
	12	0.976	[0.957, 0.994]	364		35	0.969	[0.951, 0.987]	372
	13	0.96	[0.942, 0.978]	372		36	0.978	[0.959, 0.996]	358
	14	0.972	[0.954, 0.989]	373		37	0.976	[0.959, 0.992]	372
	15	0.966	[0.947, 0.984]	361		38	0.963	[0.944, 0.981]	404
	16	0.961	[0.943, 0.979]	351		39	0.975	[0.958, 0.992]	355
	17	0.981	[0.962, 0.998]	396		40	0.969	[0.951, 0.987]	407
	18	0.958	[0.940, 0.976]	397		41	0.972	[0.956, 0.989]	389
	19	0.973	[0.954, 0.991]	425		42	0.978	[0.960, 0.995]	359
	20	0.978	[0.961, 0.994]	376		43	0.973	[0.954, 0.992]	416
	21	0.956	[0.937, 0.974]	364		44	0.956	[0.939, 0.973]	388
	22	0.969	[0.951, 0.986]	407		45	0.969	[0.952, 0.986]	369
	23	0.979	[0.961, 0.996]	374					

Table 5.7 : Dynamic MSV-D model parameter estimation results of $\sigma_{\omega,i}$.

True	i	Mean	95% interval	Ineff.	True	i	Mean	95% interval	Ineff.
0.01	1	0.008	[0.0009, 0.0154]	548	0.01	24	0.014	[0.0008, 0.0201]	468
	2	0.018	[0.0005, 0.0253]	534		25	0.021	[0.0004, 0.0275]	466
	3	0.021	[0.0008, 0.0281]	478		26	0.013	[0.0005, 0.0205]	544
	4	0.014	[0.0004, 0.0210]	506		27	0.007	[0.0004, 0.0141]	475
	5	0.011	[0.0005, 0.0178]	502		28	0.022	[0.0004, 0.0282]	552
	6	0.008	[0.0004, 0.0151]	456		29	0.013	[0.0009, 0.0205]	489
	7	0.015	[0.0008, 0.0220]	468		30	0.022	[0.0008, 0.0284]	477
	8	0.018	[0.0005, 0.0242]	536		31	0.010	[0.0005, 0.0162]	444
	9	0.012	[0.0008, 0.0191]	468		32	0.015	[0.0006, 0.0213]	534
	10	0.018	[0.0004, 0.0242]	481		33	0.014	[0.0007, 0.0211]	496
	11	0.017	[0.0005, 0.0243]	558		34	0.013	[0.0005, 0.0196]	432
	12	0.019	[0.0006, 0.0256]	540		35	0.019	[0.0009, 0.0251]	499
	13	0.011	[0.0007, 0.0171]	460		36	0.007	[0.0009, 0.0132]	433
	14	0.017	[0.0005, 0.0239]	474		37	0.015	[0.0005, 0.0223]	556
	15	0.007	[0.0005, 0.0136]	535		38	0.013	[0.0006, 0.0195]	431
	16	0.011	[0.0009, 0.0177]	537		39	0.020	[0.0006, 0.0275]	534
	17	0.017	[0.0004, 0.0232]	479		40	0.016	[0.0005, 0.0220]	520
	18	0.007	[0.0004, 0.0137]	452		41	0.020	[0.0008, 0.0261]	514
	19	0.020	[0.0008, 0.0261]	512		42	0.013	[0.0004, 0.0203]	511
	20	0.011	[0.0008, 0.0165]	462		43	0.017	[0.0004, 0.0239]	495
	21	0.009	[0.0005, 0.0164]	460		44	0.017	[0.0007, 0.0231]	480
	22	0.015	[0.0004, 0.0224]	513		45	0.008	[0.0005, 0.0152]	431
	23	0.011	[0.0009, 0.0172]	541					

Table 5.8 : Dynamic MSV-D model log-volatility estimation results.

i	True Mean	Estimated Mean	RMSE
1	-8.324	-8.322	0.281
2	-8.371	-8.368	0.273
3	-8.281	-8.284	0.275
4	-8.287	-8.284	0.276
5	-8.483	-8.477	0.280

Dynamic MSV-D model estimation results of the dynamic correlations are shown in Table 5.9. Mean of the actual correlation coefficients and the estimated values are quite close with acceptable RMSE values (Table 5.9). In Figure 5.6, actual correlation coefficients and their estimates for all time periods for some of the correlation components representing dynamic correlations between asset returns (1), dynamic leverage effect (2), dynamic cross-leverage effect (3) and dynamic volatility spillover effects (4) are plotted. In all charts it can be seen that estimations follow the actual patterns successfully.

Table 5.9 : Dynamic MSV-D model estimation results of the dynamic correlations,

		$\rho_{\varepsilon,ij,t}, \rho_{\varepsilon,ii,t}, \rho_{\varepsilon,ij,t}$							
	ij	True Mean	Est. Mean	RMSE		ij	True Mean	Est. Mean	RMSE
$\rho_{\varepsilon,ij,t}$	12	0.545	0.513	0.084	$\rho_{\varepsilon,ii,t}$	11	-0.457	-0.438	0.085
	13	0.560	0.528	0.083		12	-0.434	-0.416	0.085
	14	0.516	0.488	0.091		13	-0.457	-0.441	0.084
	15	0.531	0.504	0.085		14	-0.394	-0.385	0.086
	23	0.545	0.516	0.082		15	-0.289	-0.285	0.094
	24	0.530	0.506	0.084		21	-0.478	-0.458	0.085
	25	0.516	0.494	0.088		22	-0.501	-0.480	0.082
	34	0.522	0.493	0.083		23	-0.435	-0.421	0.087
	35	0.559	0.533	0.079		24	-0.413	-0.404	0.087
	45	0.526	0.502	0.082		25	-0.310	-0.307	0.092
$\rho_{\eta,ij,t}$	12	0.797	0.768	0.073	31	-0.517	-0.494	0.082	
	13	0.439	0.424	0.088	32	-0.505	-0.483	0.081	
	14	0.404	0.393	0.086	33	-0.459	-0.445	0.083	
	15	0.354	0.348	0.088	34	-0.408	-0.395	0.085	
	23	0.456	0.441	0.074	35	-0.359	-0.355	0.088	
	24	0.426	0.414	0.084	41	-0.461	-0.443	0.087	
	25	0.365	0.361	0.087	42	-0.467	-0.449	0.086	
	34	0.379	0.371	0.077	43	-0.421	-0.406	0.085	
	35	0.355	0.351	0.086	44	-0.396	-0.387	0.084	
	45	0.297	0.296	0.090	45	-0.325	-0.322	0.086	
				51	-0.491	-0.474	0.092		
				52	-0.483	-0.465	0.083		
				53	-0.476	-0.460	0.085		
				54	-0.416	-0.409	0.084		
				55	-0.379	-0.376	0.086		

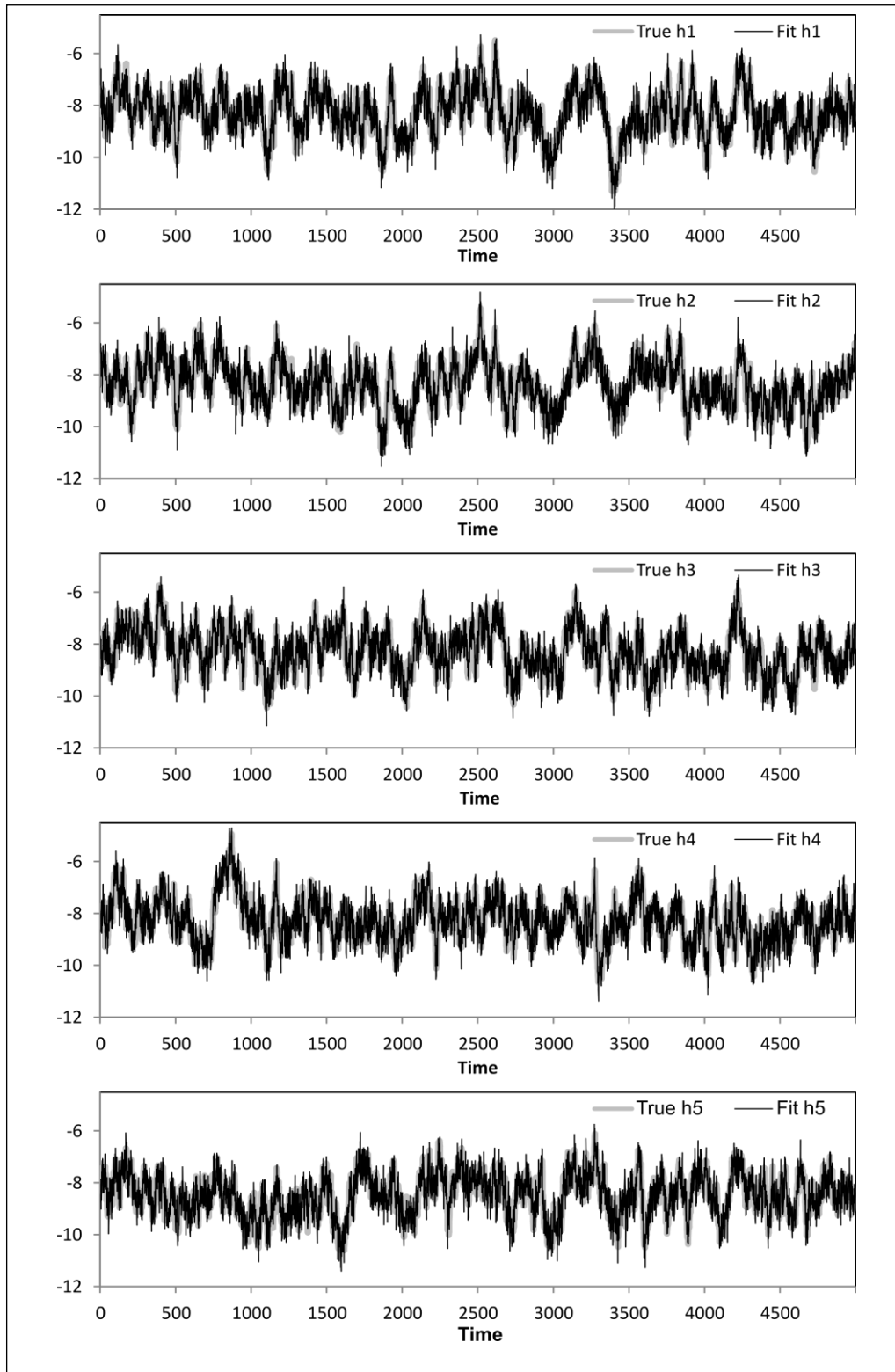


Figure 5.5 : Log-volatility fits for the dynamic MSV-D model.

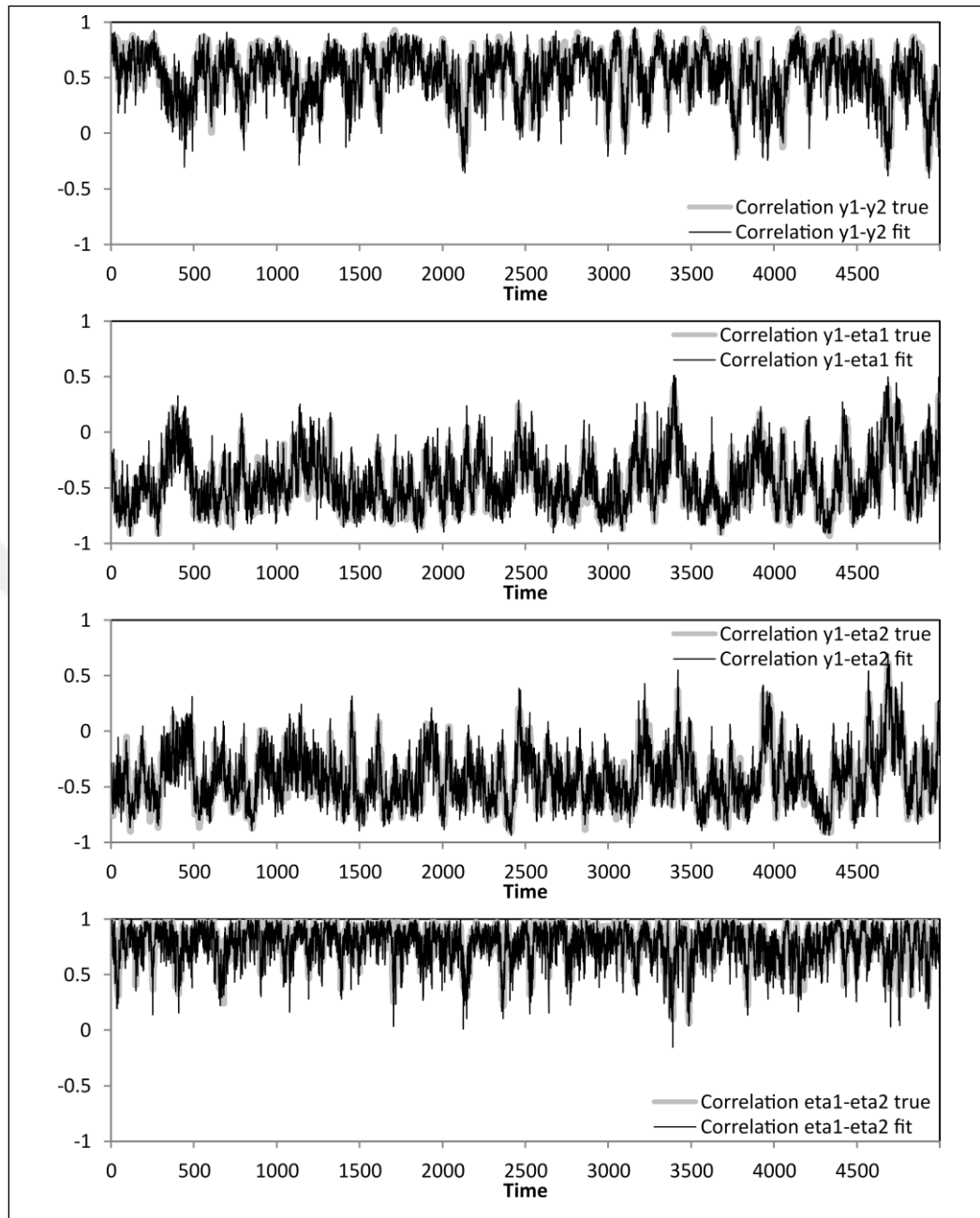


Figure 5.6 : Dynamic correlation fits for the dynamic MSV-D model.

In the complete dynamic MSV-D model, one of the important drawbacks is the significantly increased number of parameters due to the additional AR(1) processes driving the correlation coefficients each having three parameters and total number of parameters increase polynomially in dimension. This reflects into the convergence of the MCMC algorithms with significantly increased inefficiency factors which are indicators of slow convergence.

5.2 MSV-D Model on Empirical Data

A complete dynamic MSV-D model specification is applied to the return series of S&P500 index, IBM and Intel (INTC) stock returns as described in section 4.2.

The return series are plotted in Figure 5.7 where the volatility clusterings and co-movements of asset returns are visible.

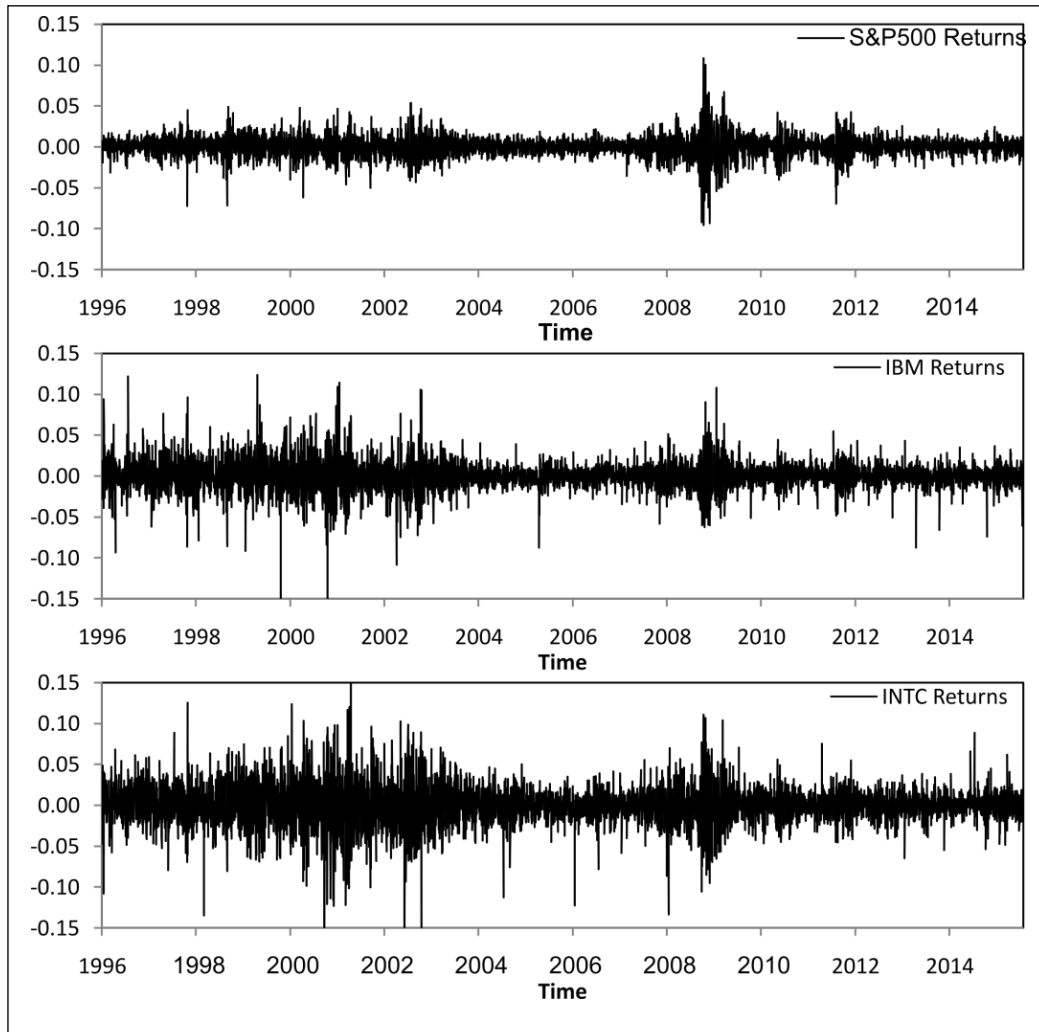


Figure 5.7 : Return series of S&P500, IBM and Intel (INTC).

MSV-D model parameter estimation results of the log-volatility process parameters, $\gamma_i, \varphi_i, \sigma_{\eta,i}$ are given in Table 5.10. And MSV-D model parameter estimation results for the correlation process parameters δ_i, θ_i and $\sigma_{\omega,i}$ are given in Table 5.11, Table 5.12 and Table 5.13 respectively.

Table 5.10 : MSV-D model parameter estimation results of γ_i , φ_i , $\sigma_{\eta,i}$ on S&P500, IBM and Intel(INTC) returns.

	i	Mean	95% interval	Inefficiency
γ_i	1	-0.149	[-0.079, -0.220]	209
	2	-0.175	[-0.088, -0.261]	215
	3	-0.074	[-0.013, -0.138]	204
φ_i	1	0.984	[0.974, 0.993]	322
	2	0.979	[0.967, 0.991]	311
	3	0.990	[0.987, 0.999]	251
$\sigma_{\eta,i}$	1	0.187	[0.125, 0.248]	384
	2	0.203	[0.144, 0.262]	405
	3	0.136	[0.081, 0.194]	354

Table 5.11 : MSV-D model parameter estimation results of δ_i on S&P500, IBM and Intel(INTC) returns.

	i	Estimate	95% interval	Inefficiency
δ_i	1	0.0116	[0.009, 0.014]	208
	2	0.0096	[0.007, 0.012]	166
	3	0.0164	[0.012, 0.020]	169
	4	-0.0156	[-0.011, -0.021]	283
	5	-0.0119	[-0.008, -0.015]	156
	6	0.0128	[0.009, 0.016]	207
	7	0.0191	[0.014, 0.024]	169
	8	0.0074	[0.005, 0.009]	152
	9	-0.0164	[-0.012, -0.021]	257
	10	0.0694	[0.051, 0.087]	173
	11	0.0248	[0.019, 0.030]	287
	12	0.0224	[0.016, 0.028]	213
	13	-0.0235	[-0.016, -0.031]	245
	14	-0.0154	[-0.012, -0.018]	157
	15	-0.0174	[-0.013, -0.022]	261

Table 5.12 : MSV-D model parameter estimation results of θ_i on S&P500, IBM and Intel(INTC) returns.

	i	Estimate	95% interval	Inefficiency
θ_i	1	0.961	[0.942, 0.978]	376
	2	0.960	[0.942, 0.978]	318
	3	0.963	[0.946, 0.979]	375
	4	0.968	[0.951, 0.988]	371
	5	0.972	[0.954, 0.990]	354
	6	0.958	[0.940, 0.974]	349
	7	0.964	[0.947, 0.981]	279
	8	0.958	[0.940, 0.975]	269
	9	0.965	[0.948, 0.982]	290
	10	0.952	[0.934, 0.968]	274
	11	0.958	[0.941, 0.976]	262
	12	0.965	[0.947, 0.983]	275
	13	0.965	[0.948, 0.982]	376
	14	0.968	[0.951, 0.985]	281
	15	0.965	[0.947, 0.983]	372

Table 5.13 : MSV-D model parameter estimation results of $\sigma_{\omega,i}$ on S&P500, IBM and Intel(INTC) returns.

	i	Estimate	95% interval	Inefficiency
$\sigma_{\omega,i}$	1	0.008	[0.004, 0.011]	348
	2	0.007	[0.004, 0.010]	402
	3	0.019	[0.011, 0.027]	384
	4	0.008	[0.005, 0.012]	390
	5	0.008	[0.004, 0.011]	336
	6	0.007	[0.004, 0.011]	404
	7	0.021	[0.011, 0.031]	333
	8	0.006	[0.003, 0.009]	366
	9	0.010	[0.005, 0.014]	336
	10	0.004	[0.002, 0.007]	324
	11	0.006	[0.003, 0.009]	379
	12	0.006	[0.003, 0.008]	404
	13	0.008	[0.005, 0.012]	320
	14	0.009	[0.005, 0.013]	325
	15	0.010	[0.006, 0.014]	379

To compare the log-volatility estimates obtained from the fitted MSV-D model a DCC-EGARCH model is also fitted to the data. DCC-EGARCH model parameter estimation results are given in Table 5.14.

Table 5.14 : GARCH model parameter estimates on S&P500, IBM and Intel(INTC) returns.

	<i>i</i>	Estimate	Standard Error
ω	1	-0.210	0.002
	2	-0.119	0.011
	3	-0.084	0.004
α	1	-0.128	0.008
	2	-0.055	0.013
	3	-0.027	0.013
β	1	0.977	0.000
	2	0.985	0.001
	3	0.988	0.001
γ	1	0.132	0.008
	2	0.169	0.024
	3	0.114	0.005
a		0.036	0.009
b		0.917	0.026

The fitted DCC-GARCH model can address static leverage effect with the parameters ω_i and γ_i and allows dynamic correlation between asset returns with parameters a and b .

The logarithms of the squared returns are considered as a reference for comparing the log-volatility estimates from the MSV-D with the DCC-EGARCH model in terms of the RMSEs of the log-volatility estimates. Log-volatility estimates from the MSV-D model and DCC-EGARCH model are compared in Table 5.15 and plotted in Figure 5.8 with the logarithms of squared returns as reference.

Table 5.15 : Log-volatility estimates MSV-D vs. GARCH on S&P500, IBM and Intel(INTC) returns.

i	MSV-D RMSE	GARCH RMSE
1	2.558	2.731
2	2.434	2.713
3	2.450	2.692

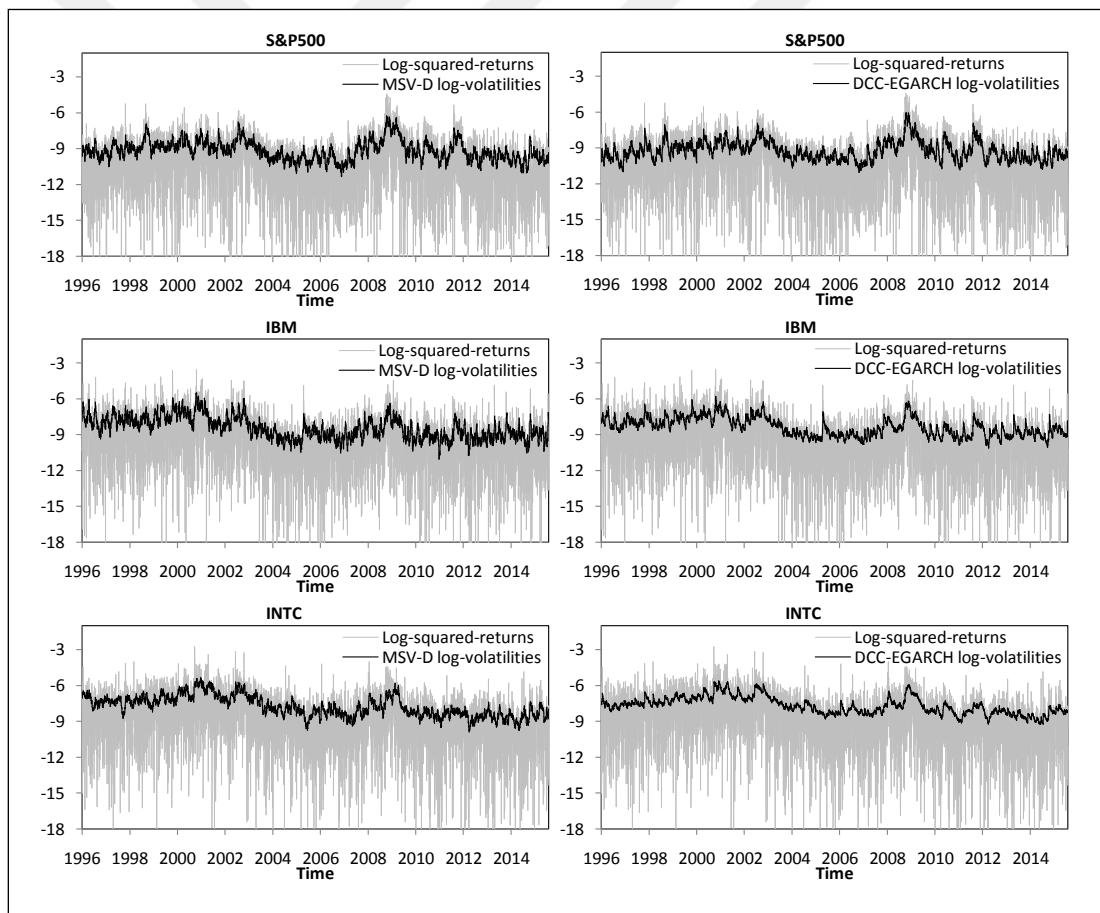


Figure 5.8 : Log-volatility estimates of S&P500, IBM and Intel (INTC).

In Table 5.15, it can be seen that the MSV-D model provides lower RMSE values than the DCC-EGARCH model indicating better performance in capturing the patterns. Both models follows the general patterns of the reference (i.e logarithm of

squared returns) but MSV-D model has a richer and better fit as seen from the charts in Figure 5.8.

Both the DCC-EGARCH model and MSV-D model produce estimates for the dynamic correlations between the asset returns as shown in Figure 5.9. Almost similar patterns are captured for the dynamic correlations between asset returns in both models.

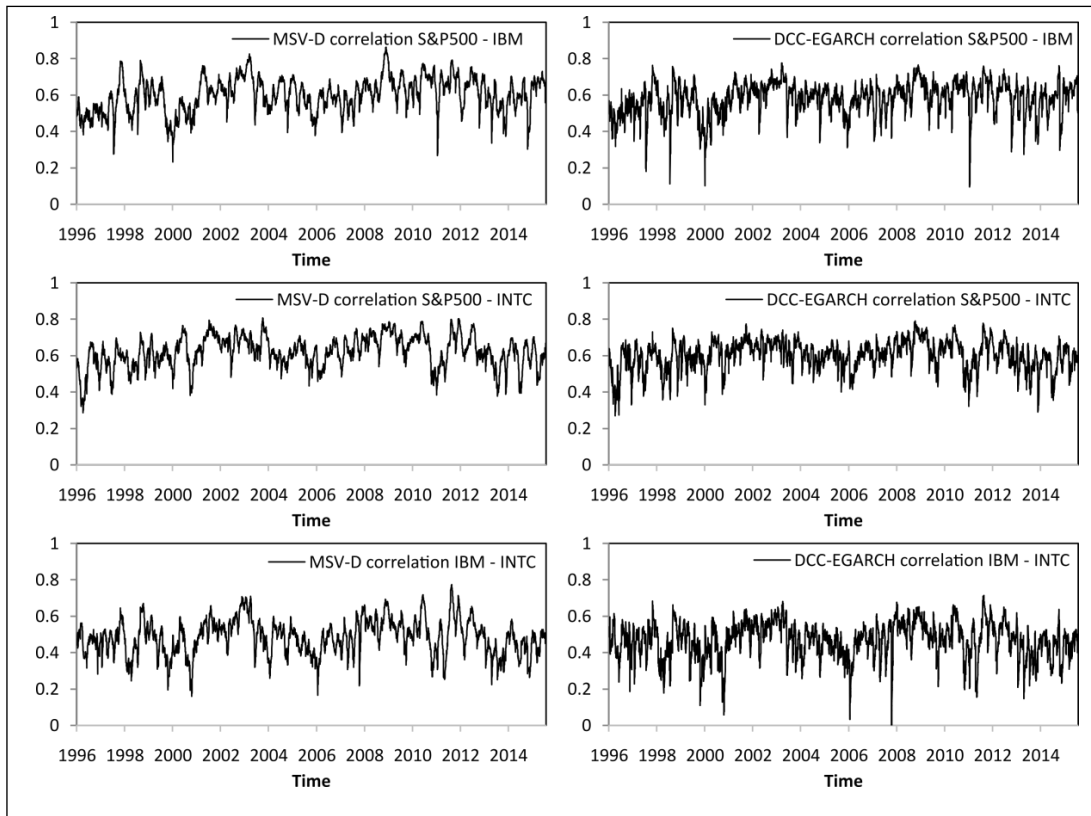


Figure 5.9 : Correlations between returns of S&P500, IBM and Intel (INTC).

A main difference between the DCC-EGARCH and MSV-D model is the treatment of the leverage effects. DCC-EGARCH model addresses the leverage effect through the parameters ω_i and γ_i in a static way whereas the MSV-D model produces time varying correlations between assets and their volatility process errors. Furthermore, MSV-D model produces dynamic cross-leverage and dynamic volatility spillover estimates which are not available in DCC-GARCH and any other volatility models. In Figure 5.10, examples of dynamic correlation estimates produced by the MSV-D model are given. First chart in Figure 5.9 is an example of dynamic leverage effect, second chart is an example of dynamic cross-leverage effect and third chart is an example of dynamic volatility spillover.

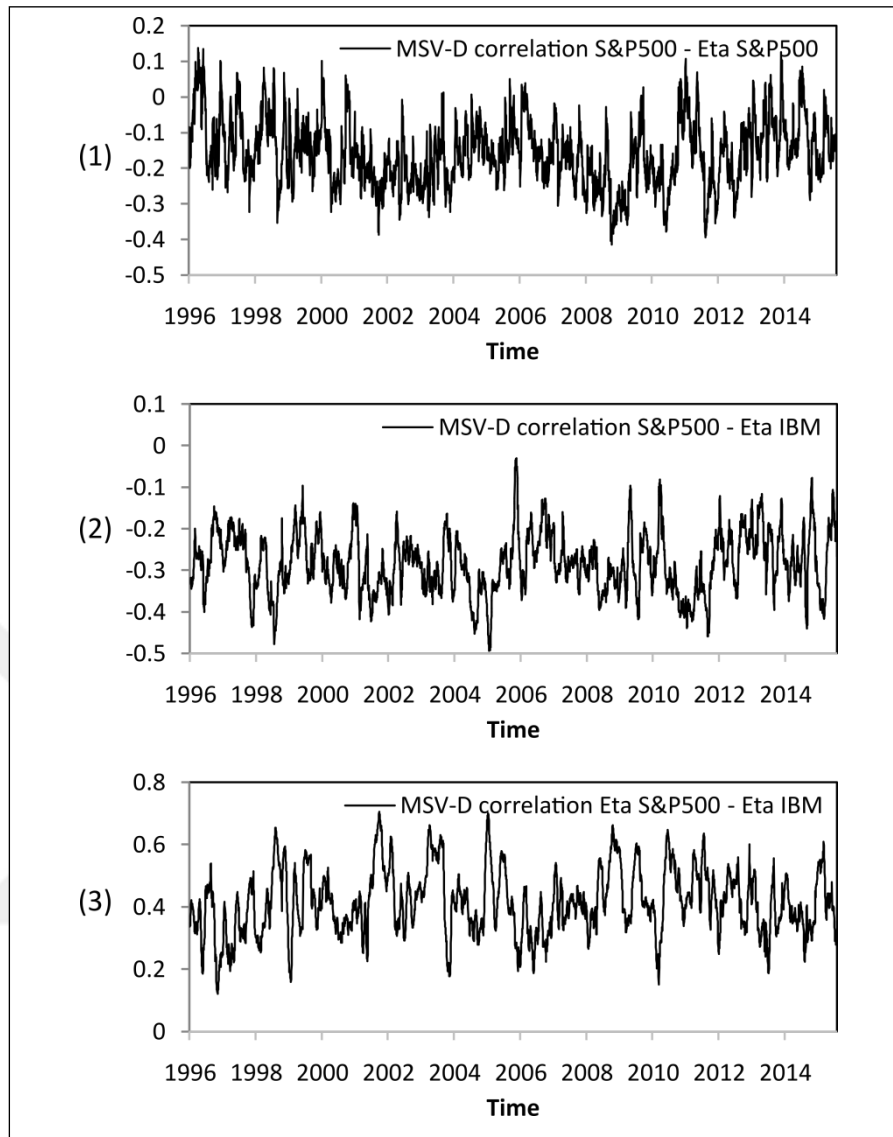


Figure 5.10 : Dynamic leverage, cross-leverage and volatility spillover estimates.

5.3 Comparative Simulations for Estimation Algorithms

In this section the results of the simulation study described in section 4.3 for comparing the proposed SGI and MCMC based estimation algorithms and computational assessment of GPU implementation is presented.

Table 5.16 shows the comparison of SGI and MCMC based estimation algorithms in terms of accuracy for different settings in terms of the state space dimension and accuracy level which is the sample size for the MCMC based algorithms and level of integration formula for the SGI based algorithms. In Table 5.16, the statistics used for the accuracy is the RMSE computed by equation 4.12. Since it is too much time

consuming filtering with MCMC based methods are not included except one measurement at dimension $p = 3$ and accuracy level $N = 400,000$.

Table 5.16 : Accuracy comparison of SGI and MCMC based estimation algorithms.

	Dimension (p)	Accuracy Level (N/l)	Filtering RMSE	Smoothing RMSE	One-step Prediction RMSE	
MCMC	1	50K		0.254	0.420	
		100K		0.190	0.319	
		200K		0.148	0.248	
		400K		0.131	0.213	
	2	50K			0.300	0.449
		100K			0.236	0.353
		200K			0.191	0.269
		400K			0.164	0.230
	3	50K			0.357	0.482
		100K			0.277	0.374
		200K			0.213	0.287
		400K	0.203	0.182	0.246	
SGI	1	4	0.632	0.607	0.675	
		5	0.408	0.395	0.455	
		6	0.255	0.228	0.297	
		7	0.161	0.129	0.203	
	2	4	0.740	0.710	0.779	
		5	0.465	0.439	0.504	
		6	0.286	0.258	0.330	
7		0.179	0.162	0.224		
3	4	0.833	0.797	0.877		
	5	0.524	0.508	0.568		
	6	0.320	0.296	0.361		
	7	0.199	0.177	0.239		

In Figure 5.11 RMSEs of log-volatility smoothing estimates in different dimensions and accuracy levels for both approaches are plotted.

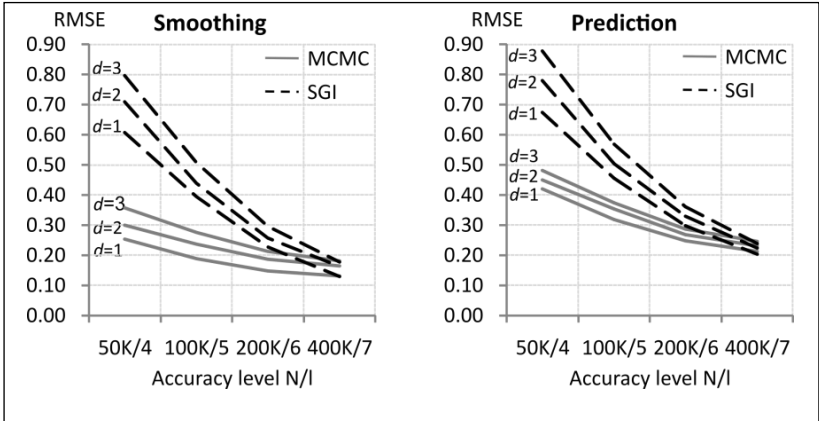


Figure 5.11 : Accuracy comparison of SGI and MCMC based estimation algorithms.

In SGI based estimation algorithms error decreases faster with the increasing accuracy level l where at each level l number of points in the integration formula almost doubles for the trapezoid rule. In MCMC based estimation algorithms doubling the sample size does not provide the same pace of decrease obtained by SGI based algorithms. This is actually an expected result of MCMC being a probabilistic method as discussed before. It is clear that it is easier to control the error with SGI based algorithms. For the current simulation settings, as seen in Table 5.16, $l = 7$ SGI estimation algorithms' accuracy surpass the sample size $N = 400,000$ MCMC estimation algorithms' accuracy in filtering smoothing and prediction problems.

Increasing dimension affects the accuracy of SGI based algorithms more than the MCMC methods. Theoretically MCMC methods are not affected by dimensionality, but a negative effect of increasing dimension, although not too large, on accuracy in MCMC based methods is observed in the simulation study. One of the reasons for this result can be the correlated samples issue and lower acceptance rates in Metropolis-Hastings steps becoming more severe in higher dimensions resulting in higher inefficiency and requiring larger samples.

Table 5.17 shows the parameter estimation results of SGI with level $l=7$ and MCMC with sample size $N = 400,000$ for the three dimensional case.

Table 5.17 : Parameter estimation accuracy comparison of the SGI and MCMC based algorithms.

		SGI $l=7$			MCMC $N=400K$		
		i	True	Estimate	RMSE	Estimate	RMSE
γ_i	-0.25	1	-0.25	-0.237	0.072	-0.260	0.067
		2	-0.25	-0.248	0.029	-0.261	0.071
		3	-0.25	-0.245	0.034	-0.231	0.096
φ_i	0.95	1	0.95	0.947	0.024	0.956	0.069
		2	0.95	0.952	0.016	0.959	0.065
		3	0.95	0.946	0.026	0.961	0.071
σ_{η}	0.02	1	0.02	0.209	0.045	0.206	0.040
		2	0.02	0.191	0.042	0.207	0.054
		3	0.02	0.205	0.037	0.214	0.067
ρ_{ij}	0.6	12	0.6	0.605	0.053	0.592	0.059
		13	0.6	0.594	0.041	0.590	0.073
		23	0.6	0.606	0.039	0.592	0.058

In Table 5.17, it can be seen that SGI based parameter estimation algorithm and MCMC based parameter estimation algorithm produce close estimates to the actual values of the parameters with lower RMSEs in SGI based estimation algorithm in most of the parameters except $\sigma_{\eta,1}$.

The estimation results shows that, SGI based estimation algorithms perform as well as the MCMC based estimation algorithms in terms of accuracy and can be considered as an alternative method with its better convergence and error control properties.

In Table 5.18, execution times of the SGI based filtering and MCMC based smoothing algorithms with serial CPU and parallel GPU implementations in seconds and calculated speed up values are shown. Filtering algorithm is the base algorithm which is used by all other estimation algorithms in SGI approach and similarly smoothing algorithm is the base algorithm for the MCMC approach as discussed in section 3.1 and that is why the acceleration comparisons are made on these algorithms in Table 5.18.

Table 5.18 : Execution times of SGI and MCMC based estimation algorithms.

Dimension	Accuracy Level	Serial Time		GPU Accelerated Time		Speed Up	
		MCMC	SGI	MCMC	SGI	MCMC	SGI
		Smoothing	Filtering	Smoothing	Filtering	Smoothing	Filtering
1	50K/4	296.51	1.14	18.60	0.05	15.94	24.95
	100K/5	596.96	4.86	38.17	0.20	15.64	24.79
	200K/6	1,206.39	20.07	79.32	0.82	15.21	24.45
	400K/7	2,394.89	81.56	161.06	3.38	14.87	24.13
2	50K/4	442.39	12.14	34.48	0.53	12.83	22.88
	100K/5	892.17	84.15	71.60	3.73	12.46	22.59
	200K/6	1,796.57	521.06	146.78	23.56	12.24	22.12
	400K/7	3,603.20	2,990.38	296.56	137.30	12.15	21.78
3	50K/4	581.65	62.30	65.43	3.16	8.89	19.73
	100K/5	1,186.06	623.00	135.70	31.98	8.74	19.48
	200K/6	2,393.78	5,292.06	275.46	274.91	8.69	19.25
	400K/7	4,798.50	40,071.01	554.74	2128.04	8.65	18.83

Figure 5.12 illustrates the serial and GPU accelerated execution times of the SGI based filtering and MCMC based smoothing algorithms for different dimensions and accuracy levels. Time axes are in logarithmic scale in the charts of Figure 5.12.

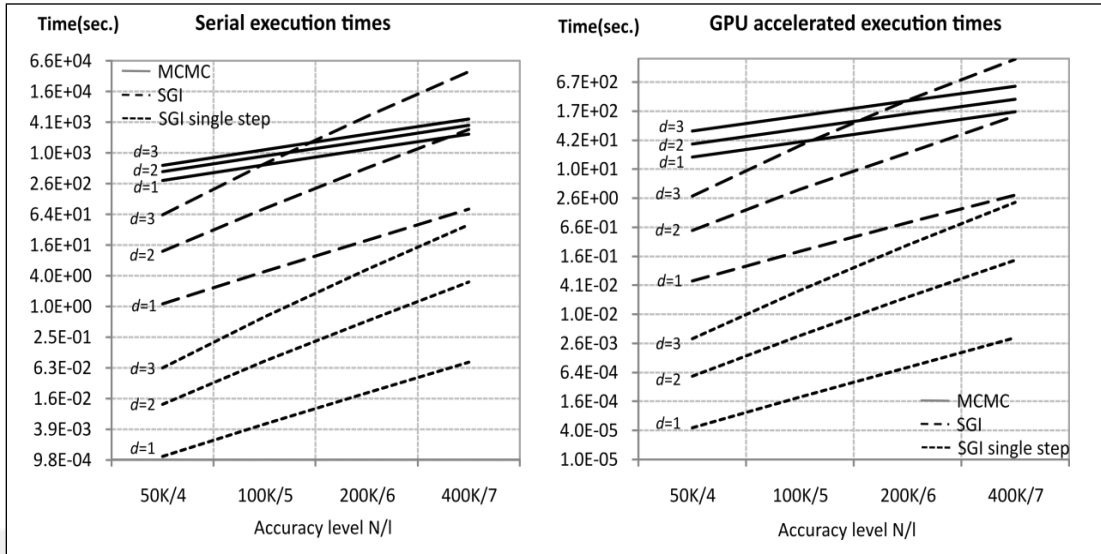


Figure 5.12 : Execution times of serial and GPU accelerated estimation algorithms.

SGI based filtering algorithm is a sequential algorithm as discussed in section 3 and its computational burden can be split across time periods so single step filtering times are included in Figure 5.12 for better comparison.

It can be seen that computational time increases faster when the accuracy level and dimension increases in SGI based algorithms which are actually costs of faster error decrease discussed previously. Increasing dimension and accuracy levels significantly affects the SGI based algorithms and dimensions higher than 5 and accuracy levels above 7 become prohibitive for SGI based algorithms run on commodity computers in serial setting in batch mode. However, one of the advantage of the SGI based algorithms is their sequential structure which allows them compete with MCMC methods in a practical application where single time step performance is critical. Single step serial execution times are well below the serial execution times of MCMC based algorithms as seen in Figure 5.12 and there is room for additional dimensions for the SGI based algorithms in single-step setting where SGI based algorithms performs better than MCMC based algorithms.

Parallelization approaches described in section 3.2 and section 3.3 work well on GPU implementations as seen in Table 5.18 with significantly decreased execution times and speed up values between 18 and 25. Figure 5.13 shows the achieved speed ups for the MCMC smoothing and SGI filtering algorithm implementations with GPU.

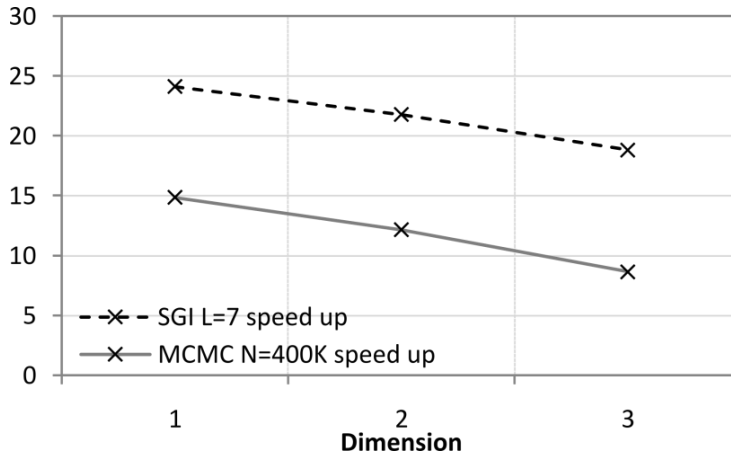


Figure 5.13 : Speed up by dimension in SGI and MCMC based algorithms.

In Figure 5.13 it is observed that speed ups obtained for the SGI based estimation algorithms are higher. In the parallelization approaches given in section 3.2 and section 3.3, processes to be executed in parallel consist of complicated operations for a typical GPU thread but it works with a possible loss of efficiency. In the MCMC based algorithms, operations assigned to a parallel process are more complicated than the SGI based algorithms. In the MCMC based algorithms, in addition to the density function evaluations, random number generation and sampling from certain distributions are required for each process which is one of the reasons for lower speed up values for the MCMC based algorithms.

It is also noteworthy that although provided significant decrease in execution times and made it possible to execute most of the analysis in the study, single GPU quickly becomes overloaded with the size (i.e. accuracy levels and number of time periods) of a typical problem and theoretical speed up values or speed up limits to be tested are beyond the computational resources provided by a single GPU. However the achieved speed up values and decreased execution times are quite promising for larger scale computational settings.

5.4 SGI Based Estimation Algorithms on Empirical Data

In this section, the proposed sparse grid integration method is applied to the foreign-exchange rate series of Euro(EUR)/Turkish Lira(TRL) and US Dollar(USD)/Turkish Lira(TRL) composed of 3669 observations to fit a MSV-B model for illustrating the proposed SGI based estimation approach on real data. For comparison purposes same

MSV-B model is fitted using the MCMC based algorithm and a CCC-GARCH model is fitted to the data as described in section 4.4.

Return series of two foreign exchange rates are shown in Figure 5.14 where the co-movement of the returns is visible with volatility clusterings.

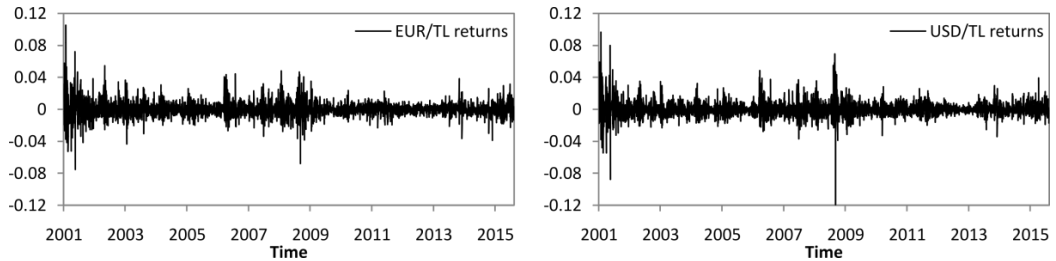


Figure 5.14 : Return series of EUR/TL and USD/TL.

MSV-B model parameter estimation results obtained by the SGI based and the MCMC based estimation algorithms are given in Table 5.19 and CCC-GARCH model parameter estimation results are given in Table 5.20.

Table 5.19 : MSV-B parameter estimation results for EUR/TL and USD/TL.

	i	MSV-B SGI		MSV-B MCMC	
		Mean	Std. Error	Mean	Std. Error
γ_i	1	-0.289	0.027	-0.273	0.035
	2	-0.378	0.033	-0.369	0.038
φ_i	1	0.971	0.0092	0.979	0.0099
	2	0.962	0.0084	0.968	0.0095
σ_η	1	0.188	0.019	0.176	0.024
	2	0.184	0.023	0.171	0.031
ρ_{ij}	12	0.649	0.018	0.646	0.026

In Table 5.19 it can be observed that the SGI based and MCMC based algorithms generated close estimations for all parameters. Standard errors of the parameter estimates with SGI method are slightly lower.

The correlation between asset returns, ρ_{ij} , in all models including the CCC-GARCH are consistent as seen in Table 5.19 and Table 5.20 . Both the MSV-B and the CCC-GARCH model exhibits strong persistence in volatilities however the parameters are not directly comparable since the structures of the models are different.

Table 5.20 : CCC-GARCH parameter estimation results for EUR/TL and USD/TL.

		CCC-GARCH	
	i	Estimate	Std. Error
ω_i	1	0.000	0.000
	2	0.000	0.000
α_i	1	0.107	0.041
	2	0.122	0.039
β_i	1	0.837	0.039
	2	0.818	0.034
ρ_{ij}	12	0.652	0.014

The comparison of the log-volatility estimates of the three fitted model are given in Table 5.21. Here, the logarithms of the squared returns are used as reference for the calculation of RMSEs as described in section 4.4.

Table 5.21 : Log-volatility estimation comparisons for EUR/TL and USD/TL.

	i	Smoothing RMSE	Prediction RMSE
MSV-B MCMC	1	2.713	2.887
	2	2.701	2.879
MSV-B SGI	1	2.658	2.843
	2	2.620	2.806
CCC-GARCH	1	2.869	3.047
	2	2.826	2.974

In Table 5.21 it can be seen that SGI based MSV-B has the lowest RMSEs for both smoothing and prediction. The choice of relatively a higher integration level, $l = 8$ is probably the reason for the better fit for SGI based estimation algorithms. In Table 5.21 it is also observed that MSV-B model has lower RMSEs with both estimation methods than the CCC-GARCH model which is an indication of a better performance in capturing the patterns of the reference.

Figure 5.15 plots the log-volatility smoothing estimates obtained by the SGI based estimation algorithm and MCMC based estimation algorithm for the MSV-B and CCC-GARCH log-volatility fit along with the logarithm of the squared returns as the reference. It can be seen from the charts in Figure 5.15 that the proposed SGI based estimation algorithm successfully captures the patterns of the log-volatilities.

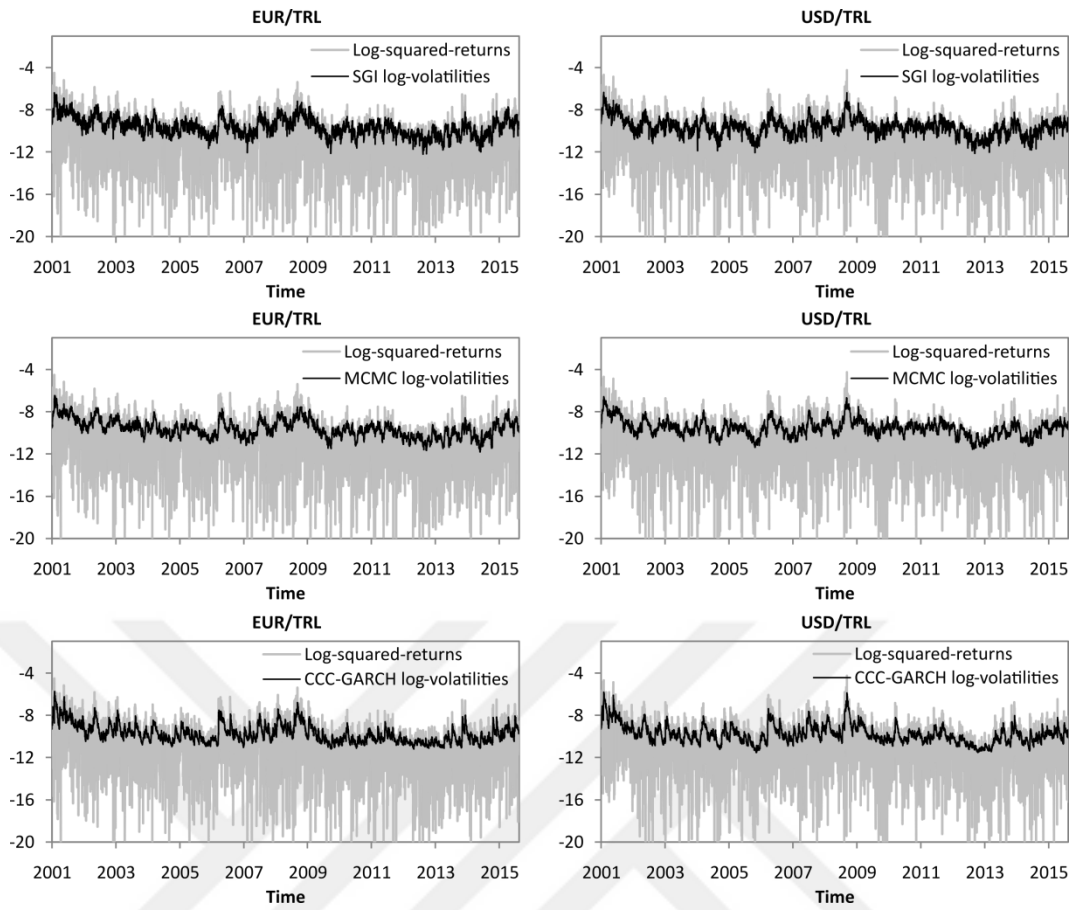


Figure 5.15 : Log-volatility smoothing estimates for EUR/TRL and USD/TRL.



6. CONCLUSION

The capabilities of the MSV models in capturing the stylized facts and dynamics of volatilities and correlations are undisputable. Different model specifications and parameterizations of MSV models can address almost any stylized facts and features about the volatilities and correlations, however addressing several stylized facts and features at the same time is not trivial since there is a requirement of efficient mechanism and parameterizations for handling the correlation and covariance matrices which have special structures and restrictions on their entries especially in time varying setting. A general parametric MSV model which can accommodate both the static and dynamic settings for leverage effects, cross-leverage effects, volatility spillovers and co-movement of asset returns at the same time and furthermore allowing usage of combinations of static and dynamic components in a single model was proposed in this study and this parameterization was referred as MSV-D. A custom-built Bayesian MCMC estimation algorithm for the MSV-D model was also developed.

The results in section 5.1 on simulated and the results in section 5.2 on empirical data showed that the proposed MSV-D model successfully captures the stylized facts of volatility and correlations in both static and dynamic settings. The proposed MSV-D model can also address dynamic cross-leverage and dynamic volatility spillovers by construction which is not an option in currently available MSV models. The proposed MSV-D model can facilitate dynamic and static components at the same time in a single model and providing a flexibility to the modeler. The results of section 5.1 also show that the Bayesian MCMC algorithm developed for the MSV-D model also does its job sufficiently well in estimation with a room for improving its sampling efficiency especially in the complete dynamic setting where more efficient ways for sampling the correlation states \mathbf{q}_t can be found with further research.

The main drawback of the proposed MSV-D model is the quick increase in the number of parameters and dimension of the correlation state space in dynamic setting due to the increasing size of the correlation matrix of returns and volatility errors

which is a common situation in most multivariate models in the literature. However, using the dynamic components selectively and putting restrictions on the parameters of dynamic correlation state processes can help for achieving parsimony.

SV models are nonlinear state space models which require computationally demanding methods for satisfactory estimations. MCMC based estimation algorithms are by far the most popular methods for estimation because of their appealing features. However there are issues on the error control and convergence inherent in MCMC methods. As an alternative approach SGI based estimation algorithms which are new to the SV field are developed and evaluated for the estimation problems of MSV models in comparison with the MCMC based estimation algorithms.

Results in section 5.3 show that the SGI based estimation algorithms perform well by achieving the accuracy of MCMC based estimation algorithms and even surpass them in certain conditions. Better error control and convergence properties of SGI methods are also shown in the results. It is showed that SGI methods, a type of numerical integration method, can be used for multi dimensional problems as an alternative to the MCMC based estimation algorithms. Despite the effect of dimensionality is significantly decreased in SGI methods when compared to the classical numerical integration methods, there is still some dependency on the dimension and SGI method can struggle on very high dimensional problems. However, the algorithms based on SGI are sequential (i.e on-line) algorithms in contrast with the batch structure of the MCMC methods and this sequential structure allows splitting the computational burden among time periods which is an important consideration for practical implementations. Construction of sparse grid formulas from other univariate formulas such as the Gaussian quadrature rules which probably be more effective and suitable to MSV model density structures is one of the future research direction for the SGI based approach. Another direction for improvement for the SGI based approach would be constructing the sparse grid by adjusting the integration formula level at each time step for better error control and computational efficiency which can significantly improve the method from the algorithmic perspective. Hybrid approaches combining Monte Carlo based methods with sparse grid integration based methods for computational advantage and better convergence could lead to further research.

The computational requirements of the both MCMC and SGI based algorithms are high and it is showed that GPU implementation is an efficient and low-cost solution in accelerating the execution times of the estimation algorithms. Although the size of a typical problem exceeds the resources provided by a single GPU and theoretical speed up values could not be tested and scaling could not be observed, single GPU results having speed up values up to 16 for MCMC based algorithms and speed up values up to 25 for SGI based algorithms are obtained as shown by the results in section 5.3. The contributions of the developed parallel approaches for the MCMC and SGI based algorithms and their GPU implementations are clear from the results which are promising for larger scale parallel architecture implementations. Alternative parallelization and acceleration approaches for estimation algorithms and extending the computing architecture and software programs to distributed and cluster settings are further research directions in the computational aspect.



REFERENCES

- Asai, M., & McAleer, M.** (2006). Asymmetric multivariate stochastic volatility. *Econometric Reviews*, 25 (2-3), 253-473.
- Asai, M., & McAleer, M.** (2009). The structure of dynamic correlations in multivariate stochastic volatility models. *Journal of Econometrics*, 150, 182-192.
- Asai, M., McAleer, M., & Yu, J.** (2006). Multivariate stochastic volatility: A review. *Econometric Reviews*, 25 (2-3), 145-175.
- Black, F., & Scholes, M.** (1973). The pricing of options and corporate liabilities. *The journal of political economy*, 637-654.
- Bollerslev, T.** (1990). Modeling the coherence in short-run nominal exchange rates: a multivariate generalized ARCH model. *The Review of Economics and Statistics*, 498-505.
- Bungarts, H. J., & Griebel, M.** (2004). Sparse grids. *Acta Numerica*, 13, 1-123.
- Carlin, B. P., Polson, N., & Stoffer, D.** (1992). A Monte Carlo approach to nonnormal and nonlinear state space modelling. *Journal of American Statistical Association*, 87 (2), 493-500.
- Chib, S.** (2001). Markov chain Monte Carlo methods: computation and inference. *Handbook of econometrics*, 5, 3569-3649.
- Chib, S., & Greenberg, E.** (1995). Understanding the Metropolis-Hastings algorithm. *The American Statistician*, 49 (4), 327-335.
- Chib, S., & Greenberg, E.** (1996). Markov Chain Monte Carlo simulation methods in econometrics. *Econometric Theory*, 12 (3), 409-431.
- Durham, G. B.** (2006). Monte Carlo methods for estimating, smoothing and filtering one and two factor stochastic volatility models. *Journal of Econometrics*, 133, 273-305.
- Engle, R.** (1982). Autoregressive conditional heteroskedasticity with estimates of United Kingdom inflation. *Econometrica: Journal of the Econometric Society*, 987- 1007.
- Engle, R.** (2002). Dynamic conditional correlation: A simple class of multivariate generalized autoregressive conditional heteroskedasticity models. *Journal of Business & Economic Statistics*, 20 (3), 339-350.
- Galant, A., Hsieh, D., & Tauchen, G.** (1997). Estimation of stochastic volatility models with diagnostics. *Journal of Econometrics*, 81, 159-192.
- Gerstner, T.** (2007). *Sparse grid quadrature methods for computational finance*. (Habilitation), University of Bonn, Bonn.

- Gerstner, T., & Griebel, M.** (1998). Numerical integration using sparse grids. *Numerical Algorithms*, 18, 209-232.
- Ghysels, E., Harvey, A., & Renault, E.** (1996). Stochastic volatility. In G. S. Maddala, & C. R. Rao (Eds.), *Handbook of Statistics* (Vol. 14, pp. 119-191). Amsterdam: Elsevier.
- Gourieroux, C., Jasiak, J., & Sufana, R.** (2009). The Wishart autoregressive process of multivariate stochastic volatility. *Journal of Econometrics*, 150, 167-181.
- Harvey, A., & Shephard, N.** (1996). Estimation of an asymmetric stochastic volatility model for asset returns. *Journal of Business and Economic Statistics*, 14 (4), 429-434.
- Harvey, A., Ruiz, E., & Shephard, N.** (1994). Multivariate stochastic variance models. *Review of Economic Studies*, 61, 247-264.
- Hastings, W. K.** (1970). Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57 (1), 97-109.
- Heiss, F., & Winschel, V.** (2006). Estimation with numerical integration on sparse grids. Munich: Technical Report, University of Munich.
- Holtz, M.** (2010). *Sparse grid quadrature in high dimensions with applications in finance and insurance* (Vol. 77). Springer Science & Business Media.
- Ishihara, T., & Omori, Y.** (2012). Efficient Bayesian estimation of a multivariate stochastic volatility model with cross leverage and heavy-tailed errors. *Computational Statistics & Data Analysis*, 56 (11), 3674-3689.
- Ishihara, T., Omori, Y., & Asai, M.** (2014). Matrix exponential stochastic volatility with cross leverage. *Computational Statistics & Data Analysis*, In Press.
- Jacquier, E., Polson, N., & Rossi, P.** (1994). Bayesian analysis of stochastic volatility models. *Journal of Business and Economic Statistics*, 12, 371-417.
- Jacquier, E., Polson, N., & Rossi, P.** (1995). Models and priors for multivariate stochastic volatility. CIRANO Working paper, Montreal, 95s-18.
- Kercheval, A. N.** (2008) On Rebonato and Jackel's Parametrization Method for Finding Nearest Correlation Matrices. *International Journal of Pure and Applied Mathematics*, 45 (3), 383.
- Kitagawa, G.** (1987). Non-Gaussian state space modeling of nonstationary time series. *Journal of American Statistics Association*, 82, 1032-1063.
- Lopes, H. F., McCulloh, R. E., & Tsay, R. S.** (2011). Cholesky stochastic volatility. Discussion Paper.
- Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., & Teller, E.** (1953). Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21 (6), 1087-1092.
- Rebonato, R., & Jäckel, P.** (2011) The most general methodology to create a valid correlation matrix for risk management and option pricing purposes. Available at SSRN 1969689.

- Rosenthal, J. S.** (2000). Parallel computing and Monte Carlo algorithms. *Far east journal of theoretical statistics*, 4 (2), 207-236.
- Sandman, G., & Koopman, S. J.** (1998). Estimation of stochastic volatility models via Monte Carlo maximum likelihood. *Journal of Econometrics*, 87, 271-301.
- Shephard, N., & Andersen, T.** (2009). *Stochastic volatility: origins and overview*. Berlin Heidelberg: Springer.
- Smolyak, S. A.** (1963). Quadrature and interpolation formulas for tensor products of certain classes of functions. *Soviet Mathematics Doklady*, 4, 240-243.
- Tanizaki, H.** (1997). Nonlinear and nonnormal filters using Monte Carlo methods. *Computational Statistics and Data Analysis*, 25 (4), 417-439.
- Tanner, M. A., & Wong, W. H.** (1987). The calculation of posterior distributions by data augmentation. *Journal of the American Statistical Association*, 82 (398), 528-540.
- Taylor, S. J.** (1982). Financial returns modeled by the product of two stochastic processes - A study of daily sugar prices 1961-79. In O. D. Anderson (Ed.), *Time Series Analysis: Theory and Practice 1* (pp. 203-226). Amsterdam, North-Holland.
- Taylor, S. J.** (1986). *Modelling financial time series*. Chichester: John Wiley.
- Tierney, L.** (1994). Markov chains for exploring posterior distributions. *The Annals of Statistics*, 22 (4), 1701-1728.
- Tsay, R.** (2002). *Analysis of financial time series: Financial econometrics*. New York: John Wiley.
- Watanabe, T.** (1999). A non-linear filtering approach to stochastic volatility models with an application to daily stock returns. *Journal of Applied Econometrics*, 14 (2), 101-121.



APPENDICES

APPENDIX A: Integral Notation Definitions

APPENDIX B: Construction of Time Varying Correlation Matrices

APPENDIX C: Important C Program Functions and Sample Codes

Appendix C.1: Correlation matrix parameterization and transformation functions

Appendix C.2: Sparse grid construction functions

Appendix C.3: Examples of GPU device kernels, device functions and their usage

APPENDIX A : Integral Notation Definitions

The definitions and representations of the multiple integral notations used throughout the study is summarised as follows.

Integral with respect to a vector

Let $f(\mathbf{x})$ be a function of vector argument where $\mathbf{x} \in \mathfrak{R}^d$ and Let $\mathbf{x} = (x_1, \dots, x_d)'$ be a d -dimensional vector, then for the integral of $f(\mathbf{x})$ with respect to the vector \mathbf{x} the following equalities hold:

$$\begin{aligned}\int f(\mathbf{x})d\mathbf{x} &\equiv \int \dots \int f(\mathbf{x})dx_1 \dots dx_d \\ &\equiv \int f(\mathbf{x})dx_1 \dots dx_d. \\ &\equiv \int f(x_1, \dots, x_d)dx_1 \dots dx_d.\end{aligned}\tag{A.1}$$

Integral with respect to a set of vectors

Let $f(\mathbf{x}_1, \dots, \mathbf{x}_t)$ be a function of vector arguments where $\mathbf{x}_i \in \mathfrak{R}^d$ and $\mathbf{X} = \{\mathbf{x}_1, \dots, \mathbf{x}_t\}$ be a set of vectors where $\mathbf{x}_i = (x_{1,i}, \dots, x_{d,i})' \in \mathfrak{R}^d$ for $i = 1, \dots, t$, then for the integral of $f(\mathbf{X})$ with respect to the set \mathbf{X} , the following equalities hold:

$$\begin{aligned}\int f(\mathbf{X})d\mathbf{X} &\equiv \int \dots \int f(\mathbf{X})d\mathbf{x}_1 \dots d\mathbf{x}_t \\ &\equiv \int f(\mathbf{x}_1, \dots, \mathbf{x}_t)d\mathbf{x}_1 \dots d\mathbf{x}_t. \\ &\equiv \int f(x_{1,1}, \dots, x_{d,1}, \dots, x_{1,t}, \dots, x_{d,t})dx_{1,1} \dots dx_{d,1} \dots dx_{1,t} \dots dx_{d,t}.\end{aligned}\tag{A.2}$$

Integral with respect to a matrix

Let $f(\mathbf{X})$ be a function of diagonal matrix argument where $\mathbf{X} \in \mathfrak{R}^{d \times d}$ and let $\mathbf{X} = \text{diag}(x_1, \dots, x_d)$ then then for the integral of $f(\mathbf{X})$ with respect to the set \mathbf{X} , the following equalities hold:

$$\begin{aligned}\int f(\mathbf{X})d\mathbf{X} &\equiv \int \dots \int f(\mathbf{X})dx_1 \dots dx_d \\ &\equiv \int f(\mathbf{X})dx_1 \dots dx_d. \\ &\equiv \int f(x_1, \dots, x_d)dx_1 \dots dx_d.\end{aligned}\tag{A.3}$$

APPENDIX B : Construction of Time Varying Correlation Matrices

Figure B.1 illustrates the mechanism that constructs valid correlation matrices based on an AR(1) process in the proposed MSV-D model.

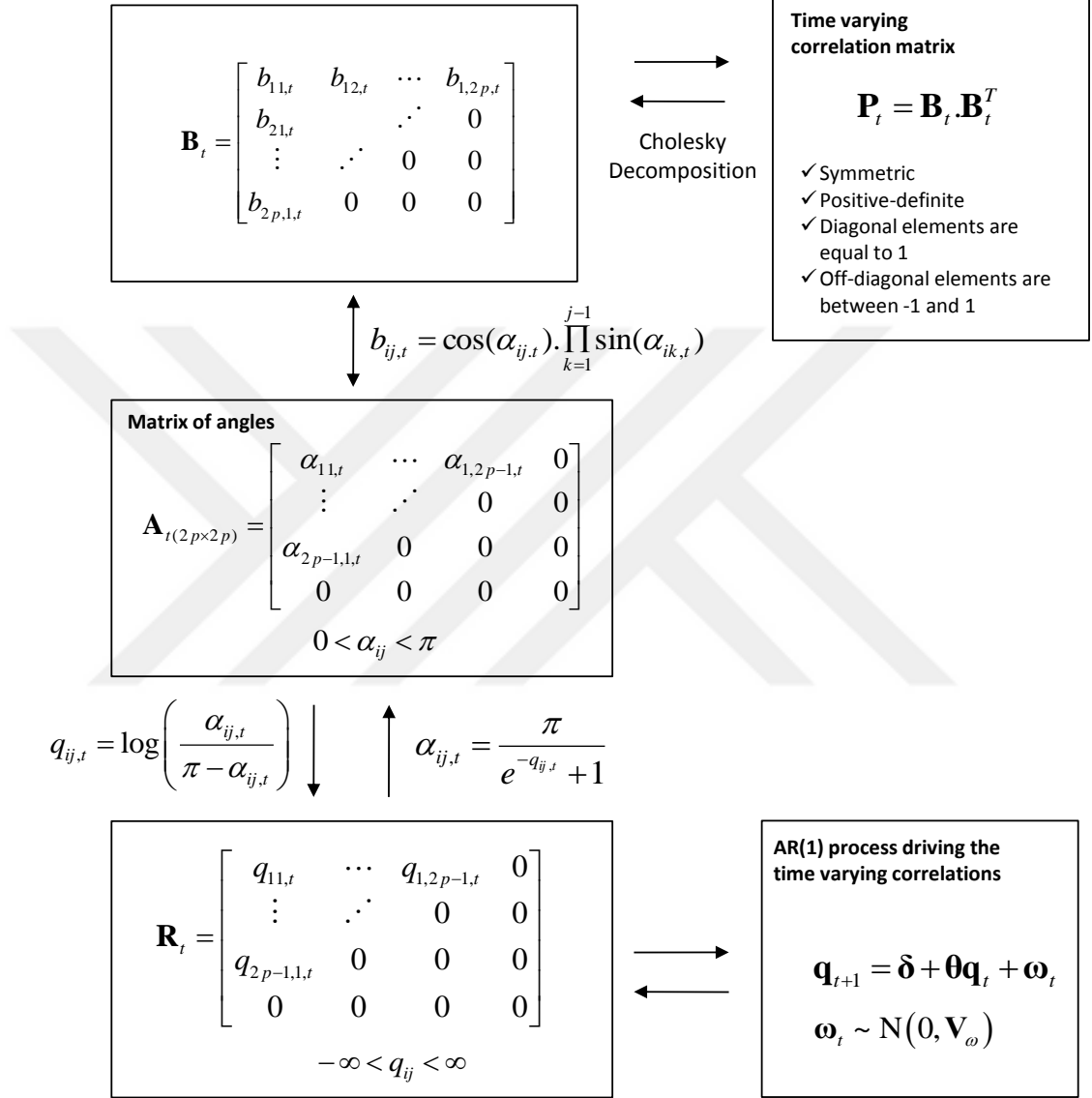


Figure B.1 : Time varying correlation matrix construction mechanism of the MSV-D model.

APPENDIX C : Important C Program Functions and Sample Codes

Appendix C.1: Correlation matrix parameterization and transformation functions

```
// Correlation matrix parameterization and transformation functions
// by Halil Erturk Esen
```

```
void corr_to_angle(int dim, float *cor, float *angle){
    int i, j, k, nn;
    float den, *element, *bmat, *temp1, *temp2;
    nn = dim*dim;
    element = (float *)malloc(sizeof(float)*nn);
    temp2 = (float *)malloc(sizeof(float)*nn);
    temp1 = (float *)malloc(sizeof(float)*nn);
    bmat = (float *)malloc(sizeof(float)*nn);
    //Elementary matrix
    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            if (i+j==dim-1){
                element[i+dim*j]=1.0;
            }
            else{
                element[i+dim*j]=0.0;
            }
        }
    }
    sqrmm(element, cor, dim, temp1); //Matrix multiplication
    sqrmm(temp1, element, dim, temp2);
    chol(temp2, dim, temp1); //Cholesky decomposition

    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            temp2[i+dim*j] = temp1[j+dim*i];
        }
    }
    sqrmm(element, temp2, dim, bmat);
    //Extract angles
    for (i=0;i<dim;i++){
        for (j=0;j<dim;j++){
            if(i+j<dim-1){
                den=1.0;
                for(k=0;k<j;k++){
                    den = den*sin(angle[i+dim*k]);
                }
                angle[i+j*dim]=acos(bmat[i+j*dim]/den);
            }
            else{
                angle[i+j*dim]=0.0;
            }
        }
    }
    // free(...);
}

void angle_to_corr(int dim, float *angle, float *cor){
    int i, j, k, nn;
    float den, *rc, *bmat, *temp;
    nn=dim*dim;
    rc = (float *)malloc(sizeof(float)*nn);
```

```

bmat = (float *)malloc(sizeof(float)*nn);
temp = (float *)malloc(sizeof(float)*nn);
for(i=0;i<dim;i++){
    for (j=0;j<dim;j++){
        if(i+j<dim){
            den=1.0;
            for(k=0;k<j;k++){
                den = den*sin(angle[i+dim*k]);
            }
            bmat[i+j*dim]=cos(angle[i+j*dim])*den;
        }
        else{
            bmat[i+j*dim]=0.0;
        }
    }
}
for(i=0;i<dim;i++){
    for (j=0;j<dim;j++){
        temp[i+dim*j] = bmat[j+dim*i];
    }
}
sqrmm(bmat, temp, dim, rc);

for(i=0;i<dim;i++){
    for(j=0;j<dim;j++){
        cor[i+dim*j]=rc[i+dim*j];
    }
}
// free(...);
}

void q_to_angle(int dim, float *q, float *angle){
    int i, j;
    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            if(i+j<dim-1){
                angle[i+dim*j]= PI/(exp(-1*q[i+dim*j-
                    (j*(j+1)/2)])+1);
            }
            else{
                angle[i+dim*j]=0.0;
            }
        }
    }
}

void angle_to_q(int dim, float *angle, float *q){
    int i, j;
    for(i=0;i<dim;i++){
        for(j=0;j<dim-1-i;j++){
            q[i+dim*j-(j*(j+1)/2)] = -1*log((PI/angle[i+dim*j])-1);
        }
    }
}

```

Appendix C.2 : Sparse grid construction functions

```
//Functions for the construction of the regular sparse grids
//for arbitrary dimension and accuracy level based
//on the trapezoid rule
//by Halil Erturk Esen
```

```
//Simplex size
int simsize(int dim, int level){
    int i, j=0, sm, cnt=0;
    int *ka;
    //printf("Dim = %d \n",dim);
    ka = (int *)malloc(sizeof(int)*(dim+1));
    for(i=0;i<=dim;i++){
        ka[i]=1;
    }
    while(j >= 0){
        j = dim;
        sm = level + dim;
        while(sm > level + dim - 1){
            ka[j] = 1;
            j = j - 1;

            if(j>=0){
                ka[j] = ka[j] + 1;
            }

            //printf("ka[%d]= %d \n", j, ka[j]);

            sm=0;
            for(i=0; i<dim; i++){
                sm = sm + ka[i];
            }
            cnt=cnt+1;
        }
        free(ka);
        return cnt;
    }
}
```

```
//Simplex construction
void simplexc(int dim, int level, int *splx){
    int i, j=0, sm, size, cnt=0;
    int *k;
    k = (int *)malloc(sizeof(int)*(dim+1));
    size = simsize(dim, level);

    for(i=0;i<=dim;i++){
        k[i]=1;
    }
    while(j >= 0){
        j = dim;
        sm = level + dim;
        while(sm > level + dim - 1){
            k[j] = 1;
            j = j - 1;
            if(j>=0){
                k[j] = k[j] + 1;
            }
        }
    }
}
```



```

        sm=0;
        for(i=0; i<dim; i++){
            sm = sm + k[i];
        }
    }
    for(i=0; i<dim; i++){
        splx[cnt+size*i] = k[i];
    }
    cnt=cnt+1;
}
free(k);
}
// Sparse grid size
int sparsegridsize(int dim, int level){
    int i, j, gsize=1, qsize, fullsize=0, sims, cnt, mn, ss;
    int *quadn, *deltan, *deltans, *qngr, *dngr,
        *simplex, *simplexnnd;
    float *quadw, *deltaw, *deltaws;

    //Quadrature and delta rule nodes and weights-----;
    qngr=(int *)malloc(sizeof(int)*level);
    dngr=(int *)malloc(sizeof(int)*level);
    for(i=level;i>0;i--){
        qngr[i-1]=intpow(2,i)-1;
    }
    qsize=level*qngr[level-1];
    quadn=(int *)malloc(sizeof(int)*qsize);
    quadw=(float *)malloc(sizeof(float)*qsize);
    deltan=(int *)malloc(sizeof(int)*qsize);
    deltaw=(float *)malloc(sizeof(float)*qsize);
    deltans=(int *)malloc(sizeof(int)*qsize);
    deltaws=(float *)malloc(sizeof(float)*qsize);

    for(i=0;i<level;i++){ //Quadrature nodes and weights
        ss=(qngr[level-1]-qngr[i])/(qngr[i]+1);
        cnt=0;
        mn=0;
        for(j=0;j<qngr[level-1];j++){
            if(cnt<ss){
                quadn[i+level*j]=0;
                quadw[i+level*j]=0;
                cnt=cnt+1;
            }
            else{
                quadn[i+level*j]=1;
                if(mn==0 || mn==(qngr[i]-1)){
                    quadw[i+level*j]=1.5f;
                }
                else{
                    quadw[i+level*j]=1;
                }
                cnt=0;
                mn=mn+1;
            }
        }
    }

    for(i=0;i<level;i++){//Delta nodes and weights
        dngr[i]=0;

```

```

        for(j=0;j<qngr[level-1];j++){
            if(i==0){
                deltan[i+level*j]=quadrn[i+level*j];
                deltaw[i+level*j]=quadrw[i+level*j];
            }
            else{
                deltan[i+level*j]=quadrn[i+level*j]-
                    quadrn[i-1+level*j];
                deltaw[i+level*j]=quadrw[i+level*j]-
                    quadrw[i-1+level*j];
            }
            if(deltan[i+level*j]==0){
                deltaw[i+level*j]=0;
            }
            dngr[i]=dngr[i]+deltan[i+level*j];
        }
    }

    for(i=0;i<level;i++){//Stacked node indices and weight matrices
        for(j=0;j<qngr[level-1];j++){
            deltans[i+level*j]=0;
            deltaws[i+level*j]=0;
        }
    }

    for(i=0;i<level;i++){
        cnt=0;
        for(j=0;j<qngr[level-1];j++){
            if(deltan[i+level*j]==1){
                deltans[i+level*cnt]=j;
                deltaws[i+level*cnt]=deltaw[i+level*j];
                //printf("Check: %d \n", i+level*cnt);
                cnt=cnt+1;
            }
        }
    }

    //-----End of difference rule nodes and weights-----;
    sims=simsize(dim, level);
    simplex=(int *)malloc(sizeof(int)*sims*dim);
    simplexnnd=(int *)malloc(sizeof(int)*sims*dim);
    simplexc(dim, level, simplex);
    for(i=0;i<sims;i++){//nnd vectors over simplex
        gsize=1;
        for(j=0;j<dim;j++){
            simplexnnd[i+sims*j]=dngr[(simplex[i+sims*j]-1)];
            gsize=gsize*simplexnnd[i+sims*j];
        }
        fullsize=fullsize+gsize;
    }
    // free(...);
    return fullsize;
}

// Final sparse grid construction
void sparsegcon(int dim, int level, int *coords, float *weights){
    int h, g, i, j, v, gsize=1, sgsz, qsize, fullsize=0, sims, cnt,
        cond, mn, ss, lin, eq, eqcheck;
    int *quadrn, *deltan, *deltans, *qngr, *dngr,

```

```

        *k, *prngr, *prntgr, *simplex, *simplexnnd, *nnd,
        *levd, *grid, *sgrid, *tempn, *gmind;
float *quadw, *deltaw, *deltaws, *prwtgr, *prwv, *wvec,
      *cwvec, *swvec, tempw, cuweight;

//Quadrature and delta rule nodes and weights-----;
qngr=(int *)malloc(sizeof(int)*level);
dngr=(int *)malloc(sizeof(int)*level);
for(i=level;i>0;i--){
    qngr[i-1]=intpow(2,i)-1;
}
qsize=level*qngr[level-1];
quadn=(int *)malloc(sizeof(int)*qsize);
quadw=(float *)malloc(sizeof(float)*qsize);
deltan=(int *)malloc(sizeof(int)*qsize);
deltaw=(float *)malloc(sizeof(float)*qsize);
deltans=(int *)malloc(sizeof(int)*qsize);
deltaws=(float *)malloc(sizeof(float)*qsize);

for(i=0;i<level;i++){ //Quadrature nodes and weights
    ss=(qngr[level-1]-qngr[i])/(qngr[i]+1);
    cnt=0;
    mn=0;
    for(j=0;j<qngr[level-1];j++){
        if(cnt<ss){
            quadn[i+level*j]=0;
            quadw[i+level*j]=0.0;
            cnt=cnt+1;
        }
        else{
            quadn[i+level*j]=1;
            if((qngr[i]-1)==0){
                quadw[i+level*j]=(2.0)/(qngr[i]+1);
            }
            else if(mn==0 || mn==(qngr[i]-1)){
                quadw[i+level*j]=(1.5)/(qngr[i]+1);
            }
            else{
                quadw[i+level*j]=(1.0)/(qngr[i]+1);
            }
            cnt=0;
            mn=mn+1;
        }
    }
}

for(i=0;i<level;i++){//Delta nodes and weights
    dngr[i]=0;
    for(j=0;j<qngr[level-1];j++){
        if(i==0){
            deltan[i+level*j]=quadn[i+level*j];
            deltaw[i+level*j]=quadw[i+level*j];
        }
        else{
            deltan[i+level*j]=quadn[i+level*j]-
                quadn[i-1+level*j];
            deltaw[i+level*j]=quadw[i+level*j]-
                quadw[i-1+level*j];
        }
    }
}

```

```

        if(deltaw[i+level*j]!=0){
            deltan[i+level*j]=1;
        }
        dngr[i]=dngr[i]+deltan[i+level*j];
    }
}

for(i=0;i<level;i++){//Stacked node indices and weight matrices
    for(j=0;j<qngr[level-1];j++){
        deltans[i+level*j]=0;
        deltaws[i+level*j]=0;
    }
}

for(i=0;i<level;i++){
    cnt=0;
    for(j=0;j<qngr[level-1];j++){
        if(deltan[i+level*j]==1){
            deltans[i+level*cnt]=j;
            deltaws[i+level*cnt]=deltaw[i+level*j];
            cnt=cnt+1;
        }
    }
}

//-----End of difference rule nodes and weights-----;

//=====
//CONSTRUCTING THE GRID AND ASSOCIATED WEIGHTS
//=====
//Simplex construction-----;
sims=simsize(dim, level);
simplex=(int *)malloc(sizeof(int)*sims*dim);
simplexnnd=(int *)malloc(sizeof(int)*sims*dim);
simplexc(dim, level, simplex);

for(i=0;i<sims;i++){//nnd vectors over simplex
    gsize=1;
    for(j=0;j<dim;j++){
        simplexnnd[i+sims*j]=dngr[(simplex[i+sims*j]-1)];
        gsize=gsize*simplexnnd[i+sims*j];
    }
    fullsize=fullsize+gsize;
}

//Full grid
grid=(int *)calloc(fullsize*dim, sizeof(int));
wvec=(float *)calloc(fullsize, sizeof(float));
cwvec=(float *)calloc(fullsize, sizeof(float));
gmind=(int *)calloc(fullsize, sizeof(int));
nnd=(int *)malloc(sizeof(int)*dim);
levd=(int *)malloc(sizeof(int)*dim);

//Loop over simplex combinations
lin=0;
for(v=0;v<sims;v++){
    for(i=0;i<dim;i++){
        nnd[i]=simplexnnd[v+sims*i];
    }
}

```

```

        lev[d[i]]=simplex[v+sims*i]-1;
    }
    //Product grid nodes primary construction-----;
    k = (int *)malloc(sizeof(int)*dim);
    gsize=1;
    for(i=0;i<dim;i++){
        k[i]=1;
        gsize=gsize*nnd[i];
    }
    prngr = (int *)malloc(sizeof(int)*gsiz*dim);
    cnt=0;
    for(i=0;i<dim;i++){
        prngr[cnt+gsiz*i]=k[i];
    }
    cnt=1;
    i=0;
    cond=0;
    while(cond == 0){
        k[i]=k[i]+1;
        cond=(k[dim-1] > nnd[dim-1]);
        if (k[i]>nnd[i]){
            k[i]=1;
            i = i + 1;
        }
        else{
            for(j=0; j<dim; j++){
                prngr[cnt+gsiz*j]=k[j];
            }
            cnt=cnt+1;
            i=0;
        }
    }
}

//-----End of product grid nodes primary construction-----;

//Product grid transformed nodes and weights construction---;

prntgr = (int *)malloc(sizeof(int)*gsiz*dim);
prwtgr = (float *)malloc(sizeof(float)*gsiz*dim);
prwv = (float *)malloc(sizeof(float)*gsiz);
for(i=0;i<gsiz;i++){
    for(j=0; j<dim; j++){
        prntgr[i+gsiz*j]= deltans[levd[j]+
                                level*(prngr[i+gsiz*j]-1)];
        prwtgr[i+gsiz*j]= deltaws[levd[j]+
                                level*(prngr[i+gsiz*j]-1)];
    }
}
for(i=0;i<gsiz;i++){
    prwv[i]=1;
    for(j=0; j<dim; j++){
        prwv[i]= prwv[i]*prwtgr[i+gsiz*j];
    }
}

//Copy product nodes and weights components to grid matrix
for(i=0;i<gsiz;i++){
    for(j=0; j<dim; j++){
        grid[lin+fullsiz*j]=prntgr[i+gsiz*j];
    }
}

```

```

        }
        wvec[lin]=prwv[i];
        lin=lin+1;
    }
    free(k);
    free(prngr);
    free(prntgr);
    free(prwtgr);
    free(prwv);
}

//Aggregate grid matrix for repeat nodes
//Sort the grid matrix
tempn=(int *)malloc(sizeof(int)*dim);
for(h=1;h<fullsize-1;h++){
    for(g=0;g<fullsize-h;g++){
        eqcheck = 1;
        j=0;
        while(j<dim && eqcheck == 1){
            if(grid[g+fullsize*j] < grid[g+1+fullsize*j]){
                eq = 0;
                eqcheck = 0;
            }
            else if(grid[g+fullsize*j] >
                    grid[g+1+fullsize*j]){
                eq = 1;
                eqcheck = 0;
            }
            else{
                eqcheck = 1;
            }
            j=j+1;
        }
        if(eq==1){
            for(j=0;j<dim;j++){
                tempn[j]=grid[g+fullsize*j];
                grid[g+fullsize*j]=grid[g+1+fullsize*j];
                grid[g+1+fullsize*j] = tempn[j];
            }
            tempw = wvec[g];
            wvec[g]=wvec[g+1];
            wvec[g+1]=tempw;
        }
    }
}

// Track the repeating nodes
cnt=0;
for(i=0;i<fullsize-1;i++){
    if(i==0){
        cuweight = wvec[i];
        cwvec[i] = cuweight;
    }
    eqcheck = 1;
    j=0;
    while(j<dim && eqcheck ==1){
        if (grid[i+1+fullsize*j] > grid[i+fullsize*j] ){
            eq = 0;
            eqcheck = 0;
        }
    }
}

```

```

        }
        else{
            eq = 1;
            eqcheck = 1;
        }
        j=j+1;
    }
    if(eq==0){
        cnt = cnt + 1;
        cuweight = wvec[i+1];
    }
    else {
        cuweight=cuweight+wvec[i+1];
    }
    gmind[i+1]=cnt;
    cwvec[i+1] = cuweight;
}

sgsize = gmind[fullsize-1]+1;

//Final sparse grid array

sgrid=(int *)calloc(sgsize*dim, sizeof(int));
swvec=(float *)calloc(sgsize, sizeof(float));
h=0;
for(i=0;i<fullsize-1;i++){
    if(gmind[i]!=gmind[i+1]){
        swvec[h]=cwvec[i];
        for(j=0;j<dim;j++){
            sgrid[h+sgsize*j]=grid[i+fullsize*j];
        }
        h=h+1;
    }
}

swvec[sgsize-1]=cwvec[fullsize-1];
for(j=0;j<dim;j++){
    sgrid[sgsize-1+sgsize*j]=grid[fullsize-1+fullsize*j];
}

for(i=0; i<sgsize; i++){
    for(j=0; j<dim; j++){
        coords[i+sgsize*j]=sgrid[i+sgsize*j];
    }
}

for(i=0; i<sgsize; i++){
    weights[i]=swvec[i];
}

//free(quadn);
}

```

Appendix C.3: Examples of GPU device kernels, device functions and their usage

Examples of GPU device functions:

```
//Vector matrix multiplication on device
__device__ void gpu_sqrvm(float *a, float *b, int dim, float *c){
    int i, j;
    for(i=0;i<dim;i++){
        c[i]=0.0;
        for(j=0;j<dim;j++){
            c[i]+=a[j]*b[j+dim*i];
        }
    }
}

// Cholesky decomposition on device
__device__ void gpu_chol(float *a, int dim, float *b){
    int i, j, k;
    float rs;

    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            b[j+dim*i] = 0.0;
        }
    }

    for(i=0;i<dim;i++){
        for(j=0;j<dim;j++){
            if(i==j){
                rs = 0.0;
                for(k=0;k<j;k++){
                    rs = rs + ( b[k+dim*i] ) * ( b[k+dim*j] );
                }
                b[j+dim*i] = sqrt(a[i+dim*j]-rs);
            }
            else if(i>j){
                rs = 0.0;
                for(k=0;k<j;k++){
                    rs = rs + ( b[k+dim*i] ) * ( b[k+dim*j] );
                }
                b[j+dim*i] = (a[i+dim*j]-rs)/b[j+dim*j];
            }
            else{
                b[j+dim*i] = 0.0;
            }
        }
    }
}

The following is a simplified illustration of the GPU device kernel from the implemented MCMC with EM smoothing algorithm device kernel.
```

```
__global__ void mcmc_kernel_s1(int k, int offs, int nt, int pcl_size, int
    ts_t, int dimh, float sc, float *ry, float *phi,
    float *gamma, float *theta, float *delta, float
    *vv_var, float *qq_var, float *yc, float *yp,
    float *hp, float *hc, float *hn, float *candid_h,
    float *qp, float *qc, float *qn, float *candid_q,
```



```

float *transgridh, float *transgridq, float *h_ini,
float *q_ini, float *h_sigmas, float *ch_qq, float
*p_mu, float *p_vv, float *ch_p_vv, curandState_t
*state, float *rrnd_h, float *rndsc_h, float *rrnd_q,
float *rndsc_q, float *u, float *matf1, float *matf2,
float *matf3, float *matf4, float *matf5, float
*matrr, float *matvv, float *matvr, float *matrv,
float *wrkh1, float *wrkh2, float *wrkh3, float *wrkh4,
float *vech1, float *vech2, float *wrkq1, float *wrkq2,
float *wrkq3, float *wrkq4, float *vecq1, float
*vecq2){

int j, m, trans_k, trans_l, term = 1, noterm = 0;
float numerw, denomw, wght;
int dimq = 2*dimh*(2*dimh-1)/2;
int dimf = 2*dimh;
int h_nn = dimh*dimh;
trans_k = 2*pcl_size+ts_t-2; //Number of rows of transformed grid
if(ts_t % 2 == 0){ //Number of columns of transformed grid
    trans_l = (int)(ts_t/2);
}
else{
    trans_l = (int)((ts_t+1)/2);
}

//GPU Thread id
int tid = offs + threadIdx.x + blockIdx.x*blockDim.x;

int vidh = dimh*tid;
int vidq = dimq*tid;
int midh = dimh*dimh*tid;
int midq = dimq*dimq*tid;
int midf = dimf*dimf*tid;

// Assigning parts of device pointers to threads with thread id.
// Random number generation on device with CURAND library.
// Computations and operations with device functions gpu_*
if(tid<nt+offs){
    //q: Candidate, prev, current, next
    for(j=0;j<dimq;j++){
        rrnd_q[j + vidq]=curand_normal(&state[tid]);
    }
    gpu_sqrnv(ch_qq, &rrnd_q[vidq], dimq, &rndsc_q[vidq]);

    for(j=0;j<dimq;j++){
        qp[j+vidq]=transgridq[k-1 +
            trans_k*tid+trans_k*trans_l*j];
        qc[j+vidq]=transgridq[k-2 + trans_k*tid +
            trans_k*trans_l*j];
        if(tid < trans_l -1){
            qn[j+vidq]=transgridq[k-1 + trans_k*(tid+1)+
                trans_k*trans_l*j];
        }
    }
    .
    .
    for(j=0;j<dimh;j++){
        hp[j+vidh]=transgridh[k-1 +
            trans_k*tid+trans_k*trans_l*j];

```

```

        hc[j+vidh]=transgridh[k-2 + trans_k*tid +
                                trans_k*trans_l*j];
        if(tid < trans_l -1){
            hn[j+vidh]=transgridh[k-1 +
                                trans_k*(tid+1) +
                                trans_k*trans_l*j];
        }
        yc[j+vidh]=ry[2*tid+1+j*ts_t];
        yp[j+vidh]=ry[2*tid+j*ts_t];
        candid_h[j+vidh]=p_mu[j+vidh]+rndsc_h[j+vidh];
    }
    u[tid] = curand_uniform(&state[tid]);

    //Parallel execution with device functions

    if(tid==0){
        numerw =
            ..*gpu_cprnorm(dimh, &hc[vidh], &p_mu[midh],
                &p_vv[midh], &wrkh1[midh], &wrkh2[midh],
                &wrkh3[midh], &vech1[vidh], &vech2[vidh]);

        denomw = ...
    }
    else if (tid>0 && tid<trans_l-1){
        .
    }
    else {
        .
    }
    if(denomw>0){
        wght= numerw/denomw;
        if (wght>1){
            wght=1.0;
        }
    }
    else{
        wght=1.0;
    }

    // Updating device pointers from threads
    if(u[tid] <= wght){
        for(j=0;j<dimh;j++){
            transgridh[k + trans_k*tid + trans_k*trans_l*j]
                = candid_h[j + vidh];
        }
        for(j=0;j<dimq;j++){
            transgridq[k + trans_k*tid + trans_k*trans_l*j]
                = candid_q[j+vidq];
        }
    }
    else{
        for(j=0;j<dimh;j++){
            transgridh[k + trans_k*tid + trans_k*trans_l*j]
                = hc[j+vidh];
        }
        for(j=0;j<dimq;j++){
            transgridq[k + trans_k*tid + trans_k*trans_l*j]
                = qc[j+vidq];
        }
    }
}

```

```

    }
}

```

The following code is a simplified illustration of calling a GPU kernel such as the one above from the host along with buffer transfers.

```

// Allocation of device buffers

cudaMalloc((void**)&dev_transgridh,trans_k*trans_l*dimh*sizeof(float));
cudaMalloc((void**)&dev_transgridq,trans_k*trans_l*dimq*sizeof(float));
cudaMalloc((void**)&dev_ry, h_tnn*sizeof(float));
.
.
// Data transfer to device memory
cudaMemcpy(dev_ry, ry, h_tnn*sizeof(float),cudaMemcpyHostToDevice);
.
.
offs = 0;
nt = (k-1)/2;
blocks_n =(int)((((k-1)/2+threadsize-1)/threadsize));

// Device kernel call for parallel execution
mcmc_kernel_s1<<<blocks_n, threadsize>>>(k, offs, nt, pcl_size, ts_t,
    dimh, ..., dev_transgridh, dev_transgridq, ....);
.
.

//Synchronization and blocking
cudaDeviceSynchronize();
.
.
//Data transfer back to host
cudaMemcpy(transgridh,dev_transgridh,trans_k*trans_l*dimh*sizeof(float),
    cudaMemcpyDeviceToHost);
cudaMemcpy(transgridq,dev_transgridq,trans_k*trans_l*dimq*sizeof(float),
    cudaMemcpyDeviceToHost);

```



CURRICULUM VITAE



Name-Surname : Halil Ertürk ESEN
Date and place of birth : 20.12.1976, Adana
E-mail : erturk.esen@gmail.com, eesen@itu.edu.tr

EDUCATION:

- B.Sc.** : 1999, Istanbul Technical University, Faculty of Management, Industrial Engineering Department.
- M.Sc.** : 2002, Bogazici University, Institute for Graduate Studies in Science and Engineering, Industrial Engineering Department.

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Esen, H. E.**, in press, 2016, Multivariate stochastic volatility estimation with sparse grid integration, *Journal of Mathematical Finance*, 6, 68-81. <http://dx.doi.org/10.4236/jmf.2016.61009>

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS :

- Akpınar, M. E., Kocak, I., Gurpinar, B., & **Esen, H. E.**, 2011, Effects of soft palate implants on acoustic characteristics of voice and articulation. *Journal of Voice*, 25(3), 381-386.
- Akpınar, H., Tüfek, İ., Atuş, F., **Esen, H. E.**, & Kural, A. R., 2009, Doppler ultrasonography-guided pelvic plexus block before systematic needle biopsy of the prostate: A prospective randomized study. *Urology*, 74(2), 267-271.
- Tüfek, İ., Akpınar, H., Atuş, F., Öbek, C., **Esen, H. E.**, Keskin, M. S., & Kural, A. R., 2012, The impact of local anesthetic volume and concentration on pain during prostate biopsy: a prospective randomized trial. *Journal of Endourology*, 26(2), 174-177.

