THE UNIVERSITY *of* EDINBURGH

Sparse Inverse Covariance Estimation in Gaussian Graphical Models

Peter Raymond Orchard

Doctor of Philosophy

Institute for Adaptive and Neural Computation

School of Informatics

University of Edinburgh

2014

Abstract

One of the fundamental tasks in science is to find explainable relationships between observed phenomena. Recent work has addressed this problem by attempting to learn the structure of graphical models - especially Gaussian models - by the imposition of sparsity constraints.

The graphical lasso is a popular method for learning the structure of a Gaussian model. It uses regularisation to impose sparsity. In real-world problems, there may be latent variables that confound the relationships between the observed variables. Ignoring these latents, and imposing sparsity in the space of the visibles, may lead to the pruning of important structural relationships. We address this problem by introducing an expectation maximisation (EM) method for learning a Gaussian model that is sparse in the joint space of visible and latent variables. By extending this to a conditional mixture, we introduce multiple structures, and allow side information to be used to predict which structure is most appropriate for each data point. Finally, we handle non-Gaussian data by extending each sparse latent Gaussian to a Gaussian copula. We train these models on a financial data set; we find the structures to be interpretable, and the new models to perform better than their existing competitors.

A potential problem with the mixture model is that it does not require the structure to persist in time, whereas this may be expected in practice. So we construct an input-output HMM with sparse Gaussian emissions. But the main result is that, provided the side information is rich enough, the temporal component of the model provides little benefit, and reduces efficiency considerably.

The GWishart distribution may be used as the basis for a Bayesian approach to learning a sparse Gaussian. However, sampling from this distribution often limits the efficiency of inference in these models. We make a small change to the state-of-the-art block Gibbs sampler to improve its efficiency. We then introduce a Hamiltonian Monte Carlo sampler that is much more efficient than block Gibbs, especially in high dimensions. We use these samplers to compare a Bayesian approach to learning a sparse Gaussian with the (non-Bayesian) graphical lasso. We find that, even when limited to the same time budget, the Bayesian method can perform better.

In summary, this thesis introduces practically useful advances in structure learning for Gaussian graphical models and their extensions. The contributions include the addition of latent variables, a non-Gaussian extension, (temporal) conditional mixtures, and methods for efficient inference in a Bayesian formulation.

Lay Summary

One of the fundamental tasks in science is to find and explain relationships between observed phenomena. For example, consider a set of stocks whose prices are recorded daily. One might expect that the prices of companies in the same market sector, or companies that have business relationships, would move together. The task is to uncover such dependencies from the price data alone.

This type of problem has received considerable attention in recent years. However, most methods do not take account of possible unobserved factors that may influence the observations; if these were known, the dependencies may be more easily explained. Continuing the financial example, a set of stock prices may appear to be interrelated in a complex manner, but if it were known that they are all from the same market sector, that would explain much about their relationships. We introduce a method to take unobserved factors into account.

We extend this method in various ways to make it more useful in practical situations. One way is to incorporate "side information". This is additional observed data that help us to predict the phenomena of interest. In the financial example, side information may consist of market indices or measures of market volatility.

We then study the question of whether it is useful to add a temporal component to these methods. For example, we might try to learn how strongly a bull or bear market persists in time. We find that the temporal component is indeed beneficial, unless the available side information is strongly predictive of the market state.

Finally, we study a "Bayesian" approach to the problem of learning relationships between observed phenomena. The Bayesian approach is more principled – more "correct" in some sense – but is often thought to run more slowly. We introduce a procedure that is more efficient than existing methods. We then compare the Bayesian approach with a popular alternative technique, and demonstrate that the Bayesian method can perform better, even when both methods are allowed to run for the same time.

Acknowledgements

I have been tremendously fortunate in the extent and quality of the supervision afforded to me during this work. My primary supervisor, Amos Storkey, guided me expertly through the PhD, giving me the freedom to pursue my own ideas, but challenging me and steering me in the right direction when necessary. His friendly, approachable nature make for a pleasant work environment, while his incredible knowledge and insight never cease to amaze me. My second supervisor, Felix Agakov, spent an incredible amount of time with me, discussing and developing the ideas in this thesis. His patience and work ethic are inspirational. I especially covet his astonishing ability to see immediately the practical applications of novel research. Without Amos and Felix, this PhD would not have been possible.

I also wish to thank the following people for their assistance in the course of this work: Subramanian Ramamoorthy, the third member of my PhD advisory panel, for reviewing my work and offering many helpful suggestions; Yichuan Zhang for many discussions concerning Hamiltonian Monte Carlo sampling, and for sharing his code; Iain Murray for some helpful discussions; and Hao Wang for sharing his code.

The Edinburgh machine learning department is a fantastic place to work: you are surrounded by smart, intelligent, motivated people. The research environment is both friendly and intellectually stimulating. I have learned so much during our reading groups and discussion sessions. I am grateful to the department for the PhD studentship allocated to me, and to the Engineering and Physical Sciences Research Council (EPSRC) for supplying that source of funding. Additionally, I thank the department for providing me funding to attend some excellent conferences.

I have made some amazing friends during my time in Edinburgh. I am grateful to Athina Spiliopoulou for her technical input, support and encouragement, and all the coffee breaks in the park; to Benjamin Rosman for his technical suggestions, and the many and varied lunch-time discussions; and all the many people both within and outside the Informatics Forum who made my time on the PhD course so interesting and enjoyable.

Most endeavours in life begin with the support of family. I would never have been in a position to begin this PhD without the incredible people that brought me up, provided for me, educated, and inspired me throughout my life. My deepest gratitude goes to my parents, Joan and Peter Orchard, and my sister Dionne Paull, for the loving, stable, close-knit family on which I can always depend. I am thankful to my grandparents, Marie and Frederick Bailey, and Joan and Archibald Orchard, whose love and sense

of fun had a profound impact during my formative years. Finally, I am grateful to my young nieces, Darcey and Mari Paull, whose presence always brings a smile to my face.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Peter Raymond Orchard*)

Table of Contents

Notation

Vectors and Matrices

a	Scalars are written in plain typeface.
$\boldsymbol{a}$	Vectors are lowercase letters written in bold.
$\boldsymbol{A}$	Matrices are uppercase letters written in bold.
a_i	Element i of vector $\boldsymbol{a}$.
A_{ij}	Element (i, j) of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{i:}$	Row i of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{:j}$	Column j of matrix $\boldsymbol{A}$.
$\boldsymbol{A}_{I,\mathcal{J}}$	The matrix consisting of the rows I and the columns $\mathcal{J}$ of matrix $\boldsymbol{A}$ (where I and $\mathcal{J}$ are sets of indices).
$\boldsymbol{a}_i$	The i^{th} in a sequence of vectors.
$\underline{a}$, $\underline{\boldsymbol{a}}$, $\underline{\boldsymbol{A}}$	Random variables, vectors, and matrices are underlined to distinguish them from their realisations.
$\|\boldsymbol{a}\|_2$	Euclidean norm (2-norm) of vector $\boldsymbol{a}$.
$diag(\boldsymbol{A})$	The diagonal of matrix $\boldsymbol{A}$ as a column vector.
$vec(\boldsymbol{A})$	Vector consisting of the stacked columns of matrix $\boldsymbol{A}$, leftmost column first.

- If it is clear from context that a symbol is a random variable, the underline is usually omitted.

Mathematical Symbols

$\mathbb{R}$	The real numbers.
$\mathbb{R}_+$	The non-negative real numbers.
S_+	The positive-semidefinite cone.
S_{++}	The positive-definite cone.
$\boldsymbol{A} \succ 0$	$\boldsymbol{A}$ is positive-definite.
$\boldsymbol{A} \succeq 0$	$\boldsymbol{A}$ is positive-semidefinite.
$\odot$	Hadamard product.
$\otimes$	Kronecker product.
$\circ$	Function composition.
$\mathbb{E}[\,\underline{a}\,]$	Expectation of $\underline{a}$.
$\mathbb{E}_{\boldsymbol{\theta}}[\,\underline{a}\,]$	Expectation of $\underline{a}$ over a distribution parameterised by $\boldsymbol{\theta}$.

Distributions

$\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$	The Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$.
$\Phi(y)$	The standard Gaussian cumulative distribution function.
$W(d, \boldsymbol{D})$	The Wishart distribution with d degrees of freedom and scale matrix $\boldsymbol{D}$.
$W_G(b, \boldsymbol{D})$	The GWishart distribution over graph G, with degrees of freedom parameter b, and scale matrix $\boldsymbol{D}$.
$\chi^2(k)$	The chi-squared distribution with k degrees of freedom.

Conventions

$\boldsymbol{x}$ or $\widetilde{\boldsymbol{x}}$	Vector of covariates (side information).
$\boldsymbol{y}$ or $\widetilde{\boldsymbol{y}}$	Vector of visible (observed) variables.
$\boldsymbol{z}$	Vector of hidden (latent) variables.
$\boldsymbol{\Sigma}$	Covariance matrix of a Gaussian model.
$\boldsymbol{\Lambda}$	Precision matrix of a Gaussian model.
C	Dimensionality of the covariate vector.
V	Dimensionality of the visible vector.
H	Dimensionality of the hidden vector.
N	Number of data points.
M	Number of mixture components.
$\boldsymbol{\theta}^{(t)}$	Parenthesised superscript t is the iteration index.
G	A graph.
$\boldsymbol{G}$	The matrix representation of graph G.

- We use $\widetilde{\boldsymbol{x}}$ to denote the augmentation of $\boldsymbol{x}$ with a unit element.
- Vectors $\boldsymbol{y}$ and $\widetilde{\boldsymbol{y}}$ are usually related by an invertible transformation.
- Lower-case versions of the dimensionality constants are typically used to index sums and products over the respective vectors, e.g. n is used to index a sum over N data points.
- Data sets are stored in matrices where each row is a single data point: $\boldsymbol{X}$ for the covariates, $\boldsymbol{Y}$ for the observed variables, and so on.

Table of Models

Chapters 2 and 3 discuss many related models. For reference purposes, the following table lists all the models mentioned in those chapters, where a checkmark indicates that a model possesses one of the following features.

- **Latents** – The model possesses Gaussian latent variables.
- **Side** – The model incorporates side information (conditioned covariates).
- **Multi-State** – The model has a discrete latent state. This includes mixtures, mixtures of experts, HMMs, and input-output HMMs.

- **Copulas** – The observed variables are non-Gaussian; this is achieved through Gaussian copulas.
- **Temporal** – All the temporal models here have a discrete latent state. The state depends on its value at the previous time step.

Model	Latents	Side	Multi-State	Copulas	Temporal
GLASSO					
MRCE		✓			
SLR	Implicit				
SLICE	✓				
SLICE+Side	✓	✓			
MGLASSO		✓	✓		
MSLICE	✓	✓	✓		
MSLICE-NoSide	✓		✓		
CopGLASSO				✓	
CopMRCE		✓		✓	
CopSLICE	✓			✓	
CopMGLASSO		✓	✓	✓	
CopMSLICE	✓	✓	✓	✓	
TFAC		✓	✓		✓
TGLASSO		✓	✓		✓
TSLICE	✓	✓	✓		✓
CopTGLASSO		✓	✓	✓	✓
CopTSLICE	✓	✓	✓	✓	✓

Note the following.

1. TFAC and TGLASSO share the same features in this table. In fact, TFAC is a special case of TGLASSO in which the transition and mean parameters are unpenalised while the penalty on the precision elements approaches infinity (enforcing diagonal precisions).
2. We have not implemented TSLICE and CopTSLICE. They are straightforward extensions of other models, but we expect they would be slow to train.

Abbreviation Reference

The following table contains each abbreviation used in this thesis, along with a page number of either the first use, or on which more information can be found.

Abbreviation	Page	Meaning
AST	117	Adaptive Simulated Tempering
BDMCMC	87	Birth-Death Markov Chain Monte Carlo
BG-HCC	98	Block Gibbs with Heuristic Clique Cover
BG-MC	98	Block Gibbs with Maximal Cliques
CBF	87	Conditional Bayes Factor
CopGLASSO	36	Copula GLASSO
CopMGLASSO	38	Copula MGLASSO
CopMRCE	69	Copula MRCE
CopMSLICE	38	Copula MSLICE
CopSLICE	36	Copula SLICE
CopTGLASSO	66	Copula TGLASSO
CopTSLICE	80	Copula TSLICE
CRF	6	Conditional Random Field
DLR	98	Dobra-Lenkoski-Rodriguez sampler
ECME	24	Expectation/Conditional Maximisation Either
EM	10	Expectation Maximisation
ESS	100	Effective Sample Size
FA	22	Factor Analysis
GARCH	6	Generalised Autoregressive Conditional Heteroskedasticity
GARCH-X	6	GARCH model incorporating covariates
GGM	81	Gaussian Graphical Model
GLASSO	14	Graphical LASSO
Go-CART	59	Graph-optimised Classification And Regression Trees
GPD	36	Generalised Pareto Distribution
HMC	89	Hamiltonian Monte Carlo
IO-HMM	63	Input-Output Hidden Markov Model
KDE	37	Kernel Density Estimator
KP	57	Kronecker Product
KS	57	Kronecker Sum

Abbreviation	Page	Meaning
LARS	13	Least Angle Regression method for LASSO
LASSO	11	Least Absolute Shrinkage and Selection Operator
MAP	14	Maximum *A Posteriori*
MCMC	85	Markov Chain Monte Carlo
MFA	33	Mixture of Factor Analysers
MGARCH	6	Multivariate GARCH
MGLASSO	38	Conditional Mixture of GLASSO
MH	85	Metropolis-Hastings
MRCE	69	Multivariate Regression with Covariance Estimation
MRF	2	Markov Random Field
MSLICE	29	Conditional Mixture of SLICE
MSLICE-NoSide	45	(Unconditional) Mixture of SLICE
OSA	71	One Step Ahead
(P)PCA	24	(Probabilistic) Principal Component Analysis
QUIC	16	Quadratic Inverse Covariance method for GLASSO
RCA	25	Residual Component Analysis
RMHMC	111	Riemann Manifold HMC
SCAD	14	Smoothly Clipped Absolute Deviation penalty
SLICE	21	Sparse Latent Inverse Covariance Estimator
SLICE+Side	45	Conditional SLICE model
SLR	17	Sparse/Low-Rank decomposition method
TFAC	69	IO-HMM model with factorised Gaussian emissions
TGLASSO	66	Temporal GLASSO model
TSLICE	80	Temporal SLICE model
WL	86	Wang-Li sampler

Chapter 1

Introduction

Learning and exploiting structure in data is a fundamental machine learning problem. Incorporating prior structural knowledge into a model, and learning the model structure from data, are important across many application domains and tasks. For example, in systems biology, one is often interested in learning structure for the purpose of knowledge discovery; in finance, the goal is often prediction of some measure of wealth.

The word "structure" is somewhat vague, and so requires some qualification. In one usage, to say that data is "structured" implies the existence of a sparsely connected graphical model that gives rise to the observations. Conversely, "unstructured" data may be too complex to be accurately represented by a sparse graphical model. This is primarily what we mean by structure in this thesis. Sometimes "structure" refers to the *causal* relationships between a set of variables; this thesis does not address the problem of learning causal relationships.

In real-world problems, data are often structured; that is, we have reason to believe that there exists an underlying sparse graphical model that accurately represents the relationships between the variables. We focus on this type of problem in this thesis. Using a sparse model may provide efficiency advantages: inference may be faster in a sparse model, and fewer data may be required to learn the model parameters. Sparse models are common in machine learning, and include trees, Markov chains, restricted Boltzmann machines, and so on. But if the relationships between the variables are unknown or uncertain, fixing the graph structure prior to learning may result in a trained model that does not accurately represent the data. In such cases, learning a sparse graph structure from data may uncover interesting relationships between the variables, and result in a more efficient model. Therefore, we view structure learning as a probabilistic inference problem in which the underlying graphical model is constrained to

be sparse.

Graph-learning methods may be divided into two classes: those that learn only a graph structure, and those that learn a full probabilistic model. The former may be sufficient if the goal is knowledge discovery, but prediction usually requires a fully specified model. In either case, the key problem is that the number of possible graphs increases exponentially with the number of nodes. So the naive approach of iterating over all possible graphs is feasible only for low-dimensional problems. Nevertheless, standard methods for learning graph structure are based on combinatorial search, often involving heuristics for traversing the space of structures. The goal is typically to find graphs that score highly according to some measure, or satisfy certain conditional independence constraints. See, for example, (Chow and Liu, 1968; Heckerman et al., 1999; Friedman et al., 1999; Silva et al., 2006; Elidan et al., 2007; Maathius and Kalisch, 2009). The majority of such approaches cannot be easily extended to handle latent variables, or may only be used for limited classes of models. Kemp and Tenenbaum (2008) described an extension that selects the best fitting model from several candidate structural forms. Other methods may only be used to learn structures of specific predefined forms, with hard constraints on the number of node parents and their cardinality in a directed network (Zhang, 2004; Harmeling and Williams, 2011). While these approaches are justified when it is known *a priori* that the structure belongs to a standard class of models, they may not be appropriate in a more general real-world setting with richer underlying models. For example, constraining the graph to have low tree width (Bradley and Guestrin, 2010) is not easily justified in finance, where share prices may establish rich dependencies, both within and across market sector boundaries. In some cases an explicit constraint on the structural form of a model can make the inference problem more complex than necessary, and so less scalable to higher dimensions.

However, restricting the model to a multivariate Gaussian makes inference considerably easier, and so this case has been well-studied in the recent literature. The inverse covariance matrix – called the precision matrix – is straightforwardly related to the model's Markov random field (MRF): zeros in the precision matrix correspond to missing edges in the MRF. Thus, in the case of a multivariate Gaussian, structure learning can be seen as the problem of learning a sparse precision matrix. This problem is a central feature of this thesis. There are many potential applications of sparse Gaussian modelling across a variety of domains. To give one example, Krumsiek et al. (2011) address the problem of reconstructing a metabolic reaction network by fitting

a sparse Gaussian model to a data set consisting of reactant concentrations measured intermittently over a period of time. They show that many of the strongest edges in the learned model correspond to known pathway interactions.

In this thesis, we shall focus primarily on problems from the financial domain. Since financial problems often involve prediction, we shall be interested in learning full predictive models, as opposed to the graphical model structure only. Covariance is often used as a measure of risk in finance, so obtaining an accurate estimate of a covariance matrix is important in many financial scenarios. For example, the portfolio selection problem entails that wealth be distributed among a set of assets such as to optimise investor utility – which often means keeping risk low while maximising returns. In financial risk management, a company may wish to know how changes in market conditions – and extreme changes in particular – might affect its investments. In this case, it is desirable to know how the covariance, and therefore the risk, responds to changes in the market.

In situations where data are scarce relative to the number of parameters in a chosen model, the data alone may be insufficient to accurately learn the parameters. Regularisation – the imposition of prior knowledge – may be necessary to learn an adequate model. Shrinkage is a form of regularisation in which a statistical estimator is combined with a component that encourages the estimate towards a supplied value. For example, Ledoit and Wolf (2003) estimate the covariance of the returns of a set of stocks by shrinking the empirical covariance towards an estimate produced by a simple one-factor model.

Sparsity may be imposed during optimisation of model parameters by introducing a term to shrink the parameters towards zero. Treating structure learning as an optimisation problem rather than a combinatorial search can reduce computation time. Various types of shrinkage penalty with different properties have been studied. For example, in bridge regression (Frank and Friedman, 1993), the objective function consists of the likelihood and a regularisation term of the form $\gamma \sum_{c=1}^{C} |\beta_c|^{\alpha}$, where $\{\beta_c\}$ are the regression parameters, γ is the strength of the shrinkage, and α determines the properties of the regulariser. If $\alpha \geq 1$, the optimisation problem is convex; the particular case of $\alpha = 2$ corresponds to ridge regression (Hoerl and Kennard, 1970). If $\alpha \leq 1$, the learned parameter vector $\boldsymbol{\beta}$ may be sparse. The case $\alpha = 1$ corresponds to LASSO regression (Tibshirani, 1996), and has both properties: the optimisation problem is convex, and the solutions may be sparse. Regularisation terms of this form – called $L1$ penalties – feature strongly in Chapters 2 and 3 of this thesis.

There has been much recent much work on learning sparse structures of Gaussian models. See, for example, (Meinshausen and Bühlmann, 2006; Levina et al., 2008; Lake and Tenenbaum, 2010). One particularly popular approach has been to maximise the likelihood of a data set while imposing $L1$ penalties on the parameters. (Since the $L1$ penalties can be viewed as coming from a Laplace prior, the result is the maximum *a posteriori* (MAP) solution). This problem is often known as the graphical lasso. Algorithms for solving it have become progressively faster in recent years; see, for example (Banerjee et al., 2008; Friedman et al., 2008; Duchi et al., 2008; d'Aspremont et al., 2008; Scheinberg et al., 2010; Hsieh et al., 2011; Rolfs et al., 2012; Hsieh et al., 2013).

Our work (Agakov et al., 2012), presented primarily in Chapter 2, is motivated by a number of observations. First, use of the graphical lasso model makes the assumption that the data is complete – that is, there are no missing observations or latent factors influencing the relations between the modelled variables. However, hidden or missing variables are common in real-world applications. We observe that the underlying structure in a data set may only become apparent in an augmented space of observed and latent variables. That is, an accurate model of the data may need to be dense if only the observed variables are included in the model; applying sparsity constraints may prune important relationships and result in misleading representations of underlying structure. But incorporating latent variables into the model may allow a sparser, more parsimonious model, while retaining accuracy. Note that sparseness of the joint structure is a standard feature of many commonly used latent variable models – for example latent trees, hidden Markov models, and restricted Boltzmann machines – all of which may give rise to dense marginal structures.

The second observation is that in many real-life applications, structural dependencies between variables are rarely homogeneous for all data points and may often depend on poorly understood latent states. In finance, for example, dependencies between asset returns may vary strongly according to market conditions. A model with a single, fixed graph structure may be inadequate to capture such variation.

The third observation is that, in practice, we may have access to a vector of side information (covariates): a set of observed variables whose distribution is of no interest, but which may be predictive of changes to the model. Such a vector may consist of features constructed by a domain expert, and may be very high-dimensional. If the model is being used for decision support, it may be desirable to understand which features of this covariate vector are most predictive of changes in the model. In finance, for

example, a transition to a new market regime may be influenced by a complex mixture of investor sentiment, rumour, news announcements, econometric technical indicators, and macro-economic factors – and a fund manager may wish to know which of these are most predictive of the transitions. Liu et al. (2010) incorporate side information into the graphical lasso: they learn a partition of the covariate space, and train a different graphical lasso model in each region. Rothman et al. (2010) utilise covariates in their linear regression model with sparse Gaussian noise; they learn the regression parameters jointly with the Gaussian precision, the latter trained by graphical lasso.

Our fourth observation is simply that a Gaussian will often be a poor model of real-world data. It has long been known that financial returns data are non-Gaussian; see, for example (Fama, 1965). One way to handle this is to decouple the marginal distributions of the observed variables from the dependency structure. A simple distribution is used for the dependence structure, while more flexible distributions are used for the marginals. Since one-dimensional distributions are relatively straightforward to work with, the more complex, flexible distributions may still be fitted quickly and accurately. Copulas are the mechanism that allow this decoupling of the marginals and the dependence structure. We describe copulas in more detail in Section 2.1.4. If a multivariate Gaussian is used for the dependence model, then the copula is known as a Gaussian copula. Copula methods, and the Gaussian copula in particular, have recently found wide use in the financial domain. Genest et al. (2009) conducted a bibliometric survey, and found a rapid increase in the publication rate of literature on copulas in finance from 1999. They found that copula applications in finance were spread roughly evenly across four categories: risk management, portfolio management, derivatives pricing, and risk measurement. For more detail on the use of copulas in finance, especially regarding credit risk and derivatives pricing, see the book by Cherubini et al. (2004). Patton (2009) gives a brief but broad review of copula methods for financial time series. Active research continues in the four categories discussed by Genest et al. (2009); vine copulas – in which a multivariate copula is constructed from a set of bivariate copulas – are currently a popular topic; and the use of copulas to study the dependence and co-movements of financial variables is now common; see, for example, (Reboredo, 2011; Brechmann et al., 2012; Czado et al., 2013; Low et al., 2013; Boubaker and Sghaier, 2013; Wang et al., 2013; Aloui et al., 2013).

Common methods for high-dimensional sparse structure learning either ignore the four observations discussed above – as is the case for fully observed sparse Gaussian MRFs – or address only one of them in a manner that is not easily extensible; see, for

example, (Liu et al., 2009; Chandrasekaran et al., 2010). Motivated by real-world problems, in Chapter 2 we extend existing approaches to describe a sparse discriminative mixture of sparse latent Gaussian copulas. The model can be used to learn multiple interpretable latent variable structures with non-Gaussian marginals, each corresponding to a unique unknown state, and to identify explanatory features useful for predicting structural changes.

The work described in Chapter 3 is motivated by some additional observations. In many real-world problems – in particular in the financial domain which motivates much of our work – the relationships between the variables may be expected to persist in time. Furthermore, covariates may provide information about changes to the model structure. We therefore extend the stationary models developed in Chapter 2 to input-output hidden Markov models (IO-HMMs) (Bengio and Frasconi, 1995).

Alternatively, the Chapter 3 work can be motivated by viewing it as a standalone work on temporal modelling. IO-HMMs, along with dynamic extensions (Sutton et al., 2007) of conditional random fields (CRFs) (Lafferty et al., 2001), are two popular conditional methods for time series. In practice, however, these models either ignore constraints on the structural sparsity (which gives rise to uninterpretable inferences), or much more commonly over-constrain the models to have relatively trivial dependence structures (in which case there is little interesting structure to discover). For example, the common applications of dynamic CRFs make factorial assumptions about the distributions of the outcomes (Sutton et al., 2007; Sokolovska et al., 2010), while IO-HMMs commonly exploit uncorrelated Gaussians as emission distributions (Bengio et al., 2001; Ernst et al., 2007). We extend the latter approaches to incorporate flexible, interpretable emission structures that are learned from data.

In finance, a popular thread of research has focussed on extending autoregressive and moving-average models for the means of the observations to their variances (Engle, 1982). This gave rise to a broad family of generalised autoregressive conditional heteroskedasticity (GARCH) methods (Bollerslev, 1986; Bollerslev et al., 1994) for tracking variance changes in returns of financial assets. Most of the applications of such methods have focused on modelling univariate observations, though multivariate extensions (MGARCH) have been proposed and are increasingly used in quantitative finance (Bauwens et al., 2006). Some GARCH methods – often named GARCH-X – make use of side information; see, for example, (Fleming et al., 2008; Han, 2010) and references therein. Often, the covariate vector is low-dimensional, but modern methods may incorporate high-dimensional covariates and feature selection; see (Su-

carrat et al., 2013), for example. Our IO-HMM model described in Chapter 3 permits high-dimensional side information to influence the changes in structure over time.

In Chapters 2 and 3, our methods compute a MAP solution to estimate model parameters. Bayesian methods may offer a more principled approach, but they are often considered to be much slower than optimisation methods. However, Mohamed et al. (2012) recently compared the $L1$ approach with Bayesian methods based on the "spike-and-slab" prior, focussing on unsupervised linear latent variable models. They found that the Bayesian methods could outperform $L1$, even when both were constrained by the same time budget. In Chapter 4, we use a Bayesian model based on the GWishart distribution to address the question of whether a Bayesian method can outperform the $L1$ optimisation approach to infer sparse precision matrices.

The GWishart distribution generalises the Wishart such that draws from the distribution respect a given graph structure; see section 4.1.2. A class of sparse Bayesian Gaussian graphical models based on the GWishart has been under development in parallel with the graphical lasso. See, for example, (Roverato, 2002; Atay-Kayis and Massam, 2005; Carvalho and West, 2007; Wang and Carvalho, 2010; Dobra et al., 2010; Mitsakakis et al., 2011; Lenkoski and Dobra, 2011; Dobra et al., 2011; Wang and Li, 2012). Inference in these models is often limited by the efficiency with which the GWishart can be sampled. Wang and Li (2012) demonstrate that the block Gibbs sampler is currently state-of-the-art for this task. A more efficient sampler would make inference in GWishart-based models faster, and could make practical the use of more complex, higher-dimensional models. Hamiltonian Monte Carlo (HMC) samplers – described in section 4.1.4 – can facilitate fast mixing in distributions where the random variables are strongly coupled. Furthermore, they naturally take advantage of sparsity: the bottleneck in HMC is often the computation of the energy gradient with respect to the distribution parameters. Fewer parameters means fewer gradients to evaluate. In Chapter 4, we develop an HMC approach to sample from the GWishart distribution.

The thesis is structured as follows. Each chapter begins with a review of the relevant background literature. In Chapter 2, we make a number of extensions to the graphical lasso model, culminating in a mixture of Gaussian copula experts. We apply the new models to real-world financial returns data, and compare test log likelihoods with competing models. We also investigate the interpretability of the learned structures. In Chapter 3, we develop IO-HMM extensions of our models, and investigate the degree to which they improve performance on financial returns data. We also study the importance of the side information in this chapter. In Chapter 4, we introduce an

HMC approach for sampling the GWishart distribution, and compare its efficiency with the block Gibbs sampler. We use the new sampler within a sparse Bayesian model in which joint samples of the graph structure and precision matrix are required, and compare the Bayesian approach with the graphical lasso when both models have the same time budget. We summarise the thesis contributions and conclude in Chapter 5.

Chapter 2

$L1$-Penalised Latent Gaussian Models

In Section 1, we discussed the close relationship between sparsity and structure. In the particular case of a multivariate Gaussian model, the structure can be read directly from the inverse covariance (precision) matrix: edges in the model's Markov Random Field (MRF) are present if and only if the corresponding entry in the precision matrix is non-zero. So a sparse estimator of the precision matrix is also an estimator of the MRF.

Shrinkage estimators are a class of estimators characterised by the combination of any estimator with a component that encourages the parameter estimate towards a supplied value. Sparse estimators may be designed by shrinking parameters towards zero. In this chapter, we study the application of such estimators to learning sparse precision matrices in Gaussian models.

Existing methods focus on models in which all variables are observed. In practice, latent variables may be present that confound the relationships between the observed variables. In some scenarios, the structure in the joint space of the observed and latent variables may be simpler than in the visible space alone. In this chapter, we introduce a method, which we call SLICE, for learning a sparse precision matrix of a Gaussian model with latent variables.

With practical applications in mind, especially to the financial domain, we extend the basic SLICE method in various ways. The structure of the asset returns in a financial market is known not to be fixed; in particular, this structure depends on market volatility. We extend SLICE to a mixture model to capture variation of structure, and we utilise side information – the market volatility in this example – to provide information on which structure is in use for each data point. Thus SLICE is extended to a conditional mixture of sparse latent Gaussians, which we call MSLICE. Finally, be-

cause real-world data such as asset returns are often poorly modelled by a Gaussian, we extend SLICE and MSLICE to handle this case. We learn the marginals of the observed variables from data, and use these to transform each variable into a new variable with Gaussian marginal, and then apply the existing sparse Gaussian methods. For SLICE, the marginals can be learned in a preprocessing step, but for MSLICE, the marginals must be trained simultaneously with the rest of the model.

The chapter is organised as follows. Section 2.1 presents the background material. This begins with the $L1$-penalisation approach to shrinkage, and in particular the problem of sparse precision estimation in a Gaussian model. In Section 2.1.3, we explain why it may be necessary to take latent variables into account when learning structure, and then describe an existing method that does so, albeit with an implicit representation of the latent variables. Background material on copulas and their application to sparse structure learning is presented in Sections 2.1.4 and 2.1.5.

Section 2.2 is devoted to the SLICE method. We discuss our motivations, then present the estimator itself, and the expectation maximisation (EM) algorithm for computing it. In Section 2.3.1, we present the mixture of experts extension, MSLICE, followed by its EM algorithm. We describe non-Gaussian extensions of these models, utilising copulas, in Section 2.4.

Section 2.5 contains an evaluation of the new methods, and a comparison with existing methods. In Section 2.5.2, we compare single-component (non-mixture) methods, demonstrating that the inclusion of latent variables can lead to a more parsimonious model, and that the non-Gaussian methods perform better than the methods they extend. Section 2.5.3 evaluates the mixture models. We show that the mixtures perform better than non-mixtures (as expected), and that the model learns to use the side information to select the most appropriate structure for each data point. We illustrate that the mixture components may be interpreted in terms of the side, and that the learned structures may be explainable with some domain knowledge.

We draw conclusions and discuss future work in Section 2.6.

2.1 Background

2.1.1 Shrinkage Methods

Our interest in shrinkage methods is primarily in their application to learning sparse precision matrices. But these methods were applied to regression problems before the

recent surge in popularity of sparse precision estimation. The maximum likelihood estimator for the parameters in a regression problem may have undesirable properties, especially if the data set is small. In particular, with C predictor variables and a data set of size $N < C$, the linear regression problem is under-defined, and there are infinitely many likelihood-equivalent solutions. Regularisation may be introduced to address this problem; one form of regularisation is to augment the objective with a penalty term which depends on the parameter settings.

Hoerl and Kennard (1970) introduced ridge regression, in which an $L2$ penalty augments the maximum likelihood objective for the linear regression model with independent and identically distributed (iid) Gaussian noise. Frank and Friedman (1993) introduced a generalisation known as bridge regression. Given a matrix of predictors $\boldsymbol{X} \in \mathbb{R}^{N \times C}$ and a vector of targets $\boldsymbol{y} \in \mathbb{R}^N$, the bridge regression estimator of the coefficients $\boldsymbol{\beta} \in \mathbb{R}^C$ is

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \left[\|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2 + \gamma \sum_{c=1}^{C} |\beta_c|^\alpha \right], \tag{2.1}$$

where $\gamma \in \mathbb{R}_+$ is the magnitude of the penalty, and $\alpha \in \mathbb{R}_+$ determines the properties of the penalty. Clearly, the penalty term favours parameters $\boldsymbol{\beta}_c$ that are closer to zero.

Three particular cases for α are worthy of note.

1. $\alpha \to 0$. In this limit, non-zero parameters are penalised, but the magnitude of the penalty is otherwise independent of the parameter value. This case is called subset selection. The size of the selected subset depends on γ.

2. $\alpha = 1$. The $L1$ norm of $\boldsymbol{\beta}$ is penalised. This case is the Least Absolute Shrinkage and Selection Operator (LASSO) of Tibshirani (1996).

3. $\alpha = 2$. The $L2$ norm of $\boldsymbol{\beta}$ is penalised. This case corresponds to ridge regression.

To understand the differences between these cases, it is useful to rewrite the optimisation problem (2.1) in the following equivalent form:

$$\hat{\boldsymbol{\beta}} = \arg\min_{\boldsymbol{\beta}} \|\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}\|_2^2, \tag{2.2}$$

subject to the constraint

$$\sum_{c=1}^{C} |\beta_c|^\alpha \leq t, \tag{2.3}$$

where $t \in \mathbb{R}_+$ is a function of γ. Figure 2.1 illustrates the shape of the constraint set for different values of α. The constraint set is convex for $\alpha \geq 1$. Convexity is a desirable

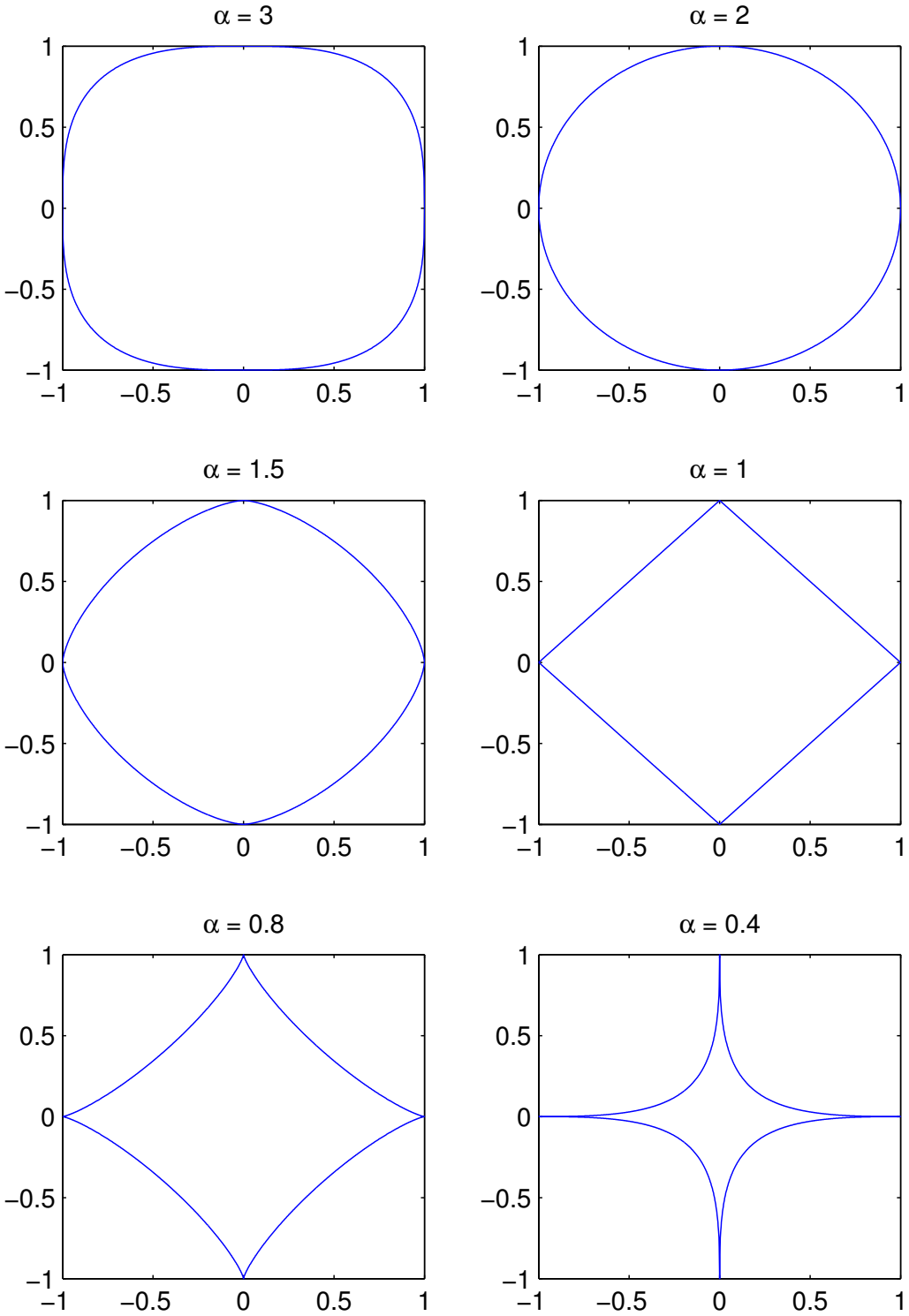
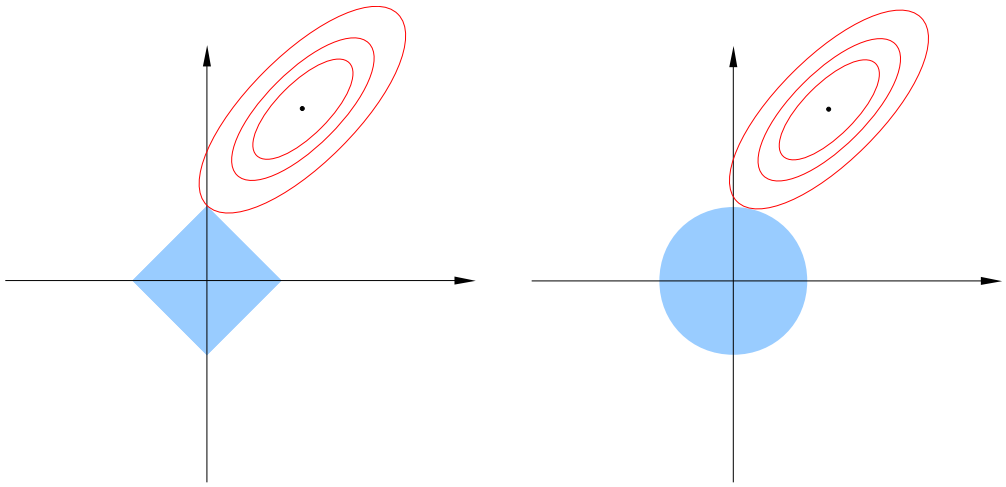


Figure 2.1: Constraint set boundaries in bridge regression for different values of α. The sets for which $\alpha \leq 1$ produce sparse solutions, but a constraint set is only convex if $\alpha \geq 1$.



(a) Lasso: $\alpha = 1$. (b) Ridge: $\alpha = 2$.

Figure 2.2: Visualisation of lasso and ridge regression. Blue regions are the constraint sets; red ellipses are contours of the maximum likelihood objective. The solution is where the contour touches the constraint set. For the lasso, this is on an axis, so one of the parameters is zero. This is possible because the constraint boundary is not smooth. For the ridge, that parameter is small, but it is not exactly zero. This diagram is adapted from Figure 3.11 (Hastie et al., 2009).

property because it means that the optimisation problem has a global maximum.

As discussed in Section 1, sparsity may also be a desirable property of the solution. In bridge regression, setting $\alpha \leq 1$ can potentially lead to a solution with many zeros; but with $\alpha > 1$, zeros in the solution are rare. This is because the constraint boundary is smooth where it intersects the axis when $\alpha > 1$, but non-smooth when $\alpha \leq 1$. Figure 2.2 illustrates why non-smoothness can lead to sparse solutions by comparing lasso and ridge.

The lasso is particularly interesting because it is the only estimator in the bridge family that has both properties: it is convex and results in sparse solutions. It can be computed by least angle regression (LARS) (Efron et al., 2004), or by coordinate descent methods such as that implemented by the glmnet software (Friedman et al., 2010b). However, depending on the application, properties of a shrinkage penalty beyond those of $L1$ may be desirable, and so various lasso extensions and modifications have been studied in the literature. For example, the elastic net (Zou and Hastie, 2005) was designed to address the following shortcomings of the lasso.

1. When $C > N$, the lasso selects at most N predictors.

2. If there is a group of highly correlated predictors, the lasso tends to select only one of them, and does not care which.

3. When $N > C$ and there are high correlations between predictors, Tibshirani (1996) observes that ridge outperforms lasso.

The elastic net addresses these issues by generalising the lasso to include an additional ridge penalty, plus a rescaling after optimisation to remove some of the shrinkage. The empirical results of Zou and Hastie (2005) indicate that the elastic net improves the prediction performance of the lasso. In the group lasso (Yuan and Lin, 2006), predictors are partitioned into groups, and penalties applied which generalise both the lasso and ridge penalties. These penalties encourage sparsity at the group level, but not within groups; that is, entire groups of variables are pruned or retained simultaneously. Fan and Li (2001) introduce the smoothly clipped absolute deviation (SCAD) penalty, which behaves similarly to $L1$ on small parameters, but applies less shrinkage to parameters with greater magnitude. The resulting estimator has reduced bias compared to the lasso, but the optimisation problem is non-convex.

The use of $L1$ penalties to learn sparse solutions is not limited to regression: the concepts in this section extend naturally to other problems. In particular, our interest is to use them to learn sparse precision matrices of Gaussian models. We discuss this problem in the next section.

The optimisation problem (2.1) is equivalent to finding the maximum *a posteriori* (MAP) solution of a Gaussian model with a prior of the form

$$p(\boldsymbol{\beta}) \propto \exp\left(-\gamma \sum_{c=1}^{C} |\beta_c|^{\alpha}\right), \tag{2.4}$$

for $\gamma, \alpha \in \mathbb{R}_{++}$. (Recall that C is the number of predictors). So lasso regression is the MAP solution with independent Laplace priors on the elements of $\boldsymbol{\beta}$. (The ridge uses a Gaussian prior). Note that the Laplace prior itself does not lead to sparsity in the posterior; it is only in combination with MAP estimation that this prior results in sparse solutions. In Chapter 4, we look at fully Bayesian methods for modelling and inferring sparse precisions.

2.1.2 Graphical Lasso

The precision matrix of a Gaussian model determines its Markov random field (MRF): zeros in the precision correspond to missing edges in the MRF. This can easily be seen

by writing the Gaussian density as a product over nodes and edges. If random vector $\underline{\boldsymbol{y}} \sim \mathcal{N}\left(\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}\right)$, the density

$$p(\boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{y}-\boldsymbol{\mu})^T \boldsymbol{\Lambda}(\boldsymbol{y}-\boldsymbol{\mu})\right) \tag{2.5}$$

$$= \left[\prod_i \exp\left(-\frac{1}{2}\Lambda_{ii}(y_i-\mu_i)^2\right)\right]\left[\prod_{i\neq j} \exp\left(-\frac{1}{2}\Lambda_{ij}(y_i-\mu_i)(y_j-\mu_j)\right)\right]. \tag{2.6}$$

Since $p\left(y_i, y_j | \boldsymbol{y} \setminus \{y_i, y_j\}\right) \propto p(\boldsymbol{y})$, then clearly y_i and y_j are conditionally independent given all the other variables if and only if $\Lambda_{ij} = 0$. Thus we can learn the MRF by learning a sparse precision matrix.

One approach to learning a sparse precision is to use a shrinkage estimator. Assume the mean $\boldsymbol{\mu}$ is known, or has already been estimated. Now consider the following estimator for the precision:

$$\hat{\boldsymbol{\Lambda}} = \arg\max_{\boldsymbol{\Lambda}} \left[\log\det\boldsymbol{\Lambda} - \operatorname{tr}(\boldsymbol{S}\boldsymbol{\Lambda}) - \sum_{i,j}\Gamma_{ij}|\Lambda_{ij}|\right], \tag{2.7}$$

where $\boldsymbol{S}$ is the sample covariance and $\Gamma_{ij} \in \mathbb{R}_+$. The first two terms comprise the log likelihood, while the final term specifies the $L1$ penalties. The log determinant is a concave function, so the log likelihood is concave. In a similar manner to Equations (2.2, 2.3), this optimisation can be written as the maximisation of a concave function (or the minimisation of a convex function) over a convex set; so the optimisation problem is convex. The penalty matrix $\boldsymbol{\Gamma}$ may be set using prior knowledge. In practice, the entries of $\boldsymbol{\Gamma}$ are typically assumed to be equal, leaving a single parameter that is chosen by cross-validation over some plausible range.

Meinshausen and Bühlmann (2006) introduce a method for learning only the edge set; that is, they learn which elements of $\boldsymbol{\Lambda}$ are non-zero, but not the values of those elements. They do this by solving an approximate version of problem (2.7): they perform a lasso regression of each variable on all the rest. Edge (i, j) is included in the final graphical model if the lasso regressions find a dependence of y_i on y_j or y_j on y_i. (A variant requires the dependence to be present in both directions). Friedman et al. (2010a) introduce two symmetrised versions of this method to learn $\boldsymbol{\Lambda}$ (not just the edge set). The first is called the symmetric lasso; it solves problem (2.7) with the likelihood replaced by a pseudo-likelihood. The second is called the paired group lasso; it regresses each variable on the rest, but applies a group lasso penalty to pairs consisting of the regression coefficient of y_i on y_j and of y_j on y_i.

Banerjee et al. (2008) derive the dual of problem (2.7), and introduce a block-wise interior-point method for solving it. Provided $\mathbf{\Lambda}$ is initialised to a positive-definite matrix, it is guaranteed to remain positive-definite, and the algorithm converges to a globally optimal solution. Banerjee et al. (2008) show that their optimisation method is equivalent to an iterative application of lasso regression, but they do not perform the optimisation this way. Friedman et al. (2008) do take this approach, and gain a substantial increase in efficiency. Friedman et al. (2008) call their method the graphical lasso, which we often abbreviate to GLASSO. The problem (2.7) is now sometimes referred to by the same name, and we adhere to this convention.

Considerable research has since been devoted to the graphical lasso, mostly in an effort to obtain the solution more efficiently. Duchi et al. (2008) use a projected gradient method for solving the dual problem, while Schmidt et al. (2009) improve on this method by using a projected quasi-Newton technique. Scheinberg et al. (2010) use an alternating linearisation method. At present, the fastest reported method appears to be that of Hsieh et al. (2011) whose quadratic inverse covariance (QUIC) algorithm is based on Newton's method.

2.1.3 Structure Learning in a Latent Gaussian Model

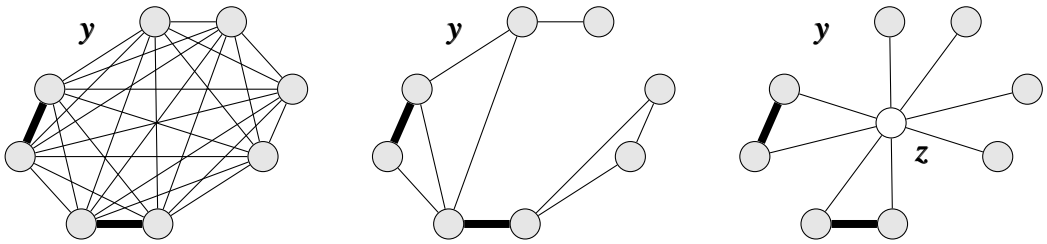
In a broad range of real-world applications – in finance or systems biology, for example – it is common for some variables to be hidden or missing. Consider a Gaussian model in which some of the variables are unobserved:

$$\underline{\boldsymbol{u}} \sim \mathcal{N}(\boldsymbol{\mu}, \mathbf{\Sigma}), \tag{2.8}$$

where $\underline{\boldsymbol{u}} = \left(\underline{\boldsymbol{y}}^T, \underline{\boldsymbol{z}}^T\right)^T$; $\underline{\boldsymbol{y}}$ is observed; and $\underline{\boldsymbol{z}}$ is latent. Label the blocks of the precision and covariance according to the partition of $\underline{\boldsymbol{u}}$:

$$\mathbf{\Lambda} = \begin{pmatrix} \mathbf{\Lambda}_{yy} & \mathbf{\Lambda}_{yz} \\ \mathbf{\Lambda}_{zy} & \mathbf{\Lambda}_{zz} \end{pmatrix}; \quad \mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{yy} & \mathbf{\Sigma}_{yz} \\ \mathbf{\Sigma}_{zy} & \mathbf{\Sigma}_{zz} \end{pmatrix}. \tag{2.9}$$

The notion of structuredness of data is tightly linked to the sparsity of the underlying data representations in the joint space of $\underline{\boldsymbol{u}}$, rather than sparsity in the data space of $\underline{\boldsymbol{y}}$ alone. Setting sparsity constraints only on the marginal precision $\mathbf{\Sigma}_{yy}^{-1}$ may lead to the pruning of important regularities, and result in a potentially misleading representation of the underlying structure. This is illustrated in Figure 2.3. In Section 2.2.1, we introduce a method for learning a sparse $\mathbf{\Lambda}$. That is, we apply sparsity in the joint space of the visible and latent variables.



(a) Dense marginal. (b) Sparse marginal. (c) Sparse joint.

Figure 2.3: (a) Fully observed model $p(\boldsymbol{y})$ with a dense structure. Thick lines indicate stronger pairwise potentials. (b) The learned structure of a fully observed model with a sparsity constraint. Important links have been pruned, resulting in extraneous conditional independences. (c) The learned structure of a sparse latent model $p(\boldsymbol{y},\boldsymbol{z})$ with sparsity applied in the joint space. The joint $p(\boldsymbol{y},\boldsymbol{z})$ is sparse, but the marginal $p(\boldsymbol{y})$ may still be dense because the latent variable couples the visible variables. But note that, in contrast to latent factor models, the conditional $p(\boldsymbol{y}|\boldsymbol{z})$ may contain residual couplings.

2.1.3.1 Sparse/Low-Rank Decomposition of a Latent Gaussian Model

In the approach of Chandrasekaran et al. (2010), the latent variables are implicit. They note that the precision of the marginal model can be written in terms of the joint precision as follows:

$$\boldsymbol{\Sigma}_{yy}^{-1} = \boldsymbol{\Lambda}_{yy} - \boldsymbol{\Lambda}_{yz}\boldsymbol{\Lambda}_{zz}^{-1}\boldsymbol{\Lambda}_{zy}. \tag{2.10}$$

They impose sparsity on $\boldsymbol{\Lambda}_{yy}$. The product $\boldsymbol{\Lambda}_{yz}\boldsymbol{\Lambda}_{zz}^{-1}\boldsymbol{\Lambda}_{zy}$ is not required to be sparse. However, if there are fewer hidden variables than visible, this product will have low rank. Hence, the marginal precision is decomposed into the sum of a sparse matrix and a low-rank matrix.

Chandrasekaran et al. (2010) propose the following estimator for the sparse matrix $\boldsymbol{M} = \boldsymbol{\Lambda}_{yy}$ and low-rank matrix $\boldsymbol{L} = \boldsymbol{\Lambda}_{yz}\boldsymbol{\Lambda}_{zz}^{-1}\boldsymbol{\Lambda}_{zy}$:

$$\left(\hat{\boldsymbol{M}},\hat{\boldsymbol{L}}\right) = \underset{(\boldsymbol{M},\boldsymbol{L})}{\arg\min}\left[-\mathcal{L}(\boldsymbol{M}-\boldsymbol{L};\boldsymbol{S}) + \lambda\left(\gamma\sum_{i,j}\left|M_{ij}\right| + \operatorname{tr}(\boldsymbol{L})\right)\right] \tag{2.11}$$

$$\text{such that} \quad \boldsymbol{M}-\boldsymbol{L} \succ 0, \quad \boldsymbol{L} \succeq 0, \tag{2.12}$$

where $\mathcal{L}(\boldsymbol{K};\boldsymbol{S}) = \log\det\boldsymbol{K} - \operatorname{tr}(\boldsymbol{S}\boldsymbol{K})$, and $\boldsymbol{S}$ is the sample covariance. $L1$ penalties are used to sparsify $\boldsymbol{M}$, while the nuclear norm is used to encourage $\boldsymbol{L}$ to have low rank. If it is necessary to find $\boldsymbol{\Lambda}_{yz}$ and $\boldsymbol{\Lambda}_{zz}$, an additional factorisation method would need to be applied to $\hat{\boldsymbol{L}}$.

Chandrasekaran et al. (2010) use two off-the-shelf solvers to find solutions to problem (2.11). They use SDPT3 (Toh et al., 1999) when the dimensionality of $\boldsymbol{y}$ is small, but find LogdetPPA (Wang et al., 2010) more efficient when the dimensionality is higher. SDPT3 is a general-purpose semidefinite programming solver; LogdetPPA is a more specialised software package. Our SLICE estimator, described in Section 2.2.1, also utilises these packages, and we discuss them further in that section.

In the rest of this chapter, we refer to the approach of Chandrasekaran et al. (2010) as the sparse/low-rank decomposition (SLR).

2.1.4 Copulas

A copula is the joint cumulative distribution function (cdf) of a set of random variables whose marginals are uniform on the interval $[0,1]$. Specifically, let $\underline{u}_v \sim \text{Unif}(0,1), 1 \leq v \leq V$, and define $C(u_1,\ldots,u_V) \equiv P(\underline{u}_1 \leq u_1,\ldots,\underline{u}_V \leq u_V)$. Then, C is called a copula function.

Now consider a V-dimensional random vector $\underline{\widetilde{\boldsymbol{y}}}$. Write $\underline{\widetilde{y}}_v$ for element v of $\underline{\widetilde{\boldsymbol{y}}}$. Let F_v denote the marginal cdf of $\underline{\widetilde{y}}_v$. It is well-known that $F_v\left(\underline{\widetilde{y}}_v\right) \sim \text{Unif}(0,1)$. Sklar's theorem states that there exists a copula C such that

$$P_{\widetilde{y}}(\widetilde{y}_1,\ldots,\widetilde{y}_V) = C\left(F_1\left(\widetilde{y}_1\right),\ldots,F_V\left(\widetilde{y}_V\right)\right), \tag{2.13}$$

where $P_{\widetilde{y}}$ is the joint cdf of $\underline{\widetilde{\boldsymbol{y}}}$. Sklar's theorem also states that the reverse is true: given any marginals $\{F_v\}_{v=1}^V$ and any copula function $C : [0,1]^V \to [0,1]$, the function $P_{\widetilde{y}}$ defined by Equation (2.13) is a valid cdf that has $\{F_v\}$ as marginals.

A useful feature of the copula construction is that it allows the marginals of a set of random variables to be decoupled from their dependence structure. This means that dependence can be studied independently of the marginals. But it also makes for flexible modelling: we can choose any model for the marginals (including different models for each random variable), and combine these with any copula to form a multivariate model. In particular, we may combine a simple copula with more complex marginal models – because learning a univariate model is typically easier than learning a high-dimensional model. For example, a kernel density estimator (KDE) may be sufficient for the marginals, but a KDE may be inappropriate for the joint model because KDEs require lots more data in high dimensions.

The Gaussian copula is one of the most popular and well-studied copulas. It is widely used in finance, for example. Let C_Λ denote the Gaussian copula function with

precision matrix $\mathbf{\Lambda}$. It is defined as follows:

$$C_\Lambda(u_1,\ldots,u_V) \equiv \Phi_\Lambda\left(\Phi^{-1}(u_1),\ldots,\Phi^{-1}(u_V)\right), \tag{2.14}$$

where Φ is the univariate Gaussian cdf with zero mean and unit covariance, and Φ_Λ is the multivariate Gaussian cdf with precision $\mathbf{\Lambda}$. So the Gaussian copula can be used to model the dependence structure of $\underset{\sim}{\widetilde{y}}$ by combining it with any set of marginal models $\{F_v\}_{v=1}^V$, as follows:

$$P_{\widetilde{y}}(\widetilde{y}_1,\ldots,\widetilde{y}_V) = \Phi_\Lambda\left(\Phi^{-1}(F_1(\widetilde{y}_1)),\ldots,\Phi^{-1}(F_V(\widetilde{y}_V))\right). \tag{2.15}$$

For additional theory and applications of copulas, there are many sources. One such is the book by Nelsen (2006).

Copulas are increasingly popular in the machine learning literature. Snelson et al. (2004) introduce the warped Gaussian process in which a Gaussian process is augmented with non-linear warping functions on the outputs. Wilson and Ghahramani (2010) define copula processes, and then study a particular example – the Gaussian copula process – which is a type of warped Gaussian process. Elidan uses copulas to parameterise the conditional distributions in Bayesian networks (Elidan, 2010, 2012a,b). In the next section, we describe another recent use of copulas in which they are combined with the graphical lasso.

2.1.5 The Nonparanormal Distribution

Define $f(\widetilde{\mathbf{y}}) \equiv (f_1(\widetilde{y}_1),\ldots,f_V(\widetilde{y}_V))$. If there exists an f such that $f\left(\underset{\sim}{\widetilde{y}}\right) \sim \mathcal{N}(\boldsymbol{\mu},\mathbf{\Sigma})$, then $\underset{\sim}{\widetilde{y}}$ is said to have a nonparanormal distribution (Liu et al., 2009). If the f_v are required to be monotone and differentiable, then they can be written

$$f_v(\widetilde{y}_v) = \mu_v + \sigma_v \Phi^{-1}(F_v(\widetilde{y}_v)), \tag{2.16}$$

and the associated density is

$$p(\widetilde{\mathbf{y}}) = \frac{1}{(2\pi)^{\frac{V}{2}}|\mathbf{\Sigma}|^{\frac{1}{2}}} \exp\left\{-\frac{1}{2}(f(\widetilde{\mathbf{y}})-\boldsymbol{\mu})^T \mathbf{\Sigma}^{-1}(f(\widetilde{\mathbf{y}})-\boldsymbol{\mu})\right\} \prod_{v=1}^V \left|f_v'(\widetilde{y}_v)\right|. \tag{2.17}$$

The parameters μ_v and σ_v are not identifiable. One may choose them, for example, such that $\widetilde{y}_v$ and $f_v(\widetilde{y}_v)$ have the same moments. If we set $\mu_v = 0$ and $\sigma_v = 1$, then by comparison with Equation (2.14), we see that $F\left(\underset{\sim}{\widetilde{y}}\right)$ is distributed as a Gaussian copula.

The nonparanormal parameters are estimated in two stages. First, the marginal estimator $\widehat{F}_v$ is set to the empirical distribution of $\underset{\sim}{\widetilde{y}}_v$, truncated at both tails. Then, a sparse precision is estimated for the Gaussian copula by applying the graphical lasso to the transformed data set $\{f(\widetilde{\mathbf{y}}_n)\}_{n=1}^N$.

2.2 Sparse Precision Estimation in a Latent Gaussian Model

In this section, we extend the graphical lasso to learn a sparse precision of the latent Gaussian model (2.8). Our method is motivated by the following concerns.

- As explained in Section 2.1.3, learning a sparse structure for the joint model $p(\boldsymbol{u})$ may allow us to learn a parsimonious representation without sacrificing important structure in the marginal model $p(\boldsymbol{y})$. Therefore, when data are scarce, such a method might enable a more parsimonious model to be learned, and achieve better performance when predicting future data.

- Learning structure in the joint space may enable knowledge discovery: it may be possible to identify a latent factor by examination of the structure; or the residual structure in the observed space may uncover an important coupling that is not accounted for by the latent factors. For example, consider a group of financial assets from the same market sector. A latent variable may account for much of the intra-sector dependence – but a residual coupling may indicate that two assets are more tightly coupled than can be accounted for by market sector alone.

- Explicitly representing the latent variables means that the model is easily extensible. Indeed, we do extend it to a conditional mixture of Gaussians in Section 2.3.1, and further to a conditional mixture of Gaussian copulas in Section 2.4.2. In contrast, it is not immediately obvious how to extend a model such as SLR (see Section 2.1.3.1) in a similar manner.

- Handling missing data is easy when latent variables are explicit: the variables for which data is missing can simply be treated as latent. Again, compare with SLR for which this is not straightforward.

Note that sparsity in the joint space is a standard assumption of many commonly used latent variable models, including latent trees, hidden Markov models, restricted Boltzmann machines, and so on. All of these may potentially give rise to dense marginals. However, it is commonly assumed that a sparse latent structure is heuristically fixed a priori, whereas our method infers it from data.

2.2.1 A Sparse, Latent, Inverse Covariance Estimator

In (Agakov et al., 2012), we generalise the graphical lasso estimator (2.7) to the latent-variable situation. We continue to partition the precision and covariance as in (2.9), which we restate here for convenience:

$$\mathbf{\Lambda} = \begin{pmatrix} \mathbf{\Lambda}_{yy} & \mathbf{\Lambda}_{yz} \\ \mathbf{\Lambda}_{zy} & \mathbf{\Lambda}_{zz} \end{pmatrix}; \quad \mathbf{\Sigma} = \begin{pmatrix} \mathbf{\Sigma}_{yy} & \mathbf{\Sigma}_{yz} \\ \mathbf{\Sigma}_{zy} & \mathbf{\Sigma}_{zz} \end{pmatrix}.$$

Consider the following optimisation:

$$\hat{\mathbf{\Lambda}} = \arg\max_{\mathbf{\Lambda} \succeq 0} \left[\log\det \mathbf{\Sigma}_{yy}^{-1} - \mathrm{tr}\left(\boldsymbol{S}_{yy}\mathbf{\Sigma}_{yy}^{-1}\right) - \sum_{i,j} \Gamma_{ij}|\Lambda_{ij}| \right], \tag{2.18}$$

where $\mathbf{\Sigma} = \mathbf{\Lambda}^{-1}$. The first two terms are the same as in the graphical lasso, and represent the log likelihood of the observed data. But now the $L1$ penalties are applied to the elements of the full precision matrix $\mathbf{\Lambda}$. Unlike graphical lasso, this optimisation problem is not convex.

It is necessary to introduce constraints on $\mathbf{\Lambda}$ in (2.18). To see this, note that the log likelihood depends on $\mathbf{\Lambda}$ only through the marginal precision $\mathbf{\Sigma}_{yy}^{-1} = \mathbf{\Lambda}_{yy} - \mathbf{\Lambda}_{yz}\mathbf{\Lambda}_{zz}^{-1}\mathbf{\Lambda}_{zy}$. If there are H hidden variables, and $\boldsymbol{R} \in \mathbb{R}^{H\times H}$ is an invertible matrix,

$$\mathbf{\Lambda}_{yz}\mathbf{\Lambda}_{zz}^{-1}\mathbf{\Lambda}_{zy} = \mathbf{\Lambda}_{yz}\boldsymbol{R}\boldsymbol{R}^{-1}\mathbf{\Lambda}_{zz}^{-1}\left(\boldsymbol{R}^T\right)^{-1}\boldsymbol{R}^T\mathbf{\Lambda}_{yz}^T = \left(\mathbf{\Lambda}_{yz}\boldsymbol{R}\right)\left(\boldsymbol{R}^T\mathbf{\Lambda}_{zz}\boldsymbol{R}\right)^{-1}\left(\mathbf{\Lambda}_{yz}\boldsymbol{R}\right)^T. \tag{2.19}$$

If we set

$$\mathbf{\Lambda}'_{yy} = \mathbf{\Lambda}_{yy}, \tag{2.20}$$

$$\mathbf{\Lambda}'_{yz} = \mathbf{\Lambda}_{yz}\boldsymbol{R}, \tag{2.21}$$

$$\mathbf{\Lambda}'_{zz} = \boldsymbol{R}^T\mathbf{\Lambda}_{zz}\boldsymbol{R}, \tag{2.22}$$

then $\mathbf{\Lambda}'$ remains symmetric positive definite, and has the same marginal precision as $\mathbf{\Lambda}$. But $\boldsymbol{R}$ may be chosen such that $\mathbf{\Lambda}'$ incurs a smaller penalty than $\mathbf{\Lambda}$; for example, let $\boldsymbol{R}$ be diagonal with $0 < R_{hh} < 1,\ h = 1\ldots H$. The $L1$ penalties prefer a solution with arbitrarily small, but non-zero, values for the elements of $\mathbf{\Lambda}_{yz}$ and $\mathbf{\Lambda}_{zz}$.

To avoid this, we require constraints analogous to fixing the variances of the latent factors in factor analysis or latent factor models. A natural choice would be to impose unit variance on the latent variables: $\mathrm{diag}\left(\mathbf{\Sigma}_{zz}\right) = \mathbf{1}$. However, this makes the optimisation (2.18) difficult. Instead, we typically impose unit *partial* variance on the latent variables: $\mathrm{diag}\left(\mathbf{\Lambda}_{zz}\right) = \mathbf{1}$. Other constraints are possible, of course; we favour

this one because it is straightforwardly interpretable, and does not overly complicate the optimisation problem. With these constraints, the estimator becomes

$$\hat{\boldsymbol{\Lambda}} = \underset{\boldsymbol{\Lambda}\succeq 0\,:\,\mathrm{diag}(\boldsymbol{\Lambda}_{zz})=\mathbf{1}}{\arg\max} \left[\log\det \boldsymbol{\Sigma}_{yy}^{-1} - \mathrm{tr}\left(\boldsymbol{S}_{yy}\boldsymbol{\Sigma}_{yy}^{-1}\right) - \sum_{i,j}\Gamma_{ij}|\Lambda_{ij}| \right]. \tag{2.23}$$

We refer to this sparse, latent, inverse covariance estimator as SLICE. We also use SLICE to refer to the estimator with different constraints, and to the EM algorithm for computing the estimator, which we introduce in Section 2.2.2.

2.2.1.1 Hierarchical Models

If we let some penalty $\Gamma_{ij} \to \infty$, then Λ_{ij} will be forced to zero and edge (i, j) will be missing from the learned model. This allows us to use prior knowledge to limit the range of structures that might be learned. For example, we might enforce a multi-layered structure by grouping the nodes into layers, and putting large penalties on edges between nodes that are not in the same or adjacent layers.

2.2.1.2 Relationship between SLICE and Factor Analysis

The factor analysis (FA) model (Everitt, 1984) consists of observed variables $\boldsymbol{y}$ and latent variables $\boldsymbol{z}$ generated from the following distributions:

$$\boldsymbol{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}) \tag{2.24}$$

$$\boldsymbol{y}|\boldsymbol{z} \sim \mathcal{N}\left(\boldsymbol{W}\boldsymbol{z} + \boldsymbol{\mu}_y, \boldsymbol{\Psi}\right), \tag{2.25}$$

where $\boldsymbol{\Psi}$ is a diagonal matrix. The joint distribution is

$$p(\boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z}) \tag{2.26}$$

$$\propto \exp\left[-\frac{1}{2}\left(\boldsymbol{y} - \boldsymbol{W}\boldsymbol{z} - \boldsymbol{\mu}_y\right)^T \boldsymbol{\Psi}^{-1}\left(\boldsymbol{y} - \boldsymbol{W}\boldsymbol{z} - \boldsymbol{\mu}_y\right) \right] \exp\left[-\frac{1}{2}\boldsymbol{z}^T\boldsymbol{z} \right] \tag{2.27}$$

$$\propto \exp\left[-\frac{1}{2}\left\{ \widetilde{\boldsymbol{y}}^T\boldsymbol{\Psi}^{-1}\widetilde{\boldsymbol{y}} - 2\widetilde{\boldsymbol{y}}^T\boldsymbol{\Psi}^{-1}\boldsymbol{W}\boldsymbol{z} + \boldsymbol{z}^T\left(\boldsymbol{W}^T\boldsymbol{\Psi}^{-1}\boldsymbol{W} + \boldsymbol{I}\right)\boldsymbol{z} \right\} \right], \tag{2.28}$$

where $\widetilde{\boldsymbol{y}} = \boldsymbol{y} - \boldsymbol{\mu}_y$. Let $\boldsymbol{u} = \left(\boldsymbol{y}^T, \boldsymbol{z}^T\right)^T$ and $\boldsymbol{\mu} = \left(\boldsymbol{\mu}_y^T, \boldsymbol{\mu}_z^T\right)^T$, where $\boldsymbol{\mu}_z = \mathbf{0}$. The joint distribution is Gaussian: $p(\boldsymbol{u}) = \mathcal{N}\left(\boldsymbol{u}\middle|\boldsymbol{\mu}, \boldsymbol{\Lambda}^{-1}\right)$, where

$$\boldsymbol{\Lambda} = \begin{pmatrix} \boldsymbol{\Psi}^{-1} & -\boldsymbol{\Psi}^{-1}\boldsymbol{W} \\ -\boldsymbol{W}^T\boldsymbol{\Psi}^{-1} & \boldsymbol{W}^T\boldsymbol{\Psi}^{-1}\boldsymbol{W} + \boldsymbol{I} \end{pmatrix}. \tag{2.29}$$

We can now see how to set the SLICE penalties and constraints to recover the factor analysis solution:

1. Set $(\Gamma_{yy})_{ij} \to \infty, i \neq j$ (to force $\boldsymbol{\Lambda}_{yy}$ to be diagonal) and all other elements of $\boldsymbol{\Gamma}$ to zero;

2. Impose the constraint $\boldsymbol{\Lambda}_{zz} = \boldsymbol{W}^T \boldsymbol{\Psi}^{-1} \boldsymbol{W} + \boldsymbol{I}$, or equivalently $\boldsymbol{\Sigma}_{zz} = \boldsymbol{I}$.

Note that, while useful for theoretical insight, this relationship between SLICE and FA does not appear to be useful for solving factor analysis efficiently.

For SLICE in general, and in contrast to factor analysis, $\boldsymbol{\Lambda}_{yy}$ is not necessarily diagonal: the conditional $p(\boldsymbol{y}|\boldsymbol{z})$ does not generally factorise as $\prod_{v=1}^{V} p(y_v|\boldsymbol{z})$. Furthermore, SLICE generally has no rotation invariance in the latent space because the $L1$ penalties break this symmetry.

2.2.1.3 Relationship between SLICE and SLR

The SLICE model is a Gaussian on the joint distribution of the observed and latent variables. So the marginal distribution of the observed variables is also a Gaussian, and its precision is given by Equation (2.10), which we restate here:

$$\boldsymbol{\Sigma}_{yy}^{-1} = \boldsymbol{\Lambda}_{yy} - \boldsymbol{\Lambda}_{yz} \boldsymbol{\Lambda}_{zz}^{-1} \boldsymbol{\Lambda}_{zy}.$$

SLR learns the two terms in this equation, but does not decompose the second term. Thus, SLR incorporates latent variables implicitly. This results in a convex optimisation problem for SLR, with the advantages that there are no local minima, and the optimisation is faster than SLICE's EM algorithm.

However, there are advantages to retaining the latent variables explicitly. Having learned the full precision matrix, the latent variables may be interpretable, aiding knowledge discovery. This may also be possible in SLR, but it requires an additional decomposition of the low-rank matrix. SLICE could handle missing data by treating the missing values as latent variables, and incorporating them into the EM algorithm. Also note that the SLICE model is a probabilistic graphical model. First, this makes it extensible – and we do extend it to a mixture of sparse, latent, non-Gaussian experts in Sections 2.3.1 and 2.4.2. Second, it means that it is more flexible for encoding prior knowledge than SLR. For example, in our experiments in Section 2.5, we have a financial data set with known market sectors. One could make a soft assignment of a latent variable to a market sector by choosing different penalties on connections from the latent variable to observed variables within the sector than without.

2.2.1.4 Relationships between SLICE and Other Models

The probabilistic PCA (PPCA) model (Tipping and Bishop, 1999) is similar to factor analysis (2.24–2.25), but with a more restrictive noise model: in FA, $\boldsymbol{\Psi}$ is diagonal, while in PPCA, $\boldsymbol{\Psi} = \sigma^2 \boldsymbol{I}$. SLICE is therefore related to PPCA in a similar manner as to FA; see Section 2.2.1.2.

In the classical formulations of FA and PPCA, the rows of the data matrix $\boldsymbol{Y}$ are independently generated data points, while the variables associated with the columns of $\boldsymbol{Y}$ are coupled. We introduced SLICE in this way. Of course, if we treat $\boldsymbol{Y}^T$ as the data matrix, then FA, PPCA, and SLICE become models in which the variables are independent but the data points are correlated. In the case of PPCA, Lawrence (2005) showed that maximum likelihood estimation for each variant requires the solution of an equivalent eigenvalue problem.

Engelhardt and Stephens (2010) introduce sparse factor analysis to model correlations between the data points (but it could, of course, be used to capture correlations between the observed variables, as above). Their model extends standard factor analysis – see Equations (2.24 – 2.25) – with an automatic relevance determination (ARD) prior to induce sparsity in the factor loadings. The model is:

$$W_{nh} \sim \mathcal{N}\left(0, \Omega_{nh}^2\right), \tag{2.30}$$

$$\Psi_{nn} \sim \text{Inv–Gamma}(\alpha, \beta), \tag{2.31}$$

$$Y_{nv} | \boldsymbol{W} \sim \mathcal{N}\left(\mu_v + (\boldsymbol{W}\boldsymbol{Z})_{nv}, \Psi_{nn}\right). \tag{2.32}$$

They set $\{\alpha, \beta\}$ by hand, marginalise the factor loadings $\boldsymbol{W}$, and learn $\{\boldsymbol{\mu}, \boldsymbol{Z}, \boldsymbol{\Omega}, \boldsymbol{\Psi}\}$ by maximum likelihood using an ECME (Liu and Rubin, 1994) algorithm. Notice that SLICE and sparse FA differ in what is sparsified. In SLICE, we marginalise the latent variables and learn a sparse precision that maximises the ($L1$-penalised) likelihood. In sparse FA, the learned parameters may be dense, while sparsity is imposed on the variables that are marginalised during optimisation.

Kalaitzis and Lawrence (2012) study the following model:

$$\boldsymbol{z}_1 \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \tag{2.33}$$

$$\boldsymbol{z}_2 \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}_2^{-1}\right), \tag{2.34}$$

$$\boldsymbol{y} | \boldsymbol{z}_1, \boldsymbol{z}_2 \sim \mathcal{N}\left(\boldsymbol{W}\boldsymbol{z}_1 + \boldsymbol{z}_2, \sigma^2 \boldsymbol{I}\right). \tag{2.35}$$

Without $\boldsymbol{z}_2$, this reduces to PPCA. Kalaitzis and Lawrence (2012) estimate the parameters of this model by integrating over $\boldsymbol{z}_1$ and maximising the likelihood with respect

to $\mathbf{\Lambda}_2$ and $\boldsymbol{W}$, with $L1$ penalties on the elements of $\mathbf{\Lambda}_2$. They do this by alternating between an EM step and a step of their residual component analysis (RCA) algorithm; the resulting algorithm is named EM/RCA. During the E step of EM, expectations of $\boldsymbol{z}_2$ are computed; during the M step, graphical lasso is run to update $\mathbf{\Lambda}_2$. The RCA step solves a generalised eigenvalue problem to update $\boldsymbol{W}$ given $\mathbf{\Lambda}_2$. The marginal covariance of the trained model is the sum of a low-rank matrix and the inverse of a sparse precision: $\mathbf{\Sigma}_{yy} = \boldsymbol{W}\boldsymbol{W}^T + \mathbf{\Lambda}_2^{-1}$. To see the relationship to SLICE, notice that the visible variables $\boldsymbol{y}$ and the latent variables $\boldsymbol{z} \equiv \left(\boldsymbol{z}_1^T, \boldsymbol{z}_2^T\right)^T$ are jointly distributed as a zero-mean Gaussian. The joint precision is a 3×3 block matrix (with blocks corresponding to $\boldsymbol{y}$, $\boldsymbol{z}_1$, and $\boldsymbol{z}_2$), with different constraints on each block. SLICE has none of the same constraints as EM/RCA, but has the additional constraint that $\operatorname{diag}(\mathbf{\Lambda}_{zz}) = \mathbf{1}$.

Städler and Bühlmann (2012) use an EM method to train the graphical lasso model when there are missing observations in the data – in contrast to the latent variables in the SLICE model. Städler and Bühlmann (2012) extend their method to a sparse regression, whereas we extend SLICE to a discriminative mixture of sparse Gaussian copulas; see Sections 2.3.1 and 2.4.2.

2.2.2 An EM Algorithm for SLICE

To compute the optimisation (2.23), we use a structural EM approach (Friedman, 1997). We assume that the data has been centred, and we fix the mean of each latent variable to zero. So the mean of $\boldsymbol{u} \equiv \left(\boldsymbol{y}^T, \boldsymbol{z}^T\right)^T$ is $\boldsymbol{\mu} = \mathbf{0}$. Let $\boldsymbol{Y} = (\boldsymbol{y}_1, ..., \boldsymbol{y}_N)^T$ be the matrix of observed data. The objective function is the expected full-data log likelihood, which can be written

$$Q\left(\mathbf{\Lambda}; \mathbf{\Lambda}^{(t-1)}\right) \equiv \log\det\mathbf{\Lambda} - \operatorname{tr}(\boldsymbol{S}\mathbf{\Lambda}) - \sum_{i,j}\Gamma_{ij}|\Lambda_{ij}|, \tag{2.36}$$

where $\boldsymbol{S}$ is an estimate of the full covariance conditioned on the previous precision:

$$\boldsymbol{S}_{yy} = \frac{1}{N}\boldsymbol{Y}^T\boldsymbol{Y}, \tag{2.37}$$

$$\boldsymbol{S}_{zy}^T = \boldsymbol{S}_{yz} = \frac{1}{N}\boldsymbol{Y}^T\overline{\boldsymbol{Z}}, \tag{2.38}$$

$$\boldsymbol{S}_{zz} = \left(\mathbf{\Lambda}_{zz}^{(t-1)}\right)^{-1} + \frac{1}{N}\overline{\boldsymbol{Z}}^T\overline{\boldsymbol{Z}}. \tag{2.39}$$

$\overline{\boldsymbol{Z}} = (\overline{\boldsymbol{z}}_1, ..., \overline{\boldsymbol{z}}_N)^T$ contains the expectations of the latent variables, where each

$$\overline{\boldsymbol{z}}_n \equiv \mathbb{E}\left[\boldsymbol{z}_n | \boldsymbol{y}_n; \mathbf{\Lambda}^{(t-1)}\right] = -\left(\mathbf{\Lambda}_{zz}^{(t-1)}\right)^{-1}\mathbf{\Lambda}_{zy}^{(t-1)}\boldsymbol{y}_n \tag{2.40}$$

is the mean of a conditional Gaussian. See Appendix A for a derivation of these equations[1].

The expectations $\overline{\boldsymbol{Z}}$ and $\boldsymbol{S}$ are computed in the E step. Essentially, this is an estimation of the first and second moments of the joint Gaussian distribution using the parameters from the previous iteration. In the M step, $Q\left(\boldsymbol{\Lambda};\boldsymbol{\Lambda}^{(t-1)}\right)$ is maximised with respect to $\boldsymbol{\Lambda}$, subject to the constraints that $\boldsymbol{\Lambda}$ must be positive semi-definite, and that $\operatorname{diag}(\boldsymbol{\Lambda}_{zz}) = \mathbf{1}$. Note that this is the same optimisation problem as the graphical lasso (2.7), but with an additional constraint. A summary of the procedure is shown in Algorithm 1.

Algorithm 1 EM for Sparse Latent Inverse Covariance Estimation (SLICE)

Initialise $\boldsymbol{\Lambda}^{(0)}$ such that $\boldsymbol{\Lambda}^{(0)} \succ 0$

for $t \leftarrow 1 : T$ **do**

E Step

$$\overline{\boldsymbol{Z}} \leftarrow -\boldsymbol{Y}\boldsymbol{\Lambda}_{yz}^{(t-1)}\left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1}$$

end E Step

M Step

$$\boldsymbol{S} \leftarrow \frac{1}{N}\begin{pmatrix} \boldsymbol{Y}^T\boldsymbol{Y} & \boldsymbol{Y}^T\overline{\boldsymbol{Z}} \\ \overline{\boldsymbol{Z}}^T\boldsymbol{Y} & N\left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1} + \overline{\boldsymbol{Z}}^T\overline{\boldsymbol{Z}} \end{pmatrix}$$

$$\boldsymbol{\Lambda}^{(t)} \leftarrow \underset{\boldsymbol{\Lambda}\,:\,\boldsymbol{\Lambda}\succeq 0,\ \operatorname{diag}(\boldsymbol{\Lambda}_{zz})=\mathbf{1}}{\arg\max} \left[\log\det\boldsymbol{\Lambda} - \operatorname{tr}(\boldsymbol{S}\boldsymbol{\Lambda}) - \textstyle\sum_{i,j}\Gamma_{ij}|\Lambda_{ij}|\right]$$

end M Step

end for

The loop may be run until some maximum number of iterations, as shown in Algorithm 1, but other stopping criteria may also be appropriate. We often computed the increase in the objective function at each iteration, and terminated the algorithm when this value fell below a threshold.

2.2.2.1 Solving the M-Step Optimisation Problem

Various software packages are capable of solving the M-step optimisation problem. We made use of the following.

1. SDPT3 (Toh et al., 1999) solves a large class of linear, quadratic, and semidefinite programming problems.

[1] Appendix A actually contains a derivation for the more general CopMSLICE model, which we introduce in Section 2.4.2.

2. LogdetPPA (Wang et al., 2010) solves log-determinant problems of the following kind:

$$\min_{\boldsymbol{\Lambda}} \left[\operatorname{tr}(\boldsymbol{C}\boldsymbol{\Lambda}) - \mu \log\det \boldsymbol{\Lambda} \;:\; A(\boldsymbol{\Lambda}) = \boldsymbol{b},\ \boldsymbol{\Lambda} \succ 0 \right] , \tag{2.41}$$

where A is a linear map, and μ, $\boldsymbol{b}$, and $\boldsymbol{C}$ are constant.

3. L1General (Schmidt et al., 2007) is a collection of algorithms for solving $L1$-penalised optimisation problems. We found the L1General2_PSSgb algorithm to perform well.

Chandrasekaran et al. (2010) used SDPT3 for their SLR method when dimensionality of the observed variables was low, and LogdetPPA in higher dimensions. For SLICE, we also found SDPT3 to be faster than LogdetPPA for low-dimensional problems, by which we mean up to around 40 dimensions or so. But we found L1General to be much faster than SDPT3 in this range. LogdetPPA is faster than both SDPT3 and L1General for higher dimensional problems. But none of the methods is completely reliable: each one was observed to fail to correctly optimise the objective on rare occasions. This was seen either by inspection, or by comparing the output of each package. So none of the methods is redundant. Finding more reliable optimisers, or characterising the problems on which they fail, is left for future work.

2.2.2.2 Initialisation

The SLICE optimisation (2.23) is non-convex, so local optima may be present. If so, the choice of initial precision will determine the local optimum found (unlike for the graphical lasso). The best optimum depends on what one desires from the solution. Typically, it is a good idea to use an initialisation that is close to a desired solution, roughly speaking. Some initialisations for $\boldsymbol{\Lambda}$ that we have utilised are as follows.

- Set $\boldsymbol{\Lambda}_{yy} = \boldsymbol{S}_{yy}^{-1}$, $\boldsymbol{\Lambda}_{yz} = \boldsymbol{0}$, and $\boldsymbol{\Lambda}_{zz} = \boldsymbol{I}$. That is, the latent variables have unit variance and are initially unconnected to the visibles or to each other, while the covariance of the visibles is set to the empirical covariance $\boldsymbol{S}_{yy}$.

- Set $\boldsymbol{\Lambda}$ to the factor analysis or PPCA solution. SLICE is constrained such that $\operatorname{diag}(\boldsymbol{\Lambda}_{zz}) = \boldsymbol{1}$, so rescale the rows and columns of the FA/PPCA solution to satisfy this constraint before running the SLICE algorithm.

- Use the rescaled FA/PPCA solution, but additionally modify the residuals $\boldsymbol{\Lambda}_{yy}$ such that $\boldsymbol{\Sigma}_{yy} = \boldsymbol{S}_{yy}$.

In practice, we found that modifying the residuals had little effect, and we most often used the rescaled factor analysis initialisation.

2.3 A Conditional Mixture of Sparse Latent Gaussians

The SLICE model, and the graphical lasso that SLICE generalises, assumes that there is a single Gaussian model – with a single structure – that describes the entire data set. In a real-life application, this may not be the case. Instead, it may be that the data points lie in groups, where each group has a different structure. For example, the structure of an individual's metabolic pathway network is thought to vary between males and females. Another example from biology is the hypothesis that cancer patients may be grouped according to their response to chemotherapy: some patients may respond better than others, the underlying cause for which is a difference in the structure of their proteomic networks. In the financial domain, which provides much of the motivation for our work, there is evidence to show that assets tend to become more strongly correlated under extreme market conditions; see, for example, (Preis et al., 2012). In such circumstances, a single Gaussian model may not be appropriate. We therefore extend SLICE to a mixture in which each mixture component is a sparse latent Gaussian model, and each component may have a different structure.

We also observe that in a real-world application, we may have access to additional covariates whose distribution and structure are of no interest, but that help to infer the mixture component to which a data point belongs. We often refer to these covariates as side information. In finance, for example, macroeconomic indicators (such as interest rates), market indices (such as the FTSE 100 index), technical indicators[2], and news stories may inform on the current market structure. More generally, a domain expert is often able to construct a high-dimensional feature vector that is predictive of structural changes. We condition our mixture on the side information: since we do not need to model the covariates, training a conditional model is more efficient. The mixture model therefore becomes a conditional mixture, otherwise known as a mixture of experts (Jacobs et al., 1991).

It is important for knowledge discovery and decision support applications that we can identify which features of the side information are most strongly predictive of structural changes. For example, in financial risk management, it is important to iden-

[2] A technical indicator is a feature, typically computed from recent market data, that is used by a technical analyst to forecast future price changes.

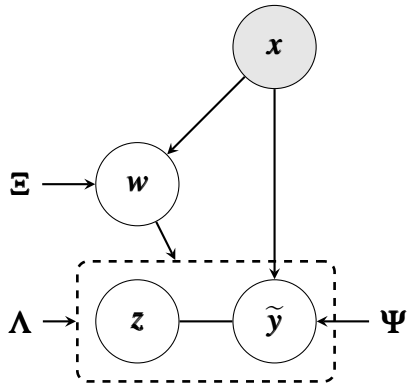


Figure 2.4: MSLICE, a conditional mixture of sparse latent Gaussians. The side information $\boldsymbol{x}$ affects the choice of mixture component $\boldsymbol{w}$ and the mean of the visible variables $\widetilde{\boldsymbol{y}}$. The latent variables $\boldsymbol{z}$ and the visibles are jointly Gaussian; the structure of this Gaussian may be different for each mixture component.

tify how particular changes in market conditions could affect portfolio returns. In a similar manner to lasso regression, we impose $L1$ penalties to sparsify the weights on the covariates and select the most important features. Thus, even though the side information may be high-dimensional, we can potentially discover a small, interpretable set of the most predictive features.

In the next section, the model is presented in more detail.

2.3.1 A Conditional Mixture of SLICE Models

We typically use M to refer to the number of mixture components (experts), and so we name this model MSLICE. It is illustrated in Figure 2.4. Let $\widetilde{\boldsymbol{y}} \in \mathbb{R}^V$ represent the visible variables. (We reserve $\boldsymbol{y}$ for the visibles after subtracting the mean, as explained shortly). Let $\boldsymbol{x} \in \mathbb{R}^C$ denote a vector of covariates, and $\boldsymbol{z} \in \mathbb{R}^H$ denote the hidden variables. Let $\boldsymbol{w} \in \{0,1\}^M$ indicate which of the experts is responsible for generating $\widetilde{\boldsymbol{y}}$; that is, $w_m = 1$ for some m, and $w_i = 0$ for all $i \neq m$. The distribution of $\boldsymbol{w}$ is parameterised by $\boldsymbol{\Xi}$, while the expert distributions are parameterised by $\boldsymbol{\Psi} \equiv \{\boldsymbol{\Psi}_m\}_{m=1}^M$ and $\boldsymbol{\Lambda} \equiv \{\boldsymbol{\Lambda}_m\}_{m=1}^M$.

In the single-component SLICE model, the data can be centred in a preprocessing step, and so it is sufficient for the Gaussian in the SLICE model to have zero mean. But with multiple components, it is not known *a priori* which experts generated which data points. We must therefore parameterise the mean of each component, and learn it. (We still set the means of the hidden variables to zero). The side information may

also be predictive of the means. In finance, for example, one expert may correspond to a falling market (with some learned structure), and the side may indicate the degree to which the market is falling. So we allow the expert means to depend on the side information. Many parameterisations are possible. In this work, we choose a linear relationship: we define the mean of the visible variables for expert m to be $\widetilde{\boldsymbol{\mu}}_m \equiv \boldsymbol{\Psi}_m^T \widetilde{\boldsymbol{x}}$, where $\widetilde{\boldsymbol{x}} = \left(1, \boldsymbol{x}^T\right)^T$ is the side information augmented with a unit element, and $\boldsymbol{\Psi}_m \in \mathbb{R}^{(C+1)\times V}$. We then define $\boldsymbol{y}_m = \widetilde{\boldsymbol{y}} - \widetilde{\boldsymbol{\mu}}_m$ to be the offset of $\widetilde{\boldsymbol{y}}$ from the mean of expert m.

We also impose $L1$ penalties on the elements of the matrices $\boldsymbol{\Psi}_m$ so that we learn which features of the side information are most predictive of the expert means. The first row of $\boldsymbol{\Psi}_m$ multiplies the unit element of $\widetilde{\boldsymbol{x}}$, and so represents an intercept: it is the mean of expert m when $\boldsymbol{x} = \boldsymbol{0}$. There is often no reason to expect an expert to have a zero intercept, so the first row of $\boldsymbol{\Psi}_m$ is usually unpenalised.

Note that with the above choices, each expert is now a sparse linear regression with SLICE as the noise model.

For the distribution $p(\boldsymbol{w}|\boldsymbol{x};\boldsymbol{\Xi})$, there are various possible options. We worked with a multinomial logit (softmax) model, setting

$$p\left(w_m = 1|\boldsymbol{x};\boldsymbol{\Xi}\right) = \frac{\exp\left(\boldsymbol{\xi}_m^T \widetilde{\boldsymbol{x}}\right)}{\sum_{i=1}^{M} \exp\left(\boldsymbol{\xi}_i^T \widetilde{\boldsymbol{x}}\right)}, \tag{2.42}$$

where $\boldsymbol{\xi}_m \equiv \boldsymbol{\Xi}_{:m}$ is the m^{th} column of matrix $\boldsymbol{\Xi} \in \mathbb{R}^{(C+1)\times M}$. As with $\boldsymbol{\Psi}_m$, we also impose $L1$ penalties on the elements of $\boldsymbol{\xi}_m$ – except, typically, the first element (which corresponds to an intercept) – so that we can select which features of the side are most predictive of the mixture component. These penalties also serve to remove a multiplicity of equivalent maximum likelihood solutions. Notice that adding a constant to some row $\boldsymbol{\Xi}_{c:}$ does not change the probabilities (2.42); so if $\boldsymbol{\Xi}_{c:}$ is unpenalised, these parameters could grow without bound during optimisation. Provided at least one of them has non-zero penalty, this situation is prevented. If an entire row is unpenalised, we impose the additional constraint that the row have zero mean.

The generative model is as follows.

$$\boldsymbol{w}|\boldsymbol{x} \sim \text{softmax}\left(\boldsymbol{\xi}_1^T \widetilde{\boldsymbol{x}}, \ldots, \boldsymbol{\xi}_M^T \widetilde{\boldsymbol{x}}\right), \tag{2.43}$$

$$\boldsymbol{u}|w_m = 1, \boldsymbol{x} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}_m\right) \tag{2.44}$$

where the softmax function is defined such that $w_m = 1$ with the probability stated in Equation (2.42); $\boldsymbol{u} \equiv \left(\boldsymbol{y}^T, \boldsymbol{z}^T\right)^T$; and the observed variables are translated according to

the generating expert: $\widetilde{\boldsymbol{y}} = \boldsymbol{y} + \widetilde{\boldsymbol{\mu}}_m$. When learning the model, the precision of each expert must still be constrained in the same way as the single-component SLICE model. Let $\boldsymbol{\theta} = (\boldsymbol{\Xi}, \boldsymbol{\Psi}, \boldsymbol{\Lambda})$ contain all the parameters to be learned. The optimisation problem for MSLICE is

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}\,:\,\boldsymbol{\Lambda}_m \succeq 0,\ \mathrm{diag}\left((\boldsymbol{\Lambda}_m)_{zz}\right)=\mathbf{1}}{\arg\max} \left[\sum_n \log \left\{ \sum_{\boldsymbol{w}_n} \int p\left(\boldsymbol{u}_n, \boldsymbol{w}_n | \boldsymbol{x}_n; \boldsymbol{\theta}\right) d\boldsymbol{z}_n \right\} - \gamma(\boldsymbol{\theta}) \right], \tag{2.45}$$

where $\gamma(\boldsymbol{\theta})$ contains all the $L1$ penalties:

$$\gamma(\boldsymbol{\theta}) \equiv \gamma_\Xi(\boldsymbol{\Xi}) + \gamma_\Psi(\boldsymbol{\Psi}) + \gamma_\Lambda(\boldsymbol{\Lambda}) \tag{2.46}$$

$$\gamma_\Xi(\boldsymbol{\Xi}) \equiv \sum_{m=1}^{M} \sum_{c=1}^{C} (\Gamma_\Xi)_{cm} \left| \Xi_{(c+1)m} \right| \tag{2.47}$$

$$\gamma_\Psi(\boldsymbol{\Psi}) \equiv \sum_{m=1}^{M} \sum_{c=1}^{C} \sum_{v=1}^{V} (\Gamma_{\Psi_m})_{cv} \left| (\Psi_m)_{(c+1)v} \right| \tag{2.48}$$

$$\gamma_\Lambda(\boldsymbol{\Lambda}) \equiv \sum_{m=1}^{M} \sum_{i=1}^{V+H} \sum_{j=1}^{V+H} (\Gamma_{\Lambda_m})_{ij} \left| (\Lambda_m)_{ij} \right|. \tag{2.49}$$

Elements of the matrices Γ_Ξ, Γ_{Ψ_m}, and Γ_{Λ_m} are all non-negative. As for SLICE, we use an EM algorithm to solve the optimisation problem.

2.3.2 An EM Algorithm for MSLICE

The EM objective function is the full data expected log likelihood, conditioned on the data and the previous values of the parameters:

$$Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(t-1)}\right) = \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}} \left[\mathcal{L}\left(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}\right) \right] - \gamma(\boldsymbol{\theta}), \tag{2.50}$$

where $\mathcal{L}\left(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}\right)$ is the log likelihood,

$$\mathcal{L}\left(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}\right) = \sum_{n=1}^{N} \sum_{m=1}^{M} w_{nm} \left[\log p\left(w_{nm} = 1 | \boldsymbol{x}_n; \boldsymbol{\Xi}\right) + \log p\left(\boldsymbol{u}_n | w_{nm} = 1; \boldsymbol{\Psi}_m, \boldsymbol{\Lambda}_m\right) \right]. \tag{2.51}$$

See Appendix A for a full derivation. We now describe the optimisation of each of the parameters $\boldsymbol{\Xi}$, $\boldsymbol{\Psi}$ and $\boldsymbol{\Lambda}$ separately.

2.3.2.1 The Mixing Model

$\boldsymbol{\Xi}$ appears in a different term to $\boldsymbol{\Psi}$ and $\boldsymbol{\Lambda}$, and so can be optimised separately. The objective for $\boldsymbol{\Xi}$ can be written

$$Q_{\Xi}\left(\boldsymbol{\Xi};\boldsymbol{\theta}^{(t-1)}\right) \equiv \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}\left[\sum_{n,m} w_{nm}\log p\left(w_{nm}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}\right)\right] - \gamma_{\Xi}\left(\boldsymbol{\Xi}\right) \tag{2.52}$$

$$= \sum_{n,m}\overline{w}_{nm}\left\{\boldsymbol{\xi}_m^T\widetilde{\boldsymbol{x}}_n - \log\sum_{i=1}^{M}\exp\left(\boldsymbol{\xi}_i^T\widetilde{\boldsymbol{x}}_n\right)\right\} - \gamma_{\Xi}\left(\boldsymbol{\Xi}\right). \tag{2.53}$$

The expectation $\overline{w}_{nm}$ of each component indicator is known as the responsibility of component m for data point n. These expectations are evaluated during the E step according to the following equation:

$$\overline{w}_{nm} = \frac{p\left(\widetilde{\boldsymbol{y}}_n|w_{nm}=1,\boldsymbol{x}_n;\boldsymbol{\Psi}_m^{(t-1)},\boldsymbol{\Lambda}_m^{(t-1)}\right)p\left(w_{nm}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}^{(t-1)}\right)}{\sum_{i=1}^{M}p\left(\widetilde{\boldsymbol{y}}_n|w_{ni}=1,\boldsymbol{x}_n;\boldsymbol{\Psi}_i^{(t-1)},\boldsymbol{\Lambda}_i^{(t-1)}\right)p\left(w_{ni}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}^{(t-1)}\right)}. \tag{2.54}$$

The first probability in the numerator is a Gaussian, the second a multinomial logit, so these are easily computed.

Maximising (2.53) with respect to $\boldsymbol{\Xi}$ is a sparse multinomial logit problem, and is solved by existing methods, such as the glmnet software (Friedman et al., 2010b). As for SLICE, we also use L1General (Schmidt et al., 2007) to solve this problem.

2.3.2.2 The Component Means

The parameters of each expert, $\boldsymbol{\Psi}_m$ and $\boldsymbol{\Lambda}_m$, are coupled in the objective (2.50). In standard EM fashion, we therefore condition on each while we optimise with respect to the other. We note that our M-step update of the pair $(\boldsymbol{\Psi}_m,\boldsymbol{\Lambda}_m)$ is similar to running a single iteration of the MRCE sparse regression algorithm of Rothman et al. (2010).

The objective function for $\boldsymbol{\Psi}_m$ is

$$\begin{aligned} &Q_{\Psi_m}\left(\boldsymbol{\Psi}_m|\boldsymbol{\Lambda}_m;\boldsymbol{\theta}^{(t-1)}\right) \equiv \\ &\operatorname{tr}\left[\boldsymbol{\Psi}_m^T\widetilde{\boldsymbol{X}}^T\boldsymbol{W}_m\left\{\overline{\boldsymbol{Z}}_m\left(\boldsymbol{\Lambda}_m\right)_{zy} + \left(\widetilde{\boldsymbol{Y}} - \frac{1}{2}\widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)\left(\boldsymbol{\Lambda}_m\right)_{yy}\right\}\right] - \gamma_{\Psi_m}(\boldsymbol{\Psi}_m), \end{aligned} \tag{2.55}$$

where

$$\overline{\boldsymbol{z}}_{nm} \equiv \mathbb{E}\left[\boldsymbol{z}_n|\widetilde{\boldsymbol{y}}_n;\boldsymbol{\Psi}_m^{(t-1)},\boldsymbol{\Lambda}_m^{(t-1)}\right] \tag{2.56}$$

$$= -\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1}\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zy}\left(\widetilde{\boldsymbol{y}}_n - \left(\boldsymbol{\Psi}_m^{(t-1)}\right)^T\widetilde{\boldsymbol{x}}_n\right) \tag{2.57}$$

is computed during the E step, $\overline{\boldsymbol{Z}}_m \equiv (\bar{\boldsymbol{z}}_{1m}, \ldots, \bar{\boldsymbol{z}}_{Nm})^T$, and $\boldsymbol{W}_m \equiv \operatorname{diag}(\overline{\boldsymbol{w}}_{:m})$. If $\boldsymbol{\Psi}_m$ is unpenalised – that is, if $\gamma_{\Psi_m}(\boldsymbol{\Psi}_m) = 0$ – we can maximise Q_{Ψ_m} in closed form, because (2.55) is then quadratic in $\boldsymbol{\Psi}_m$. In the general case, the maximum must be found using an appropriate optimiser. We again chose to use the L1General package (Schmidt et al., 2007).

2.3.2.3 The Component Precisions

The objective function for $\boldsymbol{\Lambda}_m$ is

$$Q_{\Lambda_m}\left(\boldsymbol{\Lambda}_m | \boldsymbol{\Psi}_m; \boldsymbol{\theta}^{(t-1)}\right) \equiv \frac{N_m}{2}\left[\log\det\boldsymbol{\Lambda}_m - \operatorname{tr}(\boldsymbol{S}_m\boldsymbol{\Lambda}_m)\right] - \gamma_{\Lambda_m}(\boldsymbol{\Lambda}_m), \tag{2.58}$$

where

$$(\boldsymbol{S}_m)_{yy} = \frac{1}{N_m}\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)^T \boldsymbol{W}_m \left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right), \tag{2.59}$$

$$(\boldsymbol{S}_m)_{yz} = \frac{1}{N_m}\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m, \tag{2.60}$$

$$(\boldsymbol{S}_m)_{zz} = (\boldsymbol{\Lambda}_m)_{zz}^{-1} + \frac{1}{N_m}\overline{\boldsymbol{Z}}_m^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m, \tag{2.61}$$

and $N_m = \sum_n \overline{w}_{nm}$ is the expected number of data points for which component m is responsible. The constraints on $\boldsymbol{\Lambda}_m$ are the same as those for the single-component SLICE, namely that $\boldsymbol{\Lambda}_m$ must be positive definite and that $(\boldsymbol{\Lambda}_m)_{zz}$ must have a unit diagonal. So the optimisation problem is the same constrained graphical lasso as in the M step of SLICE; see Section 2.2.2.

The EM algorithm for MSLICE is summarised in Algorithm 2.

2.3.2.4 Initialisation

The MSLICE optimisation problem (2.45) is non-convex, so the choice of initialisation in the EM algorithm affects the solution. Therefore, one should initialise the algorithm according to the application and the kind of solution desired. Roughly speaking, it may be beneficial to find an initialisation that is close to the desired optimum. Two generic initialisers that we have used are based on K-Means and a mixture of factor analysers (MFA) (Ghahramani and Hinton, 1996).

For our K-Means initialiser, we run the K-Means algorithm to partition the N data points into M clusters. We initialise K-Means itself by setting each of the cluster centres to a data point drawn at random. Let N_m denote the number of data points assigned to cluster m by K-Means. We set all of $\boldsymbol{\Xi}$ except the first row to zero, so that

Algorithm 2 EM for MSLICE

Initialise $\boldsymbol{\Xi}^{(0)}$
for $m \leftarrow 1 : M$ **do**
  Initialise $\boldsymbol{\Psi}_m^{(0)}$
  Initialise $\boldsymbol{\Lambda}_m^{(0)}$ such that $\boldsymbol{\Lambda}_m^{(0)} \succ 0$ and $\operatorname{diag}\left(\left(\boldsymbol{\Lambda}_m^{(0)}\right)_{zz}\right) = \mathbf{1}$
end for
for $t \leftarrow 1 : T$ **do**
  E Step
    for $m \leftarrow 1 : M$ **do**
      for $n \leftarrow 1 : N$ **do**
        $\overline{w}_{nm} \leftarrow$ Responsibility of expert m for data point n
        ▷ See Equation (2.54)
      end for
      $\overline{\boldsymbol{Z}}_m \leftarrow -\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m^{(t-1)}\right)\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{yz}\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1}$
    end for
  end E Step
  M Step
    $\boldsymbol{\Xi}^{(t)} \leftarrow \arg\max_{\boldsymbol{\Xi}} Q_{\Xi}\left(\boldsymbol{\Xi}; \boldsymbol{\theta}^{(t-1)}\right)$ ▷ See Equation (2.53)
    for $m \leftarrow 1 : M$ **do**
      $\boldsymbol{\Psi}_m^{(t)} \leftarrow \boldsymbol{\Psi}_m^{(t-1)}$
      $\boldsymbol{\Lambda}_m^{(t)} \leftarrow \boldsymbol{\Lambda}_m^{(t-1)}$
      $\boldsymbol{\Psi}_m^{(t)} \leftarrow \arg\max_{\boldsymbol{\Psi}_m} Q_{\Psi_m}\left(\boldsymbol{\Psi}_m | \boldsymbol{\Lambda}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$ ▷ See Equation (2.55)
      $N_m \leftarrow \sum_{n=1}^{N} \overline{w}_{nm}$
      $\overline{\boldsymbol{W}}_m \leftarrow \operatorname{diag}\left(\overline{\boldsymbol{w}}_{:m}\right)$
      $\boldsymbol{Y}_m \leftarrow \widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m^{(t)}$
      $\boldsymbol{S}_m \leftarrow \frac{1}{N_m}\begin{pmatrix} \boldsymbol{Y}_m^T\boldsymbol{W}_m\boldsymbol{Y}_m & \boldsymbol{Y}_m^T\boldsymbol{W}_m\overline{\boldsymbol{Z}}_m \\ \overline{\boldsymbol{Z}}_m^T\boldsymbol{W}_m\boldsymbol{Y}_m & N_m\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1} + \overline{\boldsymbol{Z}}_m^T\boldsymbol{W}_m\overline{\boldsymbol{Z}}_m \end{pmatrix}$
      $\boldsymbol{\Lambda}_m^{(t)} \leftarrow \underset{\boldsymbol{\Lambda}_m \,:\, \boldsymbol{\Lambda}_m \succeq 0,\ \operatorname{diag}\left((\boldsymbol{\Lambda}_m)_{zz}\right) = \mathbf{1}}{\arg\max} Q_{\Lambda_m}\left(\boldsymbol{\Lambda}_m | \boldsymbol{\Psi}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$
      ▷ See Equation (2.58)
    end for
  end M Step
end for

initially, the side information has no effect on the component prior. We set the first row of $\boldsymbol{\Xi}$ such that $p(w_m = 1|\boldsymbol{x};\boldsymbol{\Xi}) = \frac{N_m}{N}$. Similarly, we initialise $\boldsymbol{\Psi}_m$ to zero, except for the first row which is set such that the initial mean is equal to the empirical mean of cluster m. For the precision $\boldsymbol{\Lambda}_m$, we set $(\boldsymbol{\Lambda}_m)_{yz} = \mathbf{0}$, $(\boldsymbol{\Lambda}_m)_{zz} = \boldsymbol{I}$, and $(\boldsymbol{\Lambda}_m)_{yy}$ equal to the inverse of the empirical covariance of cluster m. Note that this can fail if K-Means produces clusters whose empirical covariance has low rank.

For our MFA initialiser, we begin by training a mixture of factor analysers. We use the code provided by Ghahramani and Hinton (1996), which initialises each factor loading matrix by drawing each element independently from a zero-mean Gaussian distribution, and sets each element of the initial noise variance equal to the empirical variance of the associated variable. MFA gives us cluster priors, so we set the first row $\boldsymbol{\Xi}$ to reproduce those, and the rest of $\boldsymbol{\Xi}$ to zero. Similarly, MFA outputs the component means, so we set the first row of each $\boldsymbol{\Psi}_m$ to the mean of one of the components, and the other elements of $\boldsymbol{\Psi}_m$ to zero. We form the precision $\boldsymbol{\Lambda}_m$ of each expert according to Equation (2.29) from the factor loadings and noise vectors of each component. As for SLICE, if we are employing the constraint $\operatorname{diag}\left((\boldsymbol{\Lambda}_m)_{zz}\right) = \mathbf{1}$, we scale the rows and columns of $\boldsymbol{\Lambda}_m$ to satisfy it.

2.4 Sparse Non-Gaussian Models

The graphical lasso and SLICE are both Gaussian models, and MSLICE is a mixture of Gaussian experts. But in practice, it is often necessary to work with non-Gaussian data. In finance, for example, stock price returns are well-known to be non-Gaussian. See, for example, (Fama, 1965). To further illustrate the point, the kurtosis of each stock's returns in the FTSE data described in Section 2.5.1 varies between 5.12 and 152; so the distribution of returns for each of those stocks is heavy-tailed.

Learning a general, high-dimensional, multivariate distribution is often difficult, but learning one-dimensional distributions is considerably easier. So one approach to the problem is to decompose the task: learn a flexible model of the one-dimensional marginals, and then use a simple, tractable model for the dependency structure. In the following sections, we study models with a Gaussian dependency structure, and a number of different marginals. These models extend the graphical lasso, SLICE, and MSLICE to non-Gaussian distributions.

2.4.1 Sparse Gaussian Copulas

The nonparanormal distribution (Liu et al., 2009) (see Section 2.1.5) extends the graphical lasso to a sparse Gaussian copula. Liu et al. (2009) use truncated empirical distributions for the marginals, which is fine for learning the dependency structure. But we are also interested in using the model to evaluate the likelihood of test data, and to generate from the model. The empirical marginal is insufficient for this purpose because it places all the probability mass at the training data points. The likelihood of a set of test data would be zero (provided at least one point in the test set is not in the training set), and if we were to generate from the model, only points in the training set would be produced. We therefore choose smooth marginal models. To distinguish this model from the nonparanormal, and for consistency with nomenclature introduced later, we refer to this copula version of graphical lasso as CopGLASSO.

SLICE can be extended in much the same way. Recall that we use F_v to denote the marginal cdf of visible variable v, and Φ to denote the standard Gaussian cdf. We introduce the CopSLICE model, in which we learn the marginals $\{F_v(\widetilde{y}_v)\}_{v=1}^V$, then use the EM learning algorithm for SLICE on the transformed data $\{\boldsymbol{y}_n\}_{n=1}^N$, where $y_{nv} = \Phi^{-1}(F_v(\widetilde{y}_{nv}))$. CopSLICE therefore uses a sparse latent Gaussian to capture the dependency structure.

2.4.1.1 Marginal Models

When learning the marginals, care must be taken regarding the tails because there will often be few data points in these regions. Our choice of marginal model is motivated by the Pickands-Balkema-de Haan theorem (Pickands, 1975; Balkema and De Haan, 1974) in extreme value theory, which states that for a large class of distributions, the conditional excess tends to a generalised Pareto distribution (GPD). Specifically, given a random variable A with distribution F, the conditional excess distribution is

$$F_t(a) \equiv P(A - t \geq a | A > t) = \frac{F(t+a) - F(t)}{1 - F(t)}, \tag{2.62}$$

and $F_t(a) \to G_{\xi,\sigma}(a)$ as $t \to \infty$, where

$$G_{\xi,\sigma}(a) = \begin{cases} 1 - \left(1 + \frac{\xi a}{\sigma}\right)^{-1/\xi} & \text{for } \xi \neq 0 \\ 1 - \exp\left(-\frac{a}{\sigma}\right) & \text{for } \xi = 0 \end{cases} \tag{2.63}$$

is the GPD. So we model the marginals with a piecewise distribution composed of three parts: a lower tail, an upper tail, and a "body" in the central section. We write the

marginal for visible variable v as follows:

$$F_v(\widetilde{y}_v) = \begin{cases} \mathcal{B}_v(t_{v-})\left(1 - \mathcal{L}_v(t_{v-} - \widetilde{y}_v)\right) & \text{for } \widetilde{y}_v < t_{v-} \\ \mathcal{B}_v(\widetilde{y}_v) & \text{for } t_{v-} \leq \widetilde{y}_v \leq t_{v+} \\ \mathcal{B}_v(t_{v+}) + \left(1 - \mathcal{B}_v(t_{v+})\right)\mathcal{U}_v(\widetilde{y}_v - t_{v+}) & \text{for } \widetilde{y}_v > t_{v+}, \end{cases} \tag{2.64}$$

where $\mathcal{B}_v$ is the body distribution, $\mathcal{L}_v$ the lower tail, and $\mathcal{U}_v$ the upper tail. Both $\mathcal{L}_v$ and $\mathcal{U}_v$ are GPDs.

For the body, we initially chose a kernel density estimator (KDE):

$$p(\widetilde{y}_v) = \frac{1}{N h_v} \sum_{n=1}^{N} K\left(\frac{\widetilde{y}_v - \widetilde{y}_{nv}}{h_v}\right), \tag{2.65}$$

where h_v is the bandwidth parameter. For the kernel function K, we used a Gaussian. This worked well for CopGLASSO and CopSLICE, but the CopMSLICE method described in Section 2.4.2 ran much faster when we used a Gaussian distribution for the body; see the experiment in Section 2.5.5.2. So we used that for CopGLASSO and CopSLICE too, for purposes of comparison.

For each marginal separately, we set each of its parameters t_{v-} and t_{v+} to the location of a data point in the training set; we determine which points by cross-validation (using training data only, not the test data used for evaluating the model). We estimate the parameters of $\mathcal{L}_v$ using only the training data points below t_{v-}, and $\mathcal{U}_v$ using only those above t_{v+}. We maximise the likelihood, given these data sets, of the parameters ξ and σ in Equation (2.63) for the respective GPD models. We do this using Matlab's `fminunc` function, initialising the parameters such that a GPD's first and second moments match their empirical values. Since σ is a scale parameter, it must remain positive. We impose the additional constraint that ξ also remain positive. If $\xi < 0$, then the GPD $G_{\xi,\sigma}(a)$ assigns zero probability to all $a > -\sigma/\xi$. But in some scenarios – such as the financial application in Section 2.5 – this is undesirable: extreme values are rare, so just because the training data fall within some range, it does not mean that more extreme values will not be observed in future. To maintain positivity of ξ and σ, we optimise the logarithms of these values.

For the body $\mathcal{B}_v$, if it is a KDE, the only parameter is the bandwidth; we set it using Matlab's `ksdensity` function, which employs a heuristic function of the standard deviation of the training data. If using a Gaussian for the body, we fit the parameters by minimising the sum of squared errors between the Gaussian cdf and the empirical distribution at each data point between t_{v-} and t_{v+}. We do this using Matlab's `fminsearch` function, initialising the Gaussian parameters to their empirical values.

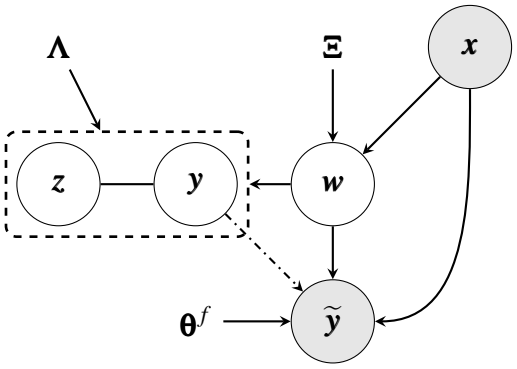


Figure 2.5: CopMSLICE, a conditional mixture of sparse latent Gaussian copulas. The model is similar to MSLICE; the main addition is the deterministic map – indicated by the dot-dashed arrow – from $\boldsymbol{y}$ to $\widetilde{\boldsymbol{y}}$. Side information $\boldsymbol{x}$ influences the choice of mixture component $\boldsymbol{w}$; variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are drawn from a joint Gaussian (with a structure that depends on the mixture component); and then $\boldsymbol{y}$ is transformed deterministically to $\widetilde{\boldsymbol{y}}$. The side information influences the mean of the resulting distribution of $\widetilde{\boldsymbol{y}}$.

2.4.2 Non-Gaussian Extension of MSLICE

In a similar manner to CopGLASSO and CopSLICE, we extend MSLICE to CopMSLICE by making each component in the mixture a Gaussian copula. The model is illustrated in Figure 2.5. The generative model is the same as that for MSLICE – see Equations (2.43) and (2.44) – except that the transformation of $\boldsymbol{y}$ to $\widetilde{\boldsymbol{y}}$ is generalised: given $w_m = 1$, we now set $\widetilde{y}_v = f_{mv}^{-1}(y_v) + \widetilde{\mu}_{mv}$, where $f_{mv} \equiv \Phi^{-1} \circ F_{mv}$, and $\widetilde{\mu}_{mv} = \left(\boldsymbol{\Psi}_m^T\right)_{v:} \widetilde{\boldsymbol{x}}$. Let $\boldsymbol{\zeta}_{mv}$ parameterise F_{mv}; let $\boldsymbol{\theta}_m^f \equiv \{\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m\}$ contain all the parameters of f_m; and let $\boldsymbol{\theta}^f \equiv \left\{\boldsymbol{\theta}_m^f\right\}$.

If the Gaussian experts have no latent variables, then we refer to this particular case of CopMSLICE as CopMGLASSO. (We likewise refer to the non-latent version of MSLICE as MGLASSO).

Training CopMSLICE is more difficult than training CopGLASSO or CopSLICE. We do not know which data points were generated by which experts, so we cannot learn the marginals F_{mv} independently of the rest of the model. Instead, we learn them during the EM algorithm, using the responsibilities inferred during the E step to weight the data points. Specifically, the EM objective is

$$Q\left(\boldsymbol{\theta}; \boldsymbol{\theta}^{(t-1)}\right) = \mathbb{E}_{\boldsymbol{\theta}^{(t-1)}}\left[\mathcal{L}\left(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}\right)\right] - \gamma(\boldsymbol{\theta}), \tag{2.66}$$

where $\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta})$ is the log likelihood,

$$\begin{aligned}\mathcal{L}(\boldsymbol{X}, \boldsymbol{Y}; \boldsymbol{\theta}) = \sum_{n=1}^{N} \sum_{m=1}^{M} w_{nm} \Big[& \log p(w_{nm} = 1 | \boldsymbol{x}_n; \boldsymbol{\Xi}) \\ & + \log p\left(\boldsymbol{u}_n | w_{nm} = 1; \boldsymbol{\theta}_m^f, \boldsymbol{\Lambda}_m\right) \\ & + \sum_v \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv}) \Big].\end{aligned} \tag{2.67}$$

See Appendix A for a full derivation. Apart from the computation of $\boldsymbol{y}_m = f_m(\widetilde{\boldsymbol{y}} - \boldsymbol{\mu}_m)$, the E step is much the same for CopMSLICE as for MSLICE: $\overline{w}_{nm}$ is evaluated according to Equation (2.54), while

$$\overline{\boldsymbol{z}}_{nm} = -\left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1} \left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zy} f_m\left(\widetilde{\boldsymbol{y}}_n - \left(\boldsymbol{\Psi}_m^{(t-1)}\right)^T \widetilde{\boldsymbol{x}}_n; \boldsymbol{\zeta}_m^{(t-1)}\right). \tag{2.68}$$

Compare this with Equation (2.57) for MSLICE.

The optimisation of $\boldsymbol{\Xi}$ in the M step can be done separately from the other parameters, and in precisely the same way as for MSLICE. The rest of the parameters are coupled in the objective (2.66). We condition on $\boldsymbol{\Lambda}$ while we optimise the objective with respect to $\boldsymbol{\theta}^f$, and vice versa.

2.4.2.1 The Component Precisions

Let $\boldsymbol{Y}_m = f_m\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)$, where f_m is defined to operate on the rows of a matrix argument. Define $\boldsymbol{S}_m$ such that

$$(\boldsymbol{S}_m)_{yy} = \frac{1}{N_m} \boldsymbol{Y}_m^T \boldsymbol{W}_m \boldsymbol{Y}_m, \tag{2.69}$$

$$(\boldsymbol{S}_m)_{yz} = \frac{1}{N_m} \boldsymbol{Y}_m^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m, \tag{2.70}$$

$$(\boldsymbol{S}_m)_{zz} = (\boldsymbol{\Lambda}_m)_{zz}^{-1} + \frac{1}{N_m} \overline{\boldsymbol{Z}}_m^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m. \tag{2.71}$$

Then the objective for $\boldsymbol{\Lambda}_m$ can be written

$$Q_{\Lambda_m}\left(\boldsymbol{\Lambda}_m | \boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m; \boldsymbol{\theta}^{(t-1)}\right) \equiv \frac{N_m}{2} \left[\log\det \boldsymbol{\Lambda}_m - \operatorname{tr}(\boldsymbol{S}_m \boldsymbol{\Lambda}_m)\right] - \gamma_{\Lambda_m}(\boldsymbol{\Lambda}_m). \tag{2.72}$$

The constraints on $\boldsymbol{\Lambda}_m$ are the same as in MSLICE, so the optimisation problem is the same constrained graphical lasso.

2.4.2.2 The Gaussianising Functions

The objective for $\boldsymbol{\theta}_m^f$ is

$$\begin{aligned} Q_{\theta_m^f}\left(\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m | \boldsymbol{\Lambda}_m; \boldsymbol{\theta}^{(t-1)}\right) \equiv & -\frac{N_m}{2}\left\{\operatorname{tr}\left((\boldsymbol{S}_m)_{yy}(\boldsymbol{\Lambda}_m)_{yy}\right) + 2\operatorname{tr}\left((\boldsymbol{S}_m)_{zy}(\boldsymbol{\Lambda}_m)_{yz}\right)\right\} \\ & + \sum_n \overline{w}_{nm} \sum_v \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv}) \\ & - \gamma_{\Psi_m}(\boldsymbol{\Psi}_m) - \gamma_{\zeta_m}(\boldsymbol{\zeta}_m). \end{aligned} \tag{2.73}$$

Note that $\boldsymbol{S}_m$ depends on $\boldsymbol{\theta}_m^f$ through $\boldsymbol{Y}_m$. If the marginal models F_{mv} are chosen to be simple, the gradients of $Q_{\theta_m^f}$ with respect to the components of $\boldsymbol{\theta}_m^f$ may be available, but in general this is not the case. We take a very simple approach to increasing the objective during the M step. We consider each parameter separately (conditioned on all others), and look at three values: the current value, and two values a fixed distance either side of it. The value that achieves the highest $Q_{\theta_m^f}$ is the new value of the parameter. There will, of course, be more principled, more efficient, ways to set $\boldsymbol{\theta}^f$, but this simple method is sufficient to demonstrate and evaluate CopMSLICE.

2.4.2.3 The Marginal Models

For the marginals, we use a piecewise distribution composed of a body and two tail distributions as presented in Section 2.4.1.1. For the body, our initial idea was to use the kernel density estimator (2.65) as we did with the single-component models. But there are two problems with this. The first is that each KDE is modelling a marginal within a mixture component, so it should use the data generated by that mixture component only – but we do not know which components generated which data points. So in the M step, we weight each data point within a component according to the responsibilities computed in the E step. This means that we only approximate the maximisation of (2.73), and we note that this may decrease the EM objective. The second problem with the KDE is that it is very slow – see the experiment in Section 2.5.5.2 – because $\boldsymbol{Y}_m$ must be re-evaluated whenever $\boldsymbol{\theta}_m^f$ changes, and the KDE evaluation is expensive. So we replaced the KDE in the body with a simpler model: a Gaussian. Although much faster, evaluation of the Gaussian cdf is still a bottleneck. A logistic distribution may be better because, while similar to the Gaussian, its cdf is faster to evaluate. But that is left for future work.

The EM algorithm for CopMSLICE is summarised in Algorithm 3. We initialised

Algorithm 3 EM for CopMSLICE

Initialise $\boldsymbol{\Xi}^{(0)}$

for $m \leftarrow 1 : M$ **do**

Initialise $\boldsymbol{\Lambda}_m^{(0)}$ such that $\boldsymbol{\Lambda}_m^{(0)} \succ 0$ and $\operatorname{diag}\left(\left(\boldsymbol{\Lambda}_m^{(0)}\right)_{zz}\right) = \mathbf{1}$

Initialise $\boldsymbol{\Psi}_m^{(0)}$ and $\boldsymbol{\zeta}_m^{(0)}$

$\boldsymbol{Y}_m \leftarrow f_m\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m^{(0)}; \boldsymbol{\zeta}_m^{(0)}\right)$

end for

for $t \leftarrow 1 : T$ **do**

E Step

for $m \leftarrow 1 : M$ **do**

for $n \leftarrow 1 : N$ **do**

$\overline{w}_{nm} \leftarrow$ Responsibility of expert m for data point n

▷ See Equation (2.54)

end for

$\overline{\boldsymbol{Z}}_m \leftarrow -\boldsymbol{Y}_m \left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{yz} \left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1}$

end for

end E Step

M Step

$\boldsymbol{\Xi}^{(t)} \leftarrow \arg\max_{\boldsymbol{\Xi}} Q_{\Xi}\left(\boldsymbol{\Xi}; \boldsymbol{\theta}^{(t-1)}\right)$ ▷ See Equation (2.53)

for $m \leftarrow 1 : M$ **do**

$\boldsymbol{\Psi}_m^{(t)} \leftarrow \boldsymbol{\Psi}_m^{(t-1)}$; $\boldsymbol{\zeta}_m^{(t)} \leftarrow \boldsymbol{\zeta}_m^{(t-1)}$; $\boldsymbol{\Lambda}_m^{(t)} \leftarrow \boldsymbol{\Lambda}_m^{(t-1)}$

$\left(\boldsymbol{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}\right) \leftarrow \arg\max_{(\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m)} Q_{\theta_m^f}\left(\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m | \boldsymbol{\Lambda}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$

▷ See Equation (2.73)

$N_m \leftarrow \sum_{n=1}^{N} \overline{w}_{nm}$

$\boldsymbol{W}_m \leftarrow \operatorname{diag}\left(\overline{\boldsymbol{w}}_{:m}\right)$

$\boldsymbol{Y}_m \leftarrow f_m\left(\widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m^{(t)}; \boldsymbol{\zeta}_m^{(t)}\right)$

$$\boldsymbol{S}_m \leftarrow \frac{1}{N_m} \begin{pmatrix} \boldsymbol{Y}_m^T \boldsymbol{W}_m \boldsymbol{Y}_m & \boldsymbol{Y}_m^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m \\ \overline{\boldsymbol{Z}}_m^T \boldsymbol{W}_m \boldsymbol{Y}_m & N_m \left(\boldsymbol{\Lambda}_m^{(t-1)}\right)_{zz}^{-1} + \overline{\boldsymbol{Z}}_m^T \boldsymbol{W}_m \overline{\boldsymbol{Z}}_m \end{pmatrix}$$

$$\boldsymbol{\Lambda}_m^{(t)} \leftarrow \underset{\boldsymbol{\Lambda}_m \,:\, \boldsymbol{\Lambda}_m \succeq 0,\ \operatorname{diag}\left((\boldsymbol{\Lambda}_m)_{zz}\right) = \mathbf{1}}{\arg\max} Q_{\Lambda_m}\left(\boldsymbol{\Lambda}_m | \boldsymbol{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$$

▷ See Equation (2.72)

end for

end M Step

end for

the parameters in the same way as for MSLICE. For the marginals, we set the initial variance to one.

2.5 Evaluation

2.5.1 The Data

From Yahoo Finance, we obtained the closing price data for all composites of the FTSE 100 index over the period April 2005 to October 2011. We note that this period includes the market crash of late 2008. The 100 companies that compose the index changes over time, so the price data for some assets did not cover the whole of this time period. We removed any such assets, and we removed any dates on which at least one company's price was missing. We converted the prices to returns by dividing each asset's price by its value on the previous day. Stock splits and rights issues were supposedly built in to the downloaded prices, but these had sometimes been missed, resulting in obviously erroneous returns. We fixed these manually, along with some other clear errors in the data. The final returns data set consisted of 81 assets across 1633 days. The company name and market sector of each company in this data set is listed in Appendix B.

We also downloaded the VIX volatility index for the days immediately prior to each day in the returns data, to use as side information. The VIX is based on the S&P 500 index option prices, and is one measure of expected near-term market volatility. Higher values indicate that the market is more volatile.

We formed a reduced version of the data set in which we selected FTSE composites from three industries: banking, mining, and consumer goods. A data set with fewer dimensions is useful because the experiments can be run more quickly, and the resulting structures are easier to visualise. There were 19 companies in the reduced set.

2.5.2 Comparison of Single-Component Methods

Our first experiment involves only the five single-component (non-mixture) methods SLICE, CopSLICE, GLASSO, CopGLASSO, and SLR. We compare these methods at different levels of sparsity to investigate whether the SLICE model's explicit inclusion of latent variables is useful, and to find out if the copula models perform better than the Gaussian versions.

When training, we did not penalise diagonal elements of the precision in any of the models, so $\Gamma_{ii} = 0$. For (Cop)GLASSO, we penalised all off-diagonal elements equally: $\Gamma_{ij} = \gamma$. For (Cop)SLICE, we penalised the elements of $\mathbf{\Lambda}_{yy}$ differently from the rest: $\Gamma_{ij} = \gamma_1$ for $i, j \leq V$, and $\Gamma_{ij} = \gamma_2$ otherwise. We did this because we found from experience that weaker penalties are required on the connections from the latent variables: with a single uniform penalty, all elements of $\mathbf{\Lambda}_{yz}$ often go to zero in training, effectively pruning the latents. To illustrate this effect, consider the following two precision matrices where $V = 3$ and $H = 1$:

$$\mathbf{\Lambda}^{(1)} = \begin{pmatrix} \boldsymbol{I} & \alpha\mathbf{1} \\ \alpha\mathbf{1}^T & 1 \end{pmatrix}; \quad \mathbf{\Lambda}^{(2)} = \begin{pmatrix} \boldsymbol{I} - \alpha^2\mathbf{1}\mathbf{1}^T & \mathbf{0} \\ \mathbf{0}^T & 1 \end{pmatrix}. \tag{2.74}$$

These precisions are equivalent in terms of training data likelihood. But they will incur different $L1$ penalties. If $|\alpha| < 1$, then $|\alpha^2| < |\alpha|$. If the penalties are uniform across the precision matrix, $\mathbf{\Lambda}^{(2)}$ will incur smaller total penalty than $\mathbf{\Lambda}^{(1)}$, so the objective (2.23) will be greater for $\mathbf{\Lambda}^{(2)}$.

As a measure of sparsity, we count the number of non-zero elements in the upper triangle of the precision matrix (excluding the diagonal) of the (Cop)SLICE and (Cop)GLASSO models. But SLR does not explicitly represent the latent variables, so we first form a joint precision $\mathbf{\Lambda}^{SLR}$ as follows. We set $\mathbf{\Lambda}^{SLR}_{yy} = \boldsymbol{M}$, where $\boldsymbol{M}$ is the sparse matrix in the SLR method; see Section 2.1.3.1. SLR's low rank matrix can be written $\boldsymbol{L} = \mathbf{\Lambda}^{SLR}_{yz}\left(\mathbf{\Lambda}^{SLR}_{zz}\right)^{-1}\mathbf{\Lambda}^{SLR}_{zy}$. We perform a singular value decomposition on $\boldsymbol{L}$ which results in $\boldsymbol{L} = \boldsymbol{U}\boldsymbol{D}\boldsymbol{U}^T$, where $\boldsymbol{D}$ is a diagonal matrix with non-negative elements. Define $I \equiv \{i : D_{ii} \geq t\}$, where t is a small threshold; we used $t = 0.001$. We approximate the rank of $\boldsymbol{L}$ as $|I|$, and set $\mathbf{\Lambda}^{SLR}_{zz} = \boldsymbol{D}^{-1}_{II}$, $\mathbf{\Lambda}^{SLR}_{yz} = \boldsymbol{U}_{:I}$.

We use the reduced data set of 19 assets, and split the data into a training set consisting of the first 1116 consecutive days[3], and a testing set consisting of the remaining 517 days. We preprocessed the data by subtracting the mean of the training set from all data points, and scaling such that the empirical precision of the training data had ones along the main diagonal.

For each model, we train multiple times with different limits on the sparsity of the trained model. To do this, we choose small initial $L1$ penalties and train the model; if this results in a model that is below the sparsity limit, we increase the penalties by 5% and retrain. The limits used were $(25, 35, 50, 75, 100, 125)$.

[3] The training set covers a period from April 2005 to September 2009, and so includes the market crash of late 2008.

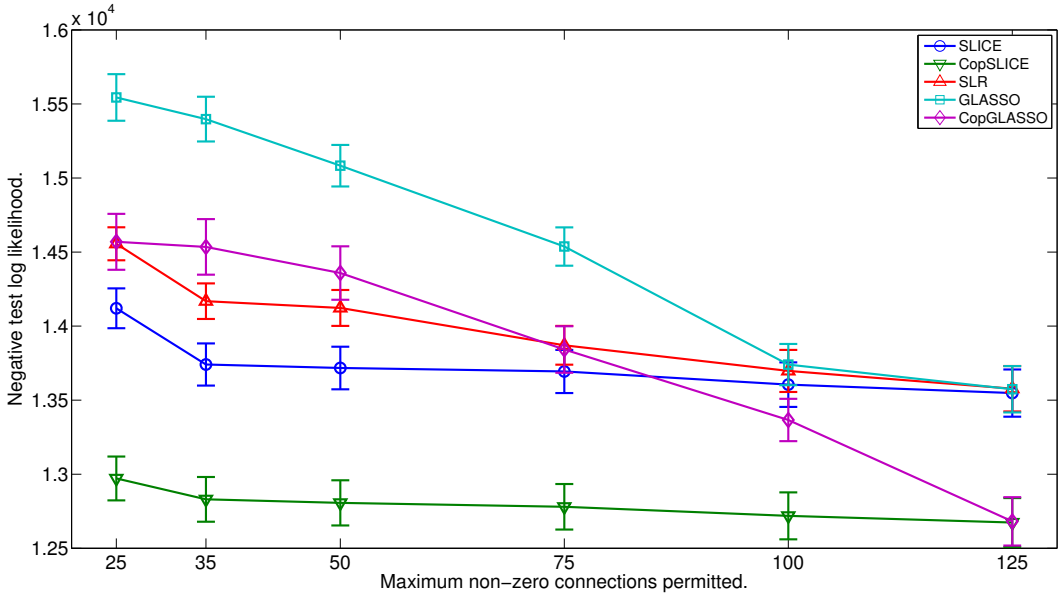


Figure 2.6: Comparison of the five single-component methods at different levels of sparsity. The metric is negative log likelihood on the test data set (so lower is better). The bars represent standard errors. The copula methods outperform their non-copula counterparts. The performance of the methods without latent variables degrades quickly as sparsity increases (to the left of the figure); the performance of the latent-variable methods degrades more slowly.

We set the parameters of each method by six-fold cross-validation (using the training data only) over a grid covering a reasonable range for each parameter. For each parameter combination, we train to each sparsity limit, and compute the mean log likelihood of the held-out fold over all sparsity limits. We choose the combination with the highest mean log likelihood. We then train on the full training set at each sparsity limit, and compute the log likelihood of the test set under the resulting distributions.

The results are shown in Figure 2.6. It is clear that the copula versions of SLICE and GLASSO perform better than the standard versions at all sparsity limits. CopSLICE is the best performer overall. When the precision is allowed to be dense, the two copula methods perform similarly, as do the three non-copula methods. But as the sparsity limit is lowered, the performance of GLASSO and CopGLASSO degrades more quickly than the models that incorporate latent variables. This suggests that the inclusion of latent variables facilitates more parsimonious representations. At the lower sparsity limits, SLICE outperforms SLR, although this may be due to our use of the SVD to form the joint precision for SLR: it may be possible to find a sparser decomposition of SLR's low-rank matrix.

2.5.3 Evaluation of Multi-Component Methods

The next experiment is designed to evaluate the mixture of expert models. The goals are to investigate whether the inclusion of side information and multiple components leads to better performance, and to find out if the learned structures may be interpretable. To these ends, we include the following models in the comparison: GLASSO, SLR, SLICE, MSLICE, CopMSLICE, SLICE with side information (which we denote SLICE+Side), and MSLICE without side information (denoted MSLICE-NoSide). Note that the model SLICE+Side is equivalent to MSLICE with a single component, and that MSLICE-NoSide is a mixture model (as opposed to a mixture of experts).

We preprocess the returns data as in Section 2.5.2. Here, we also preprocess the side information to make it zero mean and unit variance. We penalise the precision matrices as described in Section 2.5.2, but we do not penalise the MSLICE parameters $\boldsymbol{\Xi}$ or $\boldsymbol{\Psi}$. Feature selection in the side information is investigated in the next chapter; see Section 3.3.3.2. Again, we use the reduced financial data set described in Section 2.5.1, with the same split into training and testing sets used in the single-component evaluation experiment. We train all the models over a range of precision penalties. Any additional parameters are set as follows. We run 6-fold cross-validation over a grid of plausible values, and for each combination compute the log likelihood of the held-out fold. We do this for each precision penalty value, and average over both folds and penalties.

Figure 2.7 shows the negative log likelihood of the test data plotted against the range of penalties. The meaning of the penalty parameter on the x-axis is different in each model, so one should compare entire curves on this figure (as opposed to considering any particular vertical cross-section). However, we tried to give this parameter an analogous interpretation in each model. In GLASSO, the meaning is clear because we use a single penalty value on all off-diagonal elements. For SLICE, MSLICE, and CopMSLICE, the x-axis parameter is the penalty applied to $\boldsymbol{\Lambda}_{yy}$ in each component. For SLR, it is the penalty applied to the sparse matrix.

The dot-dashed black line in the figure is the negative log likelihood of the test data under a Gaussian model trained by maximum likelihood; the dashed black lines either side of it are the error bars. For visual clarity, SLICE+Side and SLR are not shown on the figure. The curve for SLICE+Side sits almost exactly on top of the SLICE curve. Using the cross-validated value for SLR's γ parameter, the SLR curve is almost exactly the same as GLASSO's. That is, SLR learns to use the sparse matrix in preference to

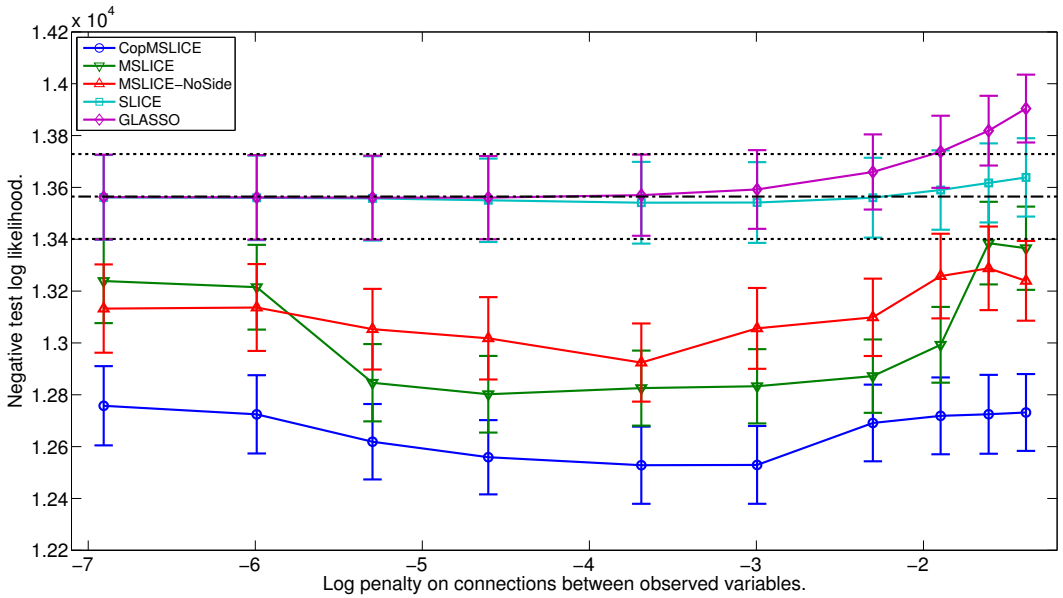


Figure 2.7: Comparison of five methods by negative log likelihood (lower is better) of test data over a range of penalties. The bars represent standard errors. SLICE and GLASSO perform similarly; the mixture models outperform the single-component methods; side information improves the performance of MSLICE; and CopMSLICE is best performer overall.

the low-rank matrix. If the value of γ for which SLR performs best on the test data is used, the SLR curve sits slightly above the SLICE curve.

On this data set, GLASSO and SLICE perform no better than the non-sparse maximum likelihood model. (But note that their sparsity could be advantageous in some applications if it makes them more interpretable). The fact that SLICE and SLICE+Side perform similarly tells us that the side information (the VIX in this case) is of little use in predicting the Gaussian mean. The mixture models all perform much better than the single component models. MSLICE outperforms MSLICE-NoSide, which shows that the VIX is useful for selecting a mixture component. CopMSLICE is the best performer: the copula addition improves on MSLICE, similarly to how CopSLICE improved over SLICE in the single component experiment of Section 2.5.2.

We now examine the MSLICE model that performed best on the test set to see if we can interpret the learned structures. Cross-validation resulted in a model with three experts; Figure 2.8 shows the responsibility of each expert for each data point in the whole data set (both training and testing data), plotted alongside the VIX. We see that the model has learned to associate each structure with a different level of market volatility. The first expert (second panel in the figure) is responsible for most of

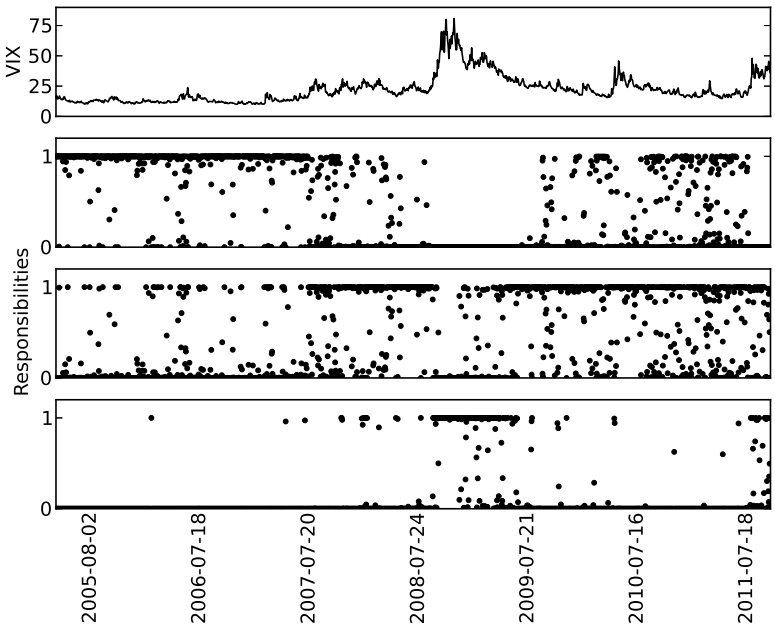


Figure 2.8: Panels 2-4 show the responsibility of experts 1-3 for each data point. Plotted alongside the VIX (top panel), it is clear that the experts 1, 2, and 3 tend to assume responsibility when the VIX is low, medium, and high-valued respectively.

the low-volatility regions, the second expert (third panel) is responsible for the mid-volatility regions, and the third expert (bottom panel) takes responsibility when volatility is highest.

The precision matrices $\mathbf{\Lambda}_1$, $\mathbf{\Lambda}_2$, and $\mathbf{\Lambda}_3$ are illustrated in the Hinton diagrams in Figures 2.9, 2.10, and 2.11 respectively. The companies are arranged by industry: the first 5 are banks, the following 8 are mining companies, and the final 6 produce consumer goods. Blue lines separate the industries. Green boxes represent positive values, while red boxes are negative values. In expert 1, most of the high magnitude entries are in $(\mathbf{\Lambda}_1)_{yz}$, with a few smaller entries in the residual $(\mathbf{\Lambda}_1)_{yy}$. So most of the low-volatility data can be explained by the latent variables. We see some correspondence between the entries of $(\mathbf{\Lambda}_1)_{yz}$ and the industries: the first and third latent variables are coupling most of the banks and mining companies, while the second variable is serving mostly to couple the consumer goods companies. Looking at the residuals $(\mathbf{\Lambda}_1)_{yy}$, the two largest entries are BATS-IMT, and BARC-RRS. The first is easily interpretable: BATS and IMT are two large tobacco companies, so their fortunes are obviously intertwined. The second might be explained by a relatively large shareholding of the Barclays group

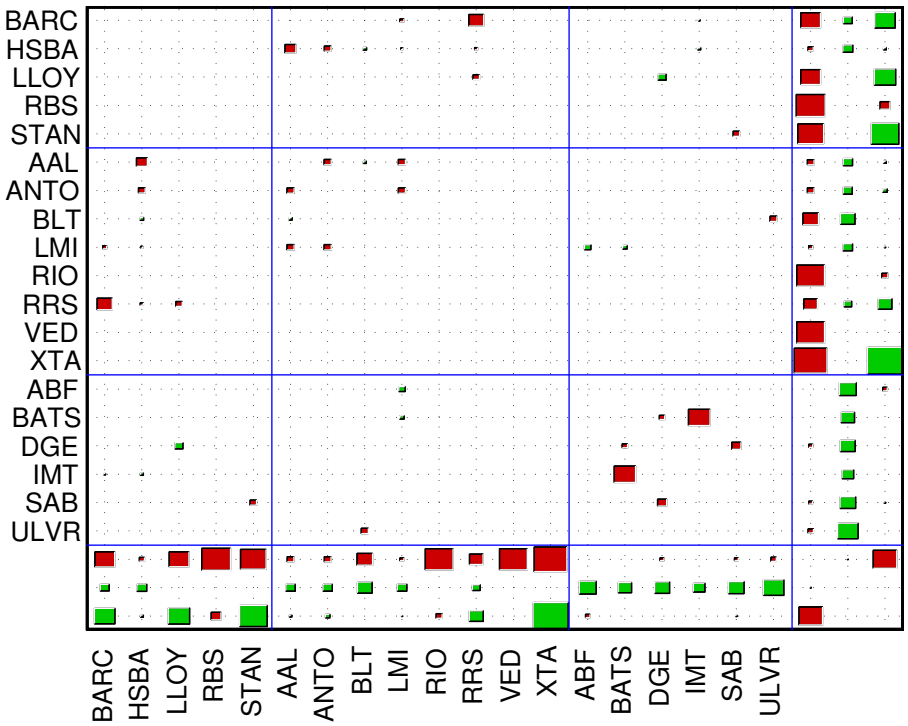


Figure 2.9: Precision matrix $\mathbf{\Lambda}_1$ of expert 1.

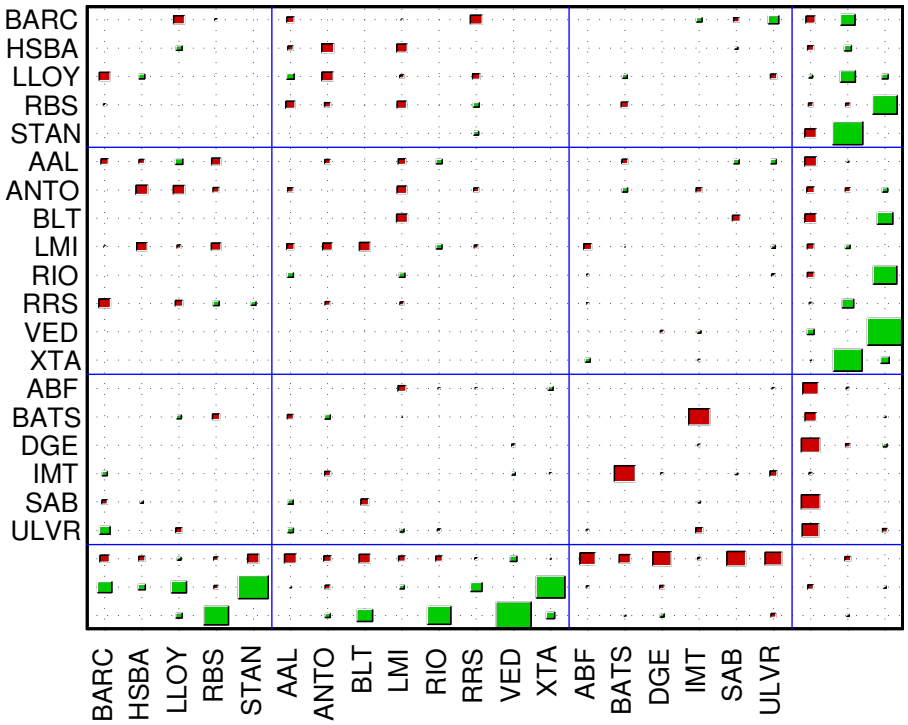


Figure 2.10: Precision matrix $\mathbf{\Lambda}_2$ of expert 2.

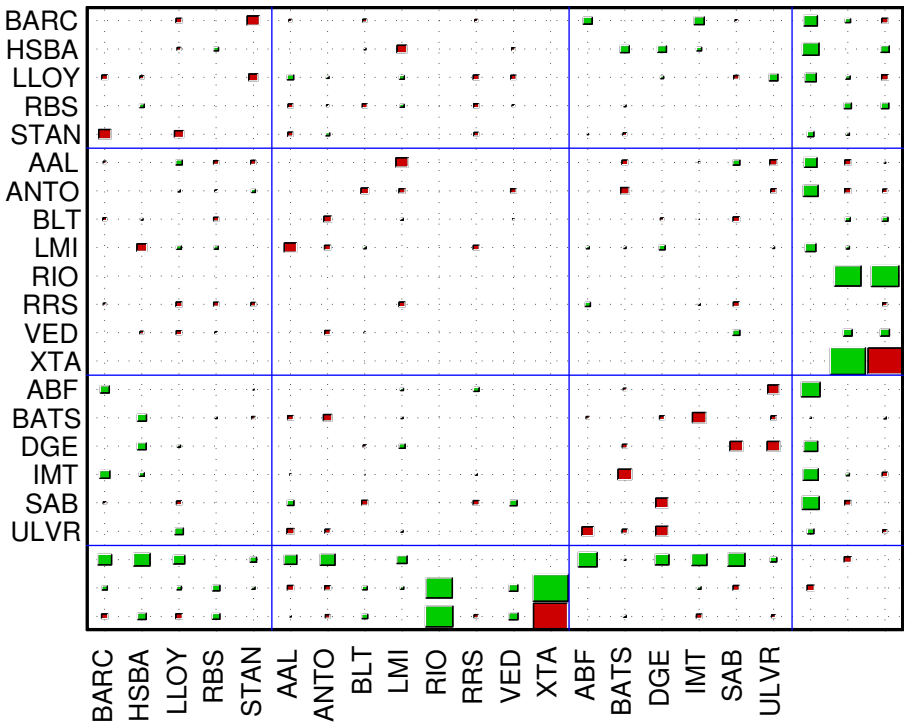


Figure 2.11: Precision matrix $\mathbf{\Lambda}_3$ of expert 3.

in Rangold Resources[4].

Looking at the precision $\mathbf{\Lambda}_2$ of the mid-volatility expert, the first latent variable is mostly used to couple the consumer goods companies. But otherwise, there is less clear structure here than in $\mathbf{\Lambda}_1$. There are more non-zero entries in the residual of this component than in the low-volatility residual. In the high-volatility expert, the precision $\mathbf{\Lambda}_3$ shows a similar pattern, but even more pronounced: there is very little structure in $(\mathbf{\Lambda}_3)_{yz}$, and there are many non-zero entries in the residual – in particular, many connections in the residual cross the industry boundaries. In summary, it appears that the market is more structured when volatility is lower; as volatility increases, this structure breaks down, and many assets become correlated.

2.5.4 Comparison on High-Dimensional Data

We now run the GLASSO, SLICE, and MSLICE on the full FTSE 100 data set consisting of 81 assets. We compare the results to the low-dimensional case, and look at the cost in computation time of the more complex models (although we study efficiency in more detail in Section 2.5.5).

[4] At time of writing, the analysis of shareholding report for RRS in 2010 can be found here: `http://www.randgoldresources.com/randgold/content/en/analysis-of-shareholding`

Table 2.1

Method	**Training Time (s)**	**Negative Test Log Likelihood**
GLASSO	4.10	111.88 (0.90)
SLICE	92.5	110.15 (0.99)
MSLICE	2310	105.58 (0.96)

We split the data into a training and testing set in the same way as the previous experiments, so these sets contained 1116 and 517 points respectively. For GLASSO, we ran 6-fold cross-validation to set γ, the penalty on the off-diagonal elements of the precision. For SLICE and MSLICE, we did no cross-validation: we simply set $\gamma_1 = \gamma$, and $\frac{\gamma_1}{\gamma_2}$ to the value used in Section 2.5.3 (where γ_1 and γ_2 are, respectively, the penalties on $\mathbf{\Lambda}_{yy}$ and the rest of $\mathbf{\Lambda}$ – see Section 2.5.2). We used 8 latent variables, because the stocks come from 8 market sectors; and we ran 200 iterations of EM. For MSLICE, we used 3 experts – again, the same as in Section 2.5.3.

The results are shown in table 2.1. Similarly to the lower-dimensional experiment of Section 2.5.3, MSLICE outperforms the single-component models in terms of test log likelihood. It also learns to associate each of its experts with a different level of market volatility: see Figure 2.12. SLICE only performs slightly better than GLASSO – again similarly to the low-dimensional experiment – which is expected since both are marginally Gaussian. However, SLICE has learned a much sparser representation, as is seen by comparing the precision matrices of the two models in Figure 2.13. The blue lines separate the 8 market sectors. Most of the dependencies in the SLICE model are accounted for by the latent variables: there is little residual structure.

2.5.5 Evaluation of Computational Costs

In this section, we study the computational costs of SLICE and its extensions. We begin with some theoretical statements about the training time of each method. Then, in the following subsections, we describe some empirical results.

Let T_Λ denote the cost of optimising the constrained graphical lasso objective (2.23). In our practical experience, this is the dominant cost. Since it depends strongly on the choice of optimiser, a thorough analysis of the additional costs is of limited value – but we make some observations below. Notice that T_Λ is independent of the

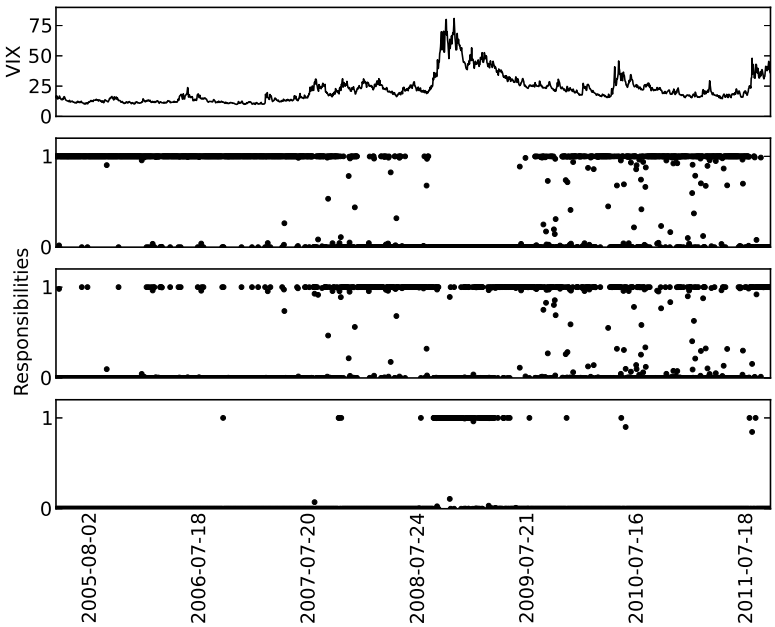


Figure 2.12: Much as in the lower-dimensional experiment, experts 1-3 (panels 2-4) assume responsibility when the VIX (top panel) is low, medium, and high-valued respectively. Compare with Figure 2.8.

number of data points N. Some optimisers may make use of sparsity to improve efficiency, and so T_Λ may depend on the penalties $\mathbf{\Gamma}_\Lambda$ and on the data itself. Of the optimisation algorithms we considered, LogdetPPA scaled best with dimension; we refer the interested reader to (Wang et al., 2010) for theoretical and empirical results on the convergence properties of this algorithm.

Recall that CopSLICE runs a preprocessing step in addition to SLICE. This step scales linearly with V (the number of visible variables) since the marginals are learned independently. The scaling of SLICE is clearly worse than $O(V)$, so CopSLICE and SLICE scale equivalently with dimension.

The E step of the SLICE algorithm involves computing $\overline{\boldsymbol{Z}}$ – see Equation (2.40) – which requires a matrix inversion[5] and matrix multiplications[5]. The computational

[5] Inversion of an $a \times a$ matrix can be computed in time $O(a^3)$. Multiplication of an $a \times b$ matrix with a $b \times c$ matrix is $O(abc)$. Algorithms with lower complexity exist for both inversion and multiplication, but they may be unstable or only useful in very high dimensions. Paolo Bientinesi (personal communication, 25th November 2013) says of matrix inverse algorithms: "... basically none of those techniques are actually used in practice. On the one hand, the constants hidden in the $O()$ notation make these results valid only asymptotically. On the other hand, the numerical stability of these algorithms is known to be an issue. The only algorithm that sometimes is used is Strassen's: $O(n^{2.8})$. In this case, the advantages due to a lower complexity are noticeable even for relatively small matrices, but in practice it is not commonly used because of its numerical stability (although in most cases it would be perfectly fine)."

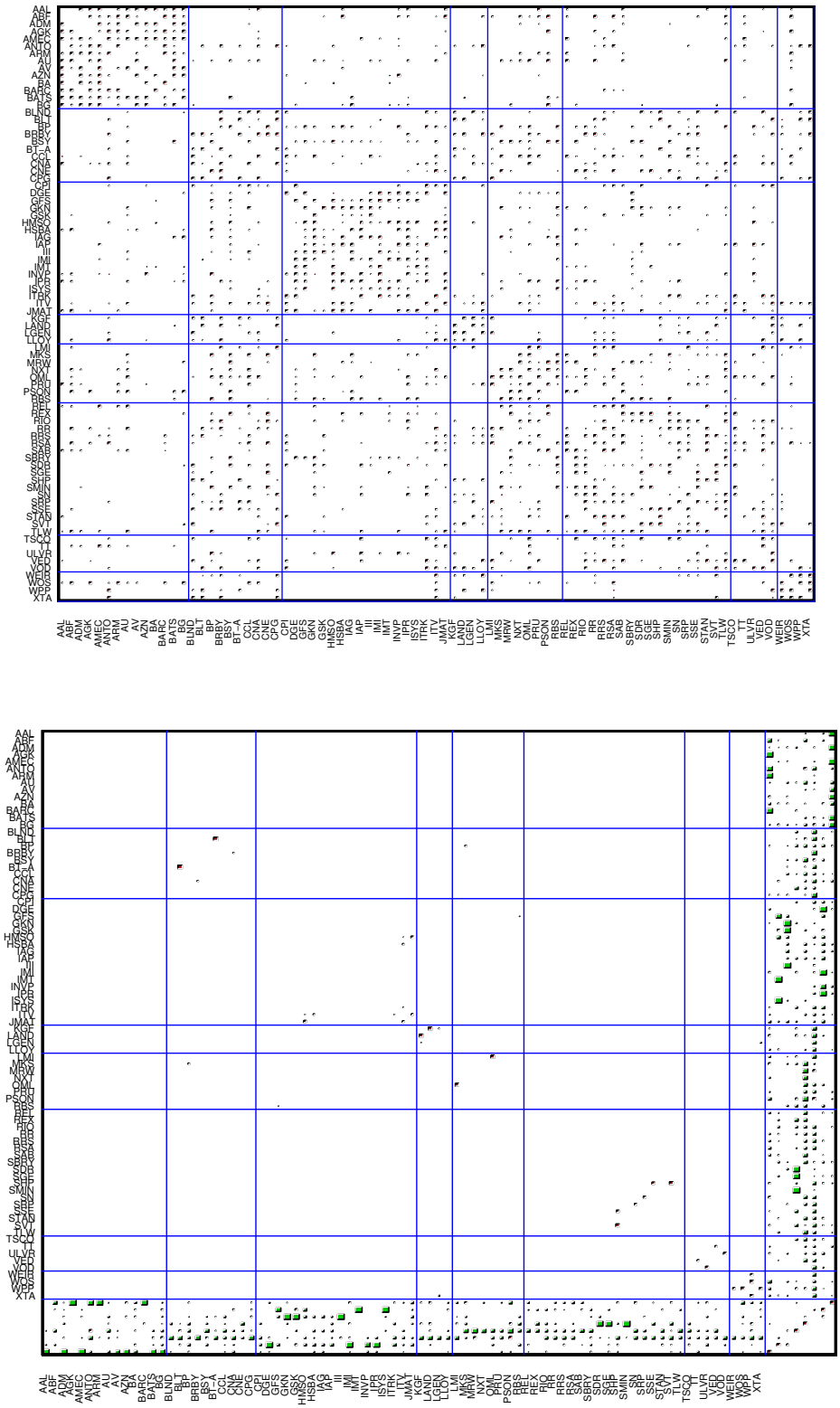


Figure 2.13: Precision matrices learned by GLASSO (top) and SLICE (bottom). SLICE is much sparser: the latent variables account for most of the dependencies.

cost is $O\left(H\left(H^2+VN+H\min(V,N)\right)\right)$, where H is the number of hidden variables. In the M step, computing the expected empirical covariance $\boldsymbol{S}$ costs $O\left(D^2N\right)$, where $D \equiv V+H$ is the dimensionality of the joint Gaussian. So the SLICE computation time is linear in N. We note the quadratic term in V and the cubic term in H (either of which may be slightly improved by efficient matrix algorithms[5], or worsened in the T_Λ term).

For MSLICE, C (the dimensionality of the side information) and M (the number of mixture components) affect the computation time. We observe the following.

- Computing $\overline{\boldsymbol{Z}}_m$ is similar to computing $\overline{\boldsymbol{Z}}$ in SLICE, except that side information is now used. The side contributes terms linear in C to the time complexity: see Equation (2.57).
- The responsibilities require that the likelihood of each data point be computed (under each component) – see Equation (2.54) – which is linear in N.
- Updating $\boldsymbol{\Xi}$ in the M step by maximising (2.53) is the sparse multinomial logit problem, whose cost is $O(NCM)$ (Krishnapuram et al., 2005).
- To update $\{\boldsymbol{\Psi}_m\}$, the objective (2.55) is maximised. Computing the matrix products prior to optimisation is linear in N. The optimisation itself depends on the choice of optimiser, so we write the cost as T_Ψ. Evaluating the objective each time $\boldsymbol{\Psi}_m$ changes is quadratic in C.
- Optimisation of each $\boldsymbol{\Lambda}_m$ is similar to the single-component case, so there is a term T_Λ for each component.
- The dependence on M is clearly linear: all computations are repeated for each component (except for the update of $\boldsymbol{\Xi}$, which is also linear in M – see above).

In summary, the total cost is linear in M and N. As for SLICE, the optimisations dominate in our practical experience: usually T_Λ is dominant, but since it does not depend on C while T_Ψ does, the $\boldsymbol{\Psi}_m$ optimisations could become dominant when C is large.

For CopMSLICE, the run time will continue to scale linearly with M. Beyond that, the cost depends strongly on the choice of marginal models. Unlike SLICE and MSLICE, the optimisation costs may depend on N. It is difficult to make further useful statements about CopMSLICE. Instead, we make some empirical studies in the following sections.

2.5.5.1 Rate of Convergence

We investigate empirically the rate of convergence of SLICE, MSLICE, and CopMSLICE. Using the same data and the same parameters as in Section 2.5.3, we recorded three metrics during training: log likelihood on the training data, log likelihood on the testing data, and sparsity of the precision matrices. We define the sparsity of a precision as the fraction of its off-diagonal elements that are non-zero (so a lower value is more sparse). For the mixture models, we record the mean sparsity of the mixture components. We trained each model over a range of penalty values – the same values as in Section 2.5.3. We found the results to be broadly similar across penalties, so in Figure 2.14 we show the results with just a single penalty. The figure illustrates the trade-off between the better performance of the more complex models, and the time required to train them. In each subfigure, we show the performance of GLASSO – indicated by a red line – for comparison. We record only the final result from GLASSO, hence the red lines are flat. GLASSO completed in 0.064 seconds. For each model, we use the same optimiser (L1General2_PSSgb) with its default settings.

The training log likelihoods converge quickly; test log likelihoods change rapidly at first, quickly outperforming GLASSO (although SLICE and GLASSO are within a standard error of each other), then continue to change more slowly; the models also continue slowly to become more sparse. We note that there is some overfitting here: each method continues to decrease the negative training likelihood despite the negative test likelihood having begun to move upward. Notice also that each of the three methods improves on its initialisation in terms of test log likelihood.

2.5.5.2 Marginal Models for CopMSLICE

For CopGLASSO and CopSLICE, the marginals are learned in a preprocessing step. For these models, learning the marginals does not greatly increase the training time. For CopMSLICE, the marginals are updated during EM, so CopMSLICE typically runs much slower than MSLICE, as illustrated in Figure 2.14. Here, we investigate empirically the training time of CopMSLICE with our two choices of marginal model, and we also look at how the training time scales with the size of the training set.

Using the same parameters as in Section 2.5.5.1, including the same penalty used to generate Figure 2.14, we ran CopMSLICE with both a Gaussian and a KDE as the model for the body of the marginals. In each case, we varied the size of the training set, and recorded the time taken to run 10 iterations of EM. The mean times per iteration,

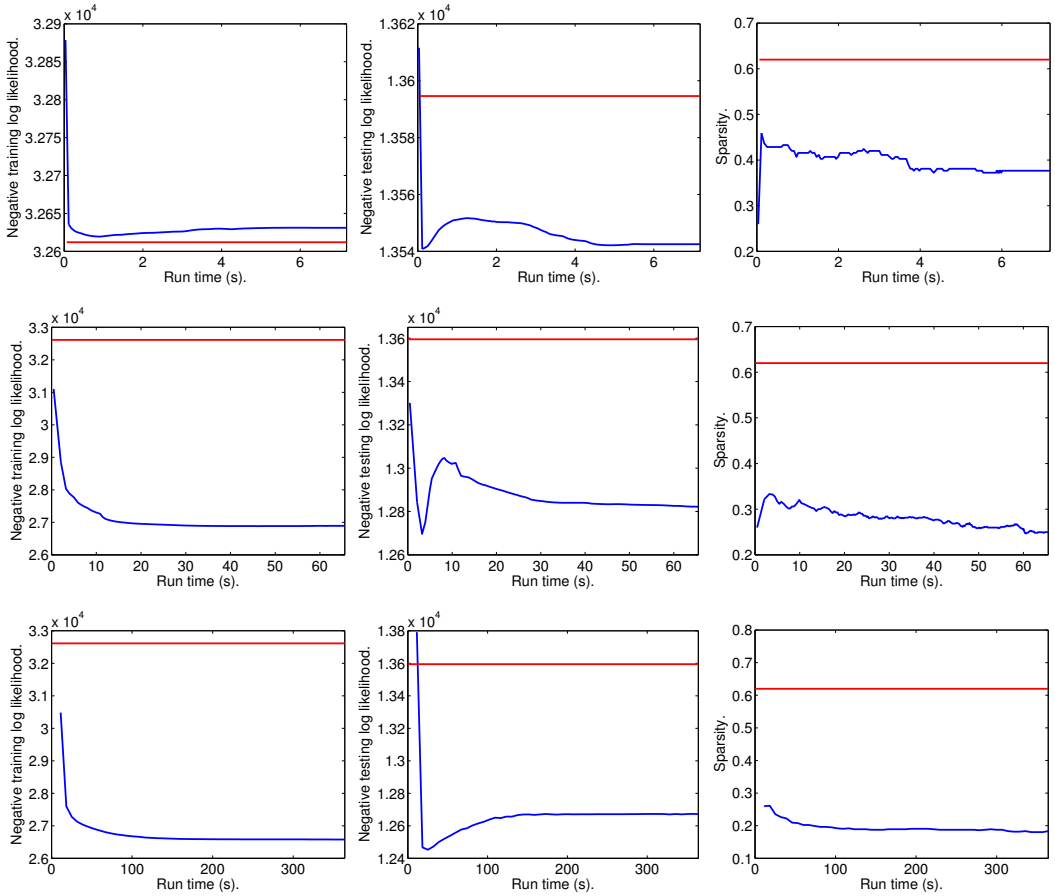


Figure 2.14: Graphs for SLICE (top row), MSLICE (middle row), and CopMSLICE (bottom row) showing the evolution, during training, of the negative log likelihood of training data (left column), negative log likelihood of testing data (middle column), and sparsity (right column). The sparsity metric is the fraction of a model's precision matrix elements that are non-zero (so lower is more sparse). The flat red line in each graph shows the performance of GLASSO, for comparison. Each of the three models improves the log likelihood of its initialisation, and quickly outperforms GLASSO on test log likelihood. Smaller changes in test log likelihood and sparsity continue over a longer period.

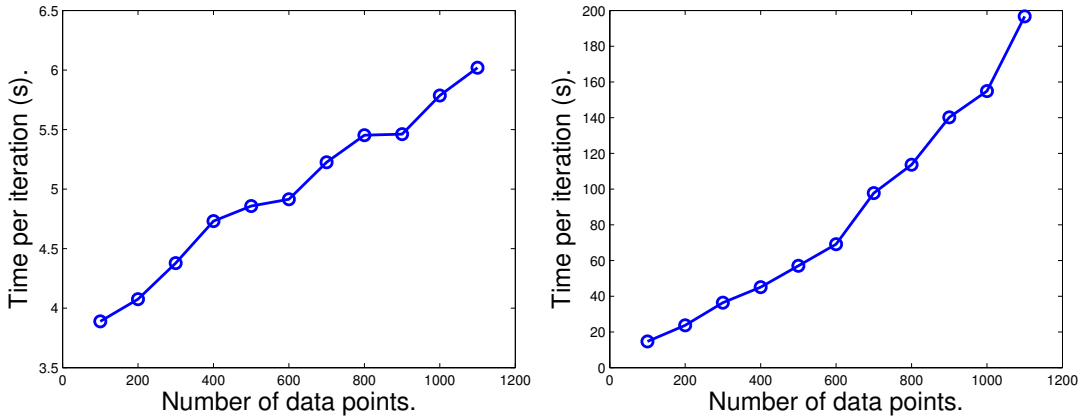


Figure 2.15: Training times per iteration of EM versus number of data points in the training set, when using a Gaussian (left) or a kernel density estimator (right) for the body of the marginal models. Training times are much lower for the Gaussian. Scaling appears approximately linear for the Gaussian, but superlinear for the KDE.

plotted against the number of data points in the training set, are shown in Figure 2.15. Clearly, training of CopMSLICE takes longer with more data. The scaling appears approximately linear with a Gaussian in the marginal body, which may be expected: the dominant computational cost is the evaluation of the marginal cdfs, and these functions must be evaluated on each data point. We used Matlab's `ksdensity` function to evaluate the KDE; its implementation affects the scaling of CopMSLICE's training time, which appears superlinear in the figure. Comparing the two subfigures, it is clear that training times are much lower with the Gaussian instead of the KDE.

2.6 Conclusions and Future Work

In this chapter we introduced SLICE, an EM algorithm for learning a sparse precision matrix of a Gaussian model with latent variables. We demonstrated that, as greater sparsity was enforced, SLICE performed better than the fully visible graphical lasso method in terms of test log likelihood: SLICE was able to find a more parsimonious representation of the data than GLASSO.

We showed how to extend the SLICE EM algorithm to learn a conditional mixture of latent Gaussians. As expected, MSLICE outperformed the single component method. We showed that the mixture components could be interpreted in terms of the side information: in our experiment with financial data, the model learned to use the mixture components to handle different levels of market volatility. The sparse preci-

sion matrices resulting from MSLICE were also interpretable to some extent: the latent variables in the low-volatility expert roughly corresponded to industries, and some residual connections could be interpreted with additional knowledge of the companies involved. This suggests that MSLICE might be used to aid knowledge discovery.

Finally, we augmented SLICE and MSLICE with Gaussianising functions, turning them into a Gaussian copula, CopSLICE, and a mixture of Gaussian copulas, CopMSLICE. We showed that the copula versions performed better than their respective Gaussian versions. For SLICE, we trained the Gaussianising functions in a preprocessing step for SLICE. But for MSLICE, the functions must be learned simultaneously with the rest of the model, so we integrated their training into the MSLICE EM algorithm.

Regarding future work, a simple extension could be to change the distribution of the mixing variable $\boldsymbol{w}$. In this thesis we used a multinomial logit, but this may not be appropriate for certain tasks. It should be straightforward to change this distribution. Alternatively, recall that at present, the number of components in the conditional mixture, and the number of latent variables in each component, are set by cross-validation. Future work could involve an extension in which these parameters are built into the model and learned in a more principled way. For example, the Dirichlet process is used in the construction of an infinite mixture (Antoniak, 1974), from which the number of components responsible for the data can be inferred. MSLICE may benefit from a similar extension.

Another direction in which we could extend our work is to build on sparse matrix-variate normal methods. The matrix-variate normal distribution can model correlations between both the visible variables and the data points at the same time. If $\boldsymbol{Y}$ has a matrix-variate normal distribution, we write $\boldsymbol{Y} \sim \mathcal{MN}\left(\boldsymbol{M}, \boldsymbol{\Psi}^{-1}, \boldsymbol{\Theta}^{-1}\right)$, where $\boldsymbol{M}$ is the mean matrix, and $\boldsymbol{\Psi}$ and $\boldsymbol{\Theta}$ are precision matrices associated with the rows and columns respectively. The definition is such that $vec(\boldsymbol{Y}) \sim \mathcal{N}\left(vec(\boldsymbol{M}), \boldsymbol{\Psi}^{-1} \otimes \boldsymbol{\Theta}^{-1}\right)$, where $\otimes$ denotes the Kronecker product (KP). Zhang and Schneider (2010) apply $L1$ penalties to both $\boldsymbol{\Psi}$ and $\boldsymbol{\Theta}$, and learn them iteratively by fixing one and running graphical lasso to optimise the other. Kalaitzis et al. (2013) note that the KP corresponds to the graph tensor product. They point out that this may result in dense dependencies between the rows and columns of $\boldsymbol{Y}$, and that the graph Cartesian product – which corresponds to the Kronecker sum (KS) – is sometimes a more natural choice of model. So Kalaitzis et al. (2013) propose to replace the KP with the KS in the matrix-variate normal, and they train this model with $L1$ penalties on the elements of $\boldsymbol{\Psi}$ and $\boldsymbol{\Theta}$. They name their

method the bigraphical lasso.

It may be useful to extend the sparse matrix-variate methods in a similar manner to how MSLICE extends GLASSO. The motivation would be to retain the advantages of MSLICE over GLASSO, but with the additional capability of capturing correlations between the data points. If some entries of the data matrix were unobserved (including entire rows or columns), we might use EM, inferring the latent variables in the E step and applying a method such as the bigraphical lasso in the M step. We could then incorporate multiple components and side information in a similar way to MSLICE.

We intend to study the application of the new methods to additional domains and tasks. (Cop)MSLICE may be useful in financial risk management. The task there is to simulate what would happen under extreme market conditions such as a doubling of the price of oil, or a collapse of the housing market. This is problematic because there are likely to be few data points for such scenarios, and there are probably no data points for combinations of extreme events. (Cop)MSLICE may be helpful because it learns a relationship between side information (which could include oil price or a house price index, for example) and the observed variables (such as asset returns). So we could set the side information to some extreme values and sample from the model to predict the outcome. In (Agakov et al., 2012), we published preliminary results suggesting that MSLICE may be used in this way, but much more work needs to be done to determine if it is indeed viable.

2.6.1 Context-Dependent Precisions

In (Cop)MSLICE, the component means and priors depend on the side information. But the side does not directly influence each component's precision matrix: the precisions remain fixed after training, and the covariate values affect only which of the fixed components is responsible for each data point. But in our financial experiment, performance might improve if the structure and the values of the precision elements varied gradually with volatility. We may also wish to incorporate prior knowledge about structural changes. For example, if we know that one company made an offer to buy another during the period of the training data, we may expect those companies' returns to become coupled after the offer – but the dependence structure between the other companies should not change. MSLICE cannot capture these kinds of changes. We have not yet developed a model suitable for these problems. In this section, we briefly review some relevant work from the literature, and describe our preliminary

attempts at solving this problem and the difficulties we encountered.

It should be straightforward to develop simple extensions of MSLICE and CopMSLICE in which the variances of the visible variables depend on the side information, but the correlations remain fixed. In MSLICE, for example, parameterise the precision of expert m given the vector of side information $\boldsymbol{x}_n$ as

$$\boldsymbol{\Lambda}'_m(\boldsymbol{x}_n) \equiv \boldsymbol{D}_m(\boldsymbol{x}_n)\boldsymbol{\Lambda}_m\boldsymbol{D}_m(\boldsymbol{x}_n), \tag{2.75}$$

where $\boldsymbol{D}_m(\boldsymbol{x}_n)$ is a diagonal matrix and $\boldsymbol{\Lambda}_m$ is independent of $\boldsymbol{x}$. Since

$$\log\det\boldsymbol{\Lambda}'_m(\boldsymbol{x}_n) = 2\log\det\boldsymbol{D}_m(\boldsymbol{x}_n) + \log\det\boldsymbol{\Lambda}_m, \tag{2.76}$$

then, with $\boldsymbol{D}_m$ held constant, the objective (2.58) for $\boldsymbol{\Lambda}_m$ in the M step of MSLICE is changed only by an additive constant, and can still be learned by graphical lasso. With the other MSLICE parameters fixed, the parameters of $\boldsymbol{D}_m(\boldsymbol{x}_n)$ could then be optimised. For CopMSLICE, a similar extension might involve making the marginal models a function of the side. The new parameters might be learned in a similar manner to that described in Section 2.4.2.2.

Allowing the partial correlations – and the structure itself – to depend on the side information is more challenging. The graph-optimised classification and regression trees (Go-CART) method (Liu et al., 2010) addresses the problem by constructing a tree on the covariate space. For each leaf of the tree, the training data points whose covariates fall into the corresponding region are used to train a graphical lasso model. The final model is therefore a set of sparse Gaussians. When a prediction is required, the region into which the covariate vector falls determines which Gaussian is used. The tree is composed of hyper-rectangles, which Liu et al. (2010) iteratively split to minimise risk on a held-out data set. This model has similar limitations to MSLICE: the sparse Gaussians are fixed after training, the covariates serving only to select which of them is responsible for each data point. It has the following further limitations caused by the covariate space partitioning.

1. The structure may change abruptly at the hyper-rectangle boundaries.

2. Use of the data may be inefficient. For example, a data point may provide information on the structure for multiple regions of the covariate space, but is only used within one region.

Liu et al. (2010) also discuss briefly the use of kernel smoothing to estimate a graph

that depends on a covariate vector. Let

$$\boldsymbol{\mu}(\boldsymbol{x}) = \frac{\sum_n K(||\boldsymbol{x}-\boldsymbol{x}_n||)\,\boldsymbol{y}_n}{\sum_n K(||\boldsymbol{x}-\boldsymbol{x}_n||)}, \tag{2.77}$$

$$\boldsymbol{S}(\boldsymbol{x}) = \frac{\sum_n K(||\boldsymbol{x}-\boldsymbol{x}_n||)\,(\boldsymbol{y}_n-\boldsymbol{\mu}(\boldsymbol{x}))\,(\boldsymbol{y}_n-\boldsymbol{\mu}(\boldsymbol{x}))^T}{\sum_n K(||\boldsymbol{x}-\boldsymbol{x}_n||)}, \tag{2.78}$$

where K is a kernel function. Then use $\boldsymbol{S}(\boldsymbol{x})$ as input to the graphical lasso to estimate a sparse Gaussian. Thus, the Gaussian's precision – both its values and its graph structure – may be different at each test data point. However, there are drawbacks to this method. Liu et al. (2010) point out that it requires global smoothness of the mean and covariance functions, and that it is challenging to reconstruct the partition of the covariate space corresponding to different graph structures. For our purposes, the method does not easily incorporate certain types of prior information – such as our example above of a company making a bid for another. Finally, kernel smoothing may not scale well to high dimensions: as the volume of the covariate space expands, the amount of data required may become prohibitive.

Cheng et al. (2012) consider a conditional Ising model. Given a covariate vector $\boldsymbol{x} \in \mathbb{R}^C$, a vector of binary variables $\boldsymbol{y} \in \{0,1\}^V$ is assumed to be distributed as

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{\theta}(\boldsymbol{x}))} \exp\left(\sum_{v=1}^{V} \theta_{vv}(\boldsymbol{x}) y_v + \sum_{(u,v):1\le u<v\le V} \theta_{uv}(\boldsymbol{x}) y_u y_v\right), \tag{2.79}$$

where the parameters $\boldsymbol{\theta}(\boldsymbol{x}) \equiv (\theta_{11}(\boldsymbol{x}), \theta_{12}(\boldsymbol{x}), \cdots, \theta_{V-1,V}(\boldsymbol{x}), \theta_{VV}(\boldsymbol{x}))$ are functions of the side information $\boldsymbol{x}$, and $Z(\boldsymbol{\theta}(\boldsymbol{x}))$ is the partition function. They choose a linear model for the parameters: $\theta_{uv}(\boldsymbol{x}) = \theta_{uv0} + \boldsymbol{\theta}_{uv}^T \boldsymbol{x}$, and use $L1$ penalties to sparsify the vector $\boldsymbol{\theta}_{uv}$ (but not the intercept θ_{uv0}). Maximising the joint conditional log likelihood $\sum_{n=1}^N \log P(\boldsymbol{y}_n|\boldsymbol{x}_n)$ is intractable, so instead they maximise $\sum_{n=1}^N \log P\left(y_{nv}|\boldsymbol{x}_n, \boldsymbol{y}_{n\setminus v}\right)$ for each variable v separately (plus penalties). This turns out to be a set of penalised logistic regression problems. Consider a similar extension of MSLICE: let $\Lambda_{ij}(\boldsymbol{x}) = \boldsymbol{\alpha}_{ij}^T \widetilde{\boldsymbol{x}}$ (where we temporarily omit the component index m for clarity). The term $\frac{N}{2}\log\det\boldsymbol{\Lambda}$ in the objective (2.58) now becomes $\frac{1}{2}\sum_n \log\det\boldsymbol{\Lambda}(\boldsymbol{x}_n)$. If a determinant must be evaluated for every point in the data set, that could be computationally expensive. Perhaps a pseudolikelihood approach akin to that of Cheng et al. (2012) would solve this problem. However, we must ensure that $\boldsymbol{\Lambda}(\boldsymbol{x})$ is positive definite – not just during training, but for any value of $\boldsymbol{x}$ – and this model does not guarantee that. Furthermore, we want $\boldsymbol{\Lambda}(\boldsymbol{x})$ to be sparse – and not just for the covariates $\{\boldsymbol{x}_n\}_{n=1}^N$ in the training data, but for any $\boldsymbol{x}$ that may be encountered at test time. This model does not impose such sparsity.

Working with the Cholesky decomposition of the precision addresses two of these issues. Let $\boldsymbol{\Lambda} = \boldsymbol{L}\boldsymbol{L}^T$ where $\boldsymbol{L}$ is a lower-triangular matrix, and parameterise $\boldsymbol{L}$ such that $L_{ij}(\boldsymbol{x}) = \boldsymbol{\alpha}_{ij}^T \widetilde{\boldsymbol{x}}$, for $i \geq j$. In this representation, $\boldsymbol{\Lambda}(\boldsymbol{x})$ is positive definite for all $\boldsymbol{x}$. Furthermore, $\log\det\boldsymbol{\Lambda}(\boldsymbol{x}) = 2\log\det\boldsymbol{L}(\boldsymbol{x})$, and since $\boldsymbol{L}$ is triangular, its determinant is the product of its diagonal entries. So the determinants could be evaluated more quickly in this representation. However, the problem of sparsifying $\boldsymbol{\Lambda}(\boldsymbol{x})$ remains. Another potential problem is that the entries of $\boldsymbol{L}$ are not as easily interpreted as those of $\boldsymbol{\Lambda}$, so it is not clear how to parameterise L_{ij}, or how to interpret the structure of the learned model.

Another approach we looked at borrows ideas from sparse coding. Encoding a signal as a sparse combination of the elements of a dictionary set has been effective in signal processing – see (Chen et al., 1998) and (Mallat, 2008), for example. We considered representing an MSLICE precision matrix as a sparse linear combination with side-dependent coefficients: let $\mathcal{D} \equiv \{\boldsymbol{\Lambda}_d\}_{d=1}^D$ denote a set of sparse precision matrices (the dictionary), and let $\boldsymbol{\Lambda}(\boldsymbol{x}) = \sum_{d=1}^D \alpha_d(\boldsymbol{x})\boldsymbol{\Lambda}_d$, where $\alpha_d(\boldsymbol{x}) \geq 0$, and $\boldsymbol{\alpha}(\boldsymbol{x}) \equiv (\alpha_1(\boldsymbol{x}), \cdots, \alpha_D(\boldsymbol{x}))^T$ is a sparse vector. The set $\mathcal{D}$ would be learned, with $L1$ penalties imposed to sparsify each $\boldsymbol{\Lambda}_d$. The precision $\boldsymbol{\Lambda}(\boldsymbol{x})$ is guaranteed to be positive definite – but compared to the Cholesky representation, the dictionary elements and their combination given some covariate vector $\boldsymbol{x}$ may be more interpretable. Another potential advantage is that a combination of sparse matrices may also be sparse. However, two problems that we encountered above are still present in this model.

1. The log likelihood contains a term $\frac{1}{2}\sum_n \log\det\left[\sum_d \alpha_d(\boldsymbol{x}_n)\boldsymbol{\Lambda}_d\right]$, so it may still be necessary to evaluate a determinant for every training data point at each iteration of the learning algorithm.

2. Imposing sparsity on $\boldsymbol{\alpha}(\boldsymbol{x})$ (for any $\boldsymbol{x}$, not just the covariate vectors in the training set) is not straightforward.

Addressing these problems is left for future work.

Chapter 3

Temporal Sparse Gaussian Models

In Chapter 2, we developed the CopMSLICE model – a conditional mixture of sparse, latent-variable, Gaussian copulas – and we showed how to learn the model by EM. We focussed on the implications of the latent variables, and examined different choices of marginal model. We have so far considered each data point to be independent and identically distributed (iid).

In this chapter, we examine the state variables $\boldsymbol{w}$ more closely. In Section 2.3.2.1, we chose a multinomial logit model for $\boldsymbol{w}$ given a vector of side information $\boldsymbol{x}$. In practice, we may have prior knowledge concerning the latent state. For example, in the experiments of Section 2.5, we modelled stock market returns with the CopMSLICE family of models, and we saw the mixture components each became responsible for a different degree of market volatility. If we think of the latent state more generally as a market regime, then we might expect the state to persist in time: financial market performance often responds to political changes, or changes in various macroeconomic factors, which tend to persist over longer periods than a single day. Therefore, it may be useful to build this notion of state persistence into the model, instead of modelling the data points as iid. So in this chapter, we extend MGLASSO and CopMGLASSO[1] to temporal models. The Gaussian components of the models in this chapter have no latent variables. Incorporating latent variables should be straightforward, but would result in significantly slower training – and the temporal extensions that we introduce already slow down the models' training compared to their static counterparts, as we shall see in Section 3.3.4.

The experiments of Section 2.5 were limited to a single dimension of side infor-

[1] Recall that we name the particular case of MSLICE with no latent variables ($H = 0$) in the constituent Gaussians MGLASSO. Similarly, CopMGLASSO is CopMSLICE with no such latent variables.

mation (the VIX volatility index in that case). In this chapter, we perform experiments with a richer side information with a larger number of features, and examine the extent to which the set of features selected by the sparse learning algorithm are interpretable. We study the interaction between the high-dimensional side information and the new temporal model of latent state evolution.

The models that we develop here are input-output hidden Markov models (IO-HMMs), so we begin with some background material on the IO-HMM. We then describe our new IO-HMM models in Section 3.2, explain how they relate to the models of the previous chapter, and show how their parameters may be learned by EM. Section 3.3 contains the results of experiments that compare the new models with those of Chapter 2; investigate how useful are the temporal aspects of the models; and examine the interpretability of the inputs selected by the learning algorithm. We look at the computational costs of training the models in Section 3.3.4. Finally, we conclude the chapter and discuss future work.

3.1 Background: The Input-Output HMM

Let $\boldsymbol{x}_n \in \mathbb{R}^C$ denote a vector of covariates (the inputs) and $\widetilde{\boldsymbol{y}}_n \in \mathbb{R}^V$ denote a vector of visible variables (the outputs) at time n. Let $\boldsymbol{w}_n \in \{0,1\}^M$ indicate which of M components is responsible for generating $\widetilde{\boldsymbol{y}}_n$; that is, $w_{nm} = 1$ for some m, and $w_{nl} = 0$ for all $l \neq m$. Vectors $\boldsymbol{x}_n$, $\widetilde{\boldsymbol{y}}_n$, and $\boldsymbol{w}_n$ exist for $1 \leq n \leq N$, while for mathematical convenience we introduce an initial state $\boldsymbol{w}_0$. A graphical model for the input-output hidden Markov model (IO-HMM) (Bengio and Frasconi, 1995) is shown in Figure 3.1. To define a concrete instance of the IO-HMM model, one must make choices for:

1. The initial state model $p(\boldsymbol{w}_0;\boldsymbol{\pi})$;
2. The transition model $p(\boldsymbol{w}_n|\boldsymbol{w}_{n-1},\boldsymbol{x}_n;\boldsymbol{\Xi})$; and
3. The emission model $p(\widetilde{\boldsymbol{y}}_n|\boldsymbol{w}_n,\boldsymbol{x}_n;\boldsymbol{\Upsilon})$.

The transition and emission models do not change with time n. Note that if we choose $p(\boldsymbol{w}_n|\boldsymbol{w}_{n-1},\boldsymbol{x}_n;\boldsymbol{\Xi}) = p(\boldsymbol{w}_n|\boldsymbol{x}_n;\boldsymbol{\Xi})$, all the observations become independent and the IO-HMM reduces to a mixture of experts.

The parameters of the IO-HMM may be learned by EM. The E step involves inferring expectations of the latent states, and can be performed efficiently by the forward-backward algorithm, which makes use of the chain structure of the graphical model.

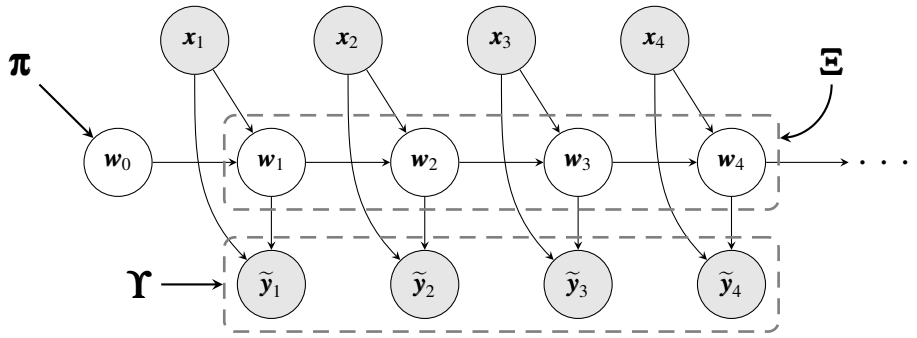


Figure 3.1: Graphical model of an input-output HMM, where $\boldsymbol{x}_n$ is the input, $\boldsymbol{w}_n$ the discrete latent state, and $\widetilde{\boldsymbol{y}}_n$ the output at time n. Here, $\boldsymbol{\pi}$ parameterises the initial state distribution, $\boldsymbol{\Xi}$ parameterises the transition model, and $\boldsymbol{\Upsilon}$ parameterises the emission model.

For an explanation of the forward-backward algorithm in the HMM, see, for example, (Bishop, 2006). The extension to the IO-HMM is straightforward. Here, we simply state the key steps of the algorithm.

The following two expectations are required for use in the M step:

$$\rho(n,m) \equiv \mathbb{E}\left[w_{nm}|\boldsymbol{X},\widetilde{\boldsymbol{Y}},\boldsymbol{\theta}^{(t-1)}\right], \tag{3.1}$$

$$\omega(n,l,m) \equiv \mathbb{E}\left[w_{n-1,l}w_{nm}|\boldsymbol{X},\widetilde{\boldsymbol{Y}},\boldsymbol{\theta}^{(t-1)}\right], \tag{3.2}$$

where t is the EM iteration index. Since these are expectations of binary variables,

$$\rho(n,m) = p\left(w_{nm}=1|\boldsymbol{X},\widetilde{\boldsymbol{Y}},\boldsymbol{\theta}^{(t-1)}\right), \tag{3.3}$$

$$\omega(n,l,m) = p\left(w_{n-1,l}=1,w_{nm}=1|\boldsymbol{X},\widetilde{\boldsymbol{Y}},\boldsymbol{\theta}^{(t-1)}\right). \tag{3.4}$$

In the following, everything is conditioned on $\boldsymbol{X}$ and $\boldsymbol{\theta}^{(t-1)}$, so we now omit these for clarity. To compute ρ and ω, first define the functions α and β as follows:[2]

$$\alpha(n,m) = \frac{p\left(\widetilde{\boldsymbol{y}}_1,\cdots,\widetilde{\boldsymbol{y}}_n,w_{nm}=1\right)}{p\left(\widetilde{\boldsymbol{y}}_1,\cdots,\widetilde{\boldsymbol{y}}_n\right)}, \tag{3.5}$$

$$\beta(n,m) = \frac{p\left(\widetilde{\boldsymbol{y}}_{n+1},\cdots,\widetilde{\boldsymbol{y}}_N|w_{nm}=1\right)}{p\left(\widetilde{\boldsymbol{y}}_{n+1},\cdots,\widetilde{\boldsymbol{y}}_N|\widetilde{\boldsymbol{y}}_1,\cdots,\widetilde{\boldsymbol{y}}_n\right)}. \tag{3.6}$$

For each n, we can compute $\alpha(n,m)$ using the following recursive relationship:

$$\alpha(n,m) = \frac{p\left(\widetilde{\boldsymbol{y}}_n|w_{nm}=1\right)\sum_l \alpha(n-1,l)p\left(w_{nm}=1|w_{n-1,l}=1\right)}{p\left(\widetilde{\boldsymbol{y}}_n|\widetilde{\boldsymbol{y}}_1,\cdots,\widetilde{\boldsymbol{y}}_{n-1}\right)}, \tag{3.7}$$

[2] The functions α and β may be defined without the denominators in Equations (3.5) and (3.6). The denominators are for scaling purposes, to avoid problems caused by finite machine precision.

beginning with $\alpha(0,m) = p(w_{0m} = 1)$. The two probabilities in the numerator are defined by the transition and emission models. The denominator can be found by normalisation, because $\sum_m \alpha(n,m) = 1$. Similarly, $\beta(n,m)$ can be computed via this recursive relationship:

$$\beta(n,m) = \frac{\sum_l \beta(n+1,l) p\left(\widetilde{\boldsymbol{y}}_{n+1} | w_{n+1,l} = 1\right) p\left(w_{n+1,l} = 1 | w_{nm} = 1\right)}{p\left(\widetilde{\boldsymbol{y}}_{n+1} | \widetilde{\boldsymbol{y}}_1, \cdots, \widetilde{\boldsymbol{y}}_n\right)}. \tag{3.8}$$

The values of $\beta(n,m)$ are computed in reverse order (decreasing n) beginning with $\beta(N,m) = 1$ for each m. The probabilities (3.1) and (3.2) can be computed in terms of α and β as follows:

$$\rho(n,m) = \alpha(n,m)\beta(n,m), \tag{3.9}$$

$$\omega(n,l,m) = \frac{\alpha(n-1,l) p\left(\widetilde{\boldsymbol{y}}_n | w_{nm} = 1\right) p\left(w_{nm} = 1 | w_{n-1,l} = 1\right) \beta(n,m)}{p\left(\widetilde{\boldsymbol{y}}_n | \widetilde{\boldsymbol{y}}_1, \cdots, \widetilde{\boldsymbol{y}}_{n-1}\right)}. \tag{3.10}$$

The M step of the EM algorithm is a sequence of optimisation problems for the model parameters, the form of which depends on the choice of transmission and emission models.

3.2 A Sparse Input-Output HMM

If a flexible representation is chosen for the emission model such that the number of parameters in $\boldsymbol{\Upsilon}$ is large compared to the number of data points N, there is a danger of overfitting. For the commonly chosen Gaussian emission model, a standard way to deal with this is to use factorised emissions. That is, if the emission model for state m is $\mathcal{N}\left(\widetilde{\boldsymbol{y}} | \boldsymbol{\mu}_m(\boldsymbol{x}), \boldsymbol{\Lambda}_m^{-1}\right)$, then $\boldsymbol{\Lambda}_m$ is constrained to be diagonal. See, for example, (Bengio et al., 2001; Ernst et al., 2007). Unfortunately, this choice can result in over-restrictive representations of the emissions. It would be more attractive to constrain them to have more flexible, but explainable dependency structures: finding the right balance between parsimony and flexibility of dependencies is particularly important in finance, where it is recognized as one of the crucial problems (Bauwens et al., 2006).

In this Section, we develop a sparse IO-HMM, which may be viewed as arising from either of the following perspectives:

1. The desire to put prior information about state persistence into the CopMSLICE model family; or

2. The need for more flexible and parsimonious emissions than in the standard application of IO-HMMs.

Our emission model will be either a multivariate Gaussian or, more generally, a Gaussian copula. For the latter, given $w_{nm} = 1$, the model is:

$$\boldsymbol{y}_n \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}_m^{-1}\right), \tag{3.11}$$

$$\widetilde{y}_{nv} = f_{mv}^{-1}\left(y_{nv}\right) + \left(\boldsymbol{\Psi}_m^T\right)_{v:} \widetilde{\boldsymbol{x}}_n. \tag{3.12}$$

The notation is the same as used in Chapter 2. That is, $f_{mv} \equiv \Phi^{-1} \circ F_{mv}$; Φ is the univariate Gaussian cdf; $\widetilde{\boldsymbol{x}}_n$ augments $\boldsymbol{x}_n$ with a unit element; and $\boldsymbol{\Psi}_m$ parameterises the conditional mean. If each f_{mv} is the identity map, the emissions are multivariate Gaussian, and we refer to this IO-HMM as TGLASSO. For the more general non-Gaussian emissions, we refer to the IO-HMM as CopTGLASSO. As for the Chapter 2 models, we apply $L1$ penalties to sparsify the precision matrices $\boldsymbol{\Lambda}_m$ and the mean parameters $\boldsymbol{\Psi}_m$.

Many choices of transition model are possible. In this work, we use a set of multinomial logit models, one for each possible previous state:

$$p\left(w_{nm} = 1 | w_{n-1,l} = 1, \boldsymbol{x}_n; \boldsymbol{\Xi}\right) \propto \exp\left(-\left(\boldsymbol{\Xi}_l^T\right)_{m:} \widetilde{\boldsymbol{x}}_n\right), \tag{3.13}$$

where $(\boldsymbol{\Xi}_l)$ parameterises the multinomial logit given that state l was the previous state. If the dimensionality of input $\boldsymbol{x}_n$ is large, overfitting may be a potential problem. Also, different features of the input may be predictive of different state transitions, and we may wish to learn this. We therefore apply $L1$ penalties to the parameters $\boldsymbol{\Xi} \equiv \{\boldsymbol{\Xi}_l\}_{l=1}^M$. Notice that, if the transitions are independent of the previous state so that

$$p\left(w_{nm} = 1 | w_{n-1,l} = 1, \boldsymbol{x}_n; \boldsymbol{\Xi}\right) = p\left(w_{nm} = 1 | \boldsymbol{x}_n; \boldsymbol{\Xi}\right) \propto \exp\left(-\boldsymbol{\Xi}_{:m}^T \widetilde{\boldsymbol{x}}_n\right), \tag{3.14}$$

then CopTGLASSO reduces to CopMGLASSO (CopMSLICE without latent variables in the Gaussian components).

For the initial state model, we use a categorical distribution. The parameter $\boldsymbol{\pi}$ is an $M \times 1$ vector of state probabilities.

We do not include latent variables in the emission Gaussians. Including them presents no technical difficulty. However, we saw in Section 2.5.5 that SLICE runs much more slowly than GLASSO. As we shall see in Section 3.3.4, the IO-HMMs run more slowly than their non-temporal counterparts, so we choose not to further increase run times by including latent variables here.

3.2.1 An EM Algorithm for CopTGLASSO

Again, we use EM to learn the parameters of CopTGLASSO. Training proceeds in much the same way as CopMGLASSO. The E step consists of running the forward-backward algorithm to compute ρ and ω – see Equations (3.1) and (3.2). For the M step, the full objective function is

$$\begin{aligned} Q\left(\boldsymbol{\theta};\boldsymbol{\theta}^{(t-1)}\right) \equiv & \sum_{m} \rho(0,m) \log p\left(w_{0m}=1;\boldsymbol{\pi}\right) \\ & + \sum_{n=1}^{N} \sum_{m} \rho(n,m) \log p\left(\widetilde{\boldsymbol{y}}_n | w_{nm}=1, \boldsymbol{x}_n; \boldsymbol{\Upsilon}\right) \\ & + \sum_{n=1}^{N} \sum_{l,m} \omega(n,l,m) \log p\left(w_{nm}=1 | w_{n-1,l}=1, \boldsymbol{x}_n; \boldsymbol{\Xi}\right) \\ & - \gamma(\boldsymbol{\theta}) . \end{aligned} \tag{3.15}$$

The optimisations for $\boldsymbol{\pi}$, $\boldsymbol{\Upsilon}$, and $\boldsymbol{\Xi}$ are independent. With our choice of the categorical distribution for the initial state, $\boldsymbol{\pi}$ is optimised simply by setting $\pi_m = \rho(0,m)$. Optimisation of the emission model parameters $\boldsymbol{\Upsilon} = \left(\boldsymbol{\Lambda}, \boldsymbol{\theta}^f\right)$ – the second term in Equation (3.15) – is the same as for CopMGLASSO: see Sections 2.4.2.1 and 2.4.2.2. For the transition parameters $\boldsymbol{\Xi}$, it is straightforward to show that the third term in Equation (3.15) results in the following objective function:

$$\begin{aligned} Q_{\Xi}\left(\boldsymbol{\Xi};\boldsymbol{\theta}^{(t-1)}\right) \equiv & \sum_{l} \sum_{n=1}^{N} \rho(n-1,l) \sum_{m} \eta(n,l,m) \log p\left(w_{nm}=1 | w_{n-1,l}=1, \boldsymbol{x}_n; \boldsymbol{\Xi}\right) \\ & - \gamma_{\Xi}\left(\boldsymbol{\Xi}\right) \end{aligned} \tag{3.16}$$

where

$$\eta(n,l,m) \equiv \frac{p\left(\widetilde{\boldsymbol{y}}_n | w_{nm}=1\right) p\left(w_{nm}=1 | w_{n-1,l}=1\right) \beta(n,m)}{p\left(\widetilde{\boldsymbol{y}}_n | \widetilde{\boldsymbol{y}}_1, \cdots, \widetilde{\boldsymbol{y}}_{n-1}\right) \beta(n-1,l)}, \tag{3.17}$$

and $\gamma_{\Xi}\left(\boldsymbol{\Xi}\right)$ contains the $L1$ penalties. With our choice of transition model, each term in the sum over l in (3.16) – together with the corresponding penalties from the $\gamma_{\Xi}\left(\boldsymbol{\Xi}\right)$ term – is the objective of an independent sparse multinomial logit problem with data point n weighted by $\rho(n-1,l)$. We define Q_{Ξ_l} such that

$$Q_{\Xi}\left(\boldsymbol{\Xi};\boldsymbol{\theta}^{(t-1)}\right) \equiv \sum_{l} Q_{\Xi_l}\left(\boldsymbol{\Xi}_l;\boldsymbol{\theta}^{(t-1)}\right). \tag{3.18}$$

We again use the L1General package to solve these optimisation problems.

The EM procedure for CopTGLASSO is summarised in Algorithm 4.

Algorithm 4 EM for CopTGLASSO

Initialise $\boldsymbol{\pi}^{(0)}$

for $m \leftarrow 1 : M$ **do**

Initialise $\boldsymbol{\Lambda}_m^{(0)}$ such that $\boldsymbol{\Lambda}_m^{(0)} \succ 0$ and $\operatorname{diag}\left(\left(\boldsymbol{\Lambda}_m^{(0)}\right)_{zz}\right) = \mathbf{1}$

Initialise $\boldsymbol{\Xi}_m^{(0)}$, $\boldsymbol{\Psi}_m^{(0)}$ and $\boldsymbol{\zeta}_m^{(0)}$

end for

for $t \leftarrow 1 : T$ **do**

E Step

for $m \leftarrow 1 : M$ **do**

$\alpha(0,m) \leftarrow \pi_m^{(t-1)}$

$\beta(N,m) \leftarrow 1$

end for

for $n \leftarrow 1 : N$ **do**

Compute $\alpha(n,:)$ via recursion (3.7)

end for

for $n \leftarrow N-1 : 0$ **do**

Compute $\beta(n,:)$ via recursion (3.8)

end for

end E Step

M Step

for $m \leftarrow 1 : M$ **do**

$\pi_m^{(t)} \leftarrow \rho(0,m)$

$\boldsymbol{\Xi}_m^{(t)} \leftarrow \arg\max_{\boldsymbol{\Xi}_m} Q_{\Xi_m}\left(\boldsymbol{\Xi}_m; \boldsymbol{\theta}^{(t-1)}\right)$ ▷ See Equations (3.16 - 3.18)

$\boldsymbol{\Psi}_m^{(t)} \leftarrow \boldsymbol{\Psi}_m^{(t-1)}$; $\boldsymbol{\zeta}_m^{(t)} \leftarrow \boldsymbol{\zeta}_m^{(t-1)}$; $\boldsymbol{\Lambda}_m^{(t)} \leftarrow \boldsymbol{\Lambda}_m^{(t-1)}$

$\left(\boldsymbol{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}\right) \leftarrow \arg\max_{(\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m)} Q_{\theta_m^f}\left(\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m | \boldsymbol{\Lambda}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$

▷ See Equation (2.73)†

$\boldsymbol{\Lambda}_m^{(t)} \leftarrow \underset{\boldsymbol{\Lambda}_m \,:\, \boldsymbol{\Lambda}_m \succeq 0,\ \operatorname{diag}\left((\boldsymbol{\Lambda}_m)_{zz}\right) = \mathbf{1}}{\arg\max} Q_{\Lambda_m}\left(\boldsymbol{\Lambda}_m | \boldsymbol{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}; \boldsymbol{\theta}^{(t-1)}\right)$

▷ See Equation (2.72)†

end for

end M Step

end for

† In the notation of this chapter, $\overline{w}_{nm}$ in Equations (2.72) and (2.73) becomes $\rho(n,m)$.

3.3 Experiments

In this section, we compare the temporal models TGLASSO and CopTGLASSO with the stationary model MGLASSO and two existing models. The first is an IO-HMM with factorised Gaussian emissions, which we refer to as TFAC. This is a special case of TGLASSO in which the transition and mean parameters are unpenalised, and the penalties on the off-diagonal elements of the precision matrices approach infinity (enforcing diagonal precisions). The other model we compare with is that trained by the MRCE method of Rothman et al. (2010), which is a sparse linear regression with sparse Gaussian noise. The MRCE model is a special case of MGLASSO with a single component ($M = 1$). We also compare with a copula version of MRCE, which we term CopMRCE. As the name suggests, the CopMRCE model is the same as CopMGLASSO with $M = 1$. For each copula model, we use the piecewise distribution described in Section 2.4.1.1 for the marginals, with a Gaussian for the body and GPDs for the tails.

We examine the predictive accuracy of the learned models, and interpretation of their precision structures – in the same way we did in Chapter 2. Additionally, our data here comprise a high-dimensional vector of side information, and we examine the selected features to see if we can interpret them. We study the effects of the temporal connections in the model, and their relationship to the side information.

3.3.1 The Data

The data set used here is of a similar kind to the FTSE returns data described in Section 2.5.1. We collected price data from Yahoo Finance for 42 of the largest constituents of the S&P 500 index by market capitalisation, from 1999 to 2011. For the input features, we use the S&P 500, Dow Jones, FTSE 100, and Nikkei 225 market indices, the VIX volatility index, and a range of econometric technical indicators from the TTR package[3] computed on the S&P 500 index. A full list of the assets and technical indicators in this data set is given in Appendix C. The input features for each day are computed using market index values from the previous day – that is, using the most recently available values on each day.

We discarded any dates on which data were missing, and converted the price data and index data to asset returns by dividing each day's price by its value on the previous

[3] The TTR package can be found at
`http://cran.r-project.org/web/packages/TTR/index.html`.

day. We aggregated the returns (but not the input features) into weekly returns, computed daily. That is, on each date in our data set, the output data are the returns over the preceding seven-day period. No information is lost in moving from daily to weekly returns. However, a change in data representation may affect model performance. Aggregating multiple days' returns smooths over the effects of daily fluctuations in price, reducing the severity of outliers. Since Gaussian-based models can be sensitive to outliers, this preprocessing step may improve the performance of the models considered in this chapter.

In the next section, we consider a single-window experiment in which we train a single model. We shall see that the results motivate the experiment in the following section – a multi-window experiment in which we train multiple models on overlapping subsequences of the data set.

3.3.2 Single-Window Experiment

We split the data into a training and a testing sequence. The end of the training sequence (and start of the test sequence) was on 26^{th} January 2009. The training sequence comprises 2500 time steps, and the test sequence has 667 time steps. We further preprocess the aggregated returns by making the training sequence zero-mean, and scaling such that the diagonal of the precision contains all ones.

For the three non-temporal models (MRCE, CopMRCE, and MGLASSO), we simply consider the training sequence to be a set of iid data points. We initialise the three temporal models (TFAC, TGLASSO, and CopTGLASSO) by first training a discriminative mixture (with factored emissions for the TFAC model); in particular, MGLASSO is used to initialise TGLASSO and CopTGLASSO. We initialise the discriminative mixtures by k-means, with the component precisions initialised to the identity. For CopTGLASSO, the marginals F_{mv} are initialised by fitting to the training data for variable v, with each data point weighted according to the responsibility of component m in the initialising mixture. All parameter choices – such as the $L1$ penalties and number of latent states – are made by cross-validation. Note that, in contrast to the previous chapter, we do penalise the parameters of the transition model ($\boldsymbol{\Xi}$) and the emission component means ($\boldsymbol{\Psi}$).

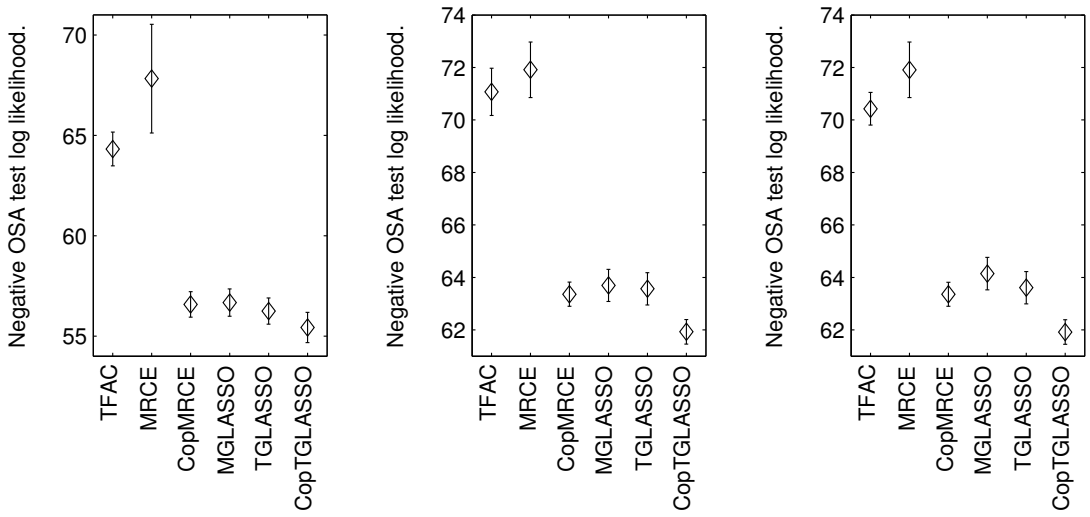


Figure 3.2: Mean OSA negative log likelihoods (lower is better) on test data for (Left) the single window experiment; (Middle) the multi-window experiment; and (Right) the multi-window experiment with no inputs. (Note that the data in the single and multiple window experiments are different, so the likelihood values are not comparable, only the relative performance of the models). The vertical bars depict standard errors across time steps. We note the following. (1) TFAC and MRCE perform poorly compared to the newer models. (2) CopTGLASSO is the best performer. (3) MGLASSO and TGLASSO perform similarly, but (4) the MGLASSO performance drops a little when inputs are removed, but the TGLASSO performance does not.

3.3.2.1 Predictive Accuracy

We evaluated the models by computing the mean of the one-step-ahead (OSA) log likelihoods at each point in the test sequence. That is, at each data point we condition on the data prior to that, and compute the log likelihood of the next point in the sequence. Figure 3.2 (Left) plots the results. A considerable performance gain is obtained over the sparse regression model (MRCE) by including non-Gaussian noise (CopMRCE) or multiple mixture components (MGLASSO). TGLASSO and CopTGLASSO each perform much better than the factorised input-output HMM (TFAC), so the more flexible sparse emission models are beneficial in the IO-HMMs. However, perhaps the most interesting result is the small difference in performance between MGLASSO and TGLASSO: the error bars in their OSA log likelihoods have considerable overlap, so it appears that the introduction of temporal connections provides minimal improvement in predictive accuracy in this experiment. The reason for this may be that the inputs provide enough information about the latent state to make the temporal connections

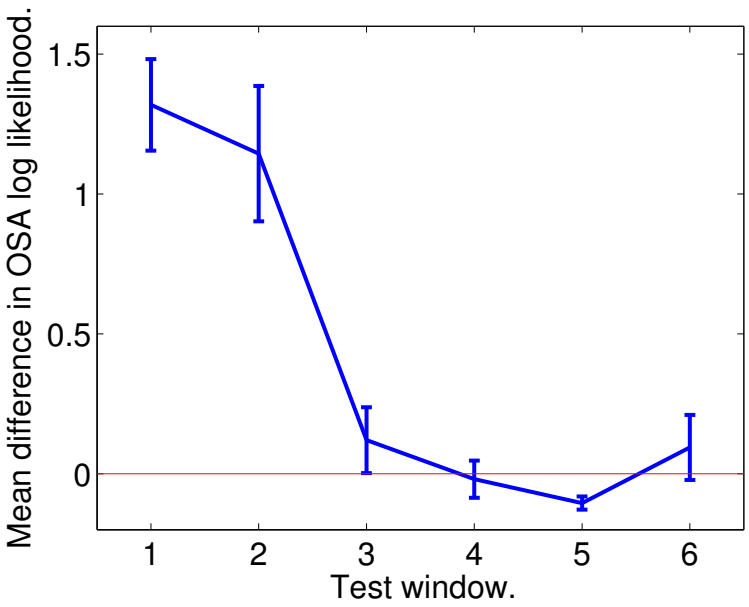


Figure 3.3: For TGLASSO and MGLASSO, we computed the one-step-ahead log likelihoods on the test set. That is, for each data point in the test set, we computed its log likelihood conditioned on all prior data. We then divided the test set into six consecutive windows, and for each model, took the mean of the OSA log likelihoods within each window. This figure plots the difference between the mean OSA log likelihoods for TGLASSO and MGLASSO. TGLASSO significantly outperforms MGLASSO over the first two windows. After that, the models perform similarly.

redundant. We investigate further in Section 3.3.3.1.

We noticed by inspecting the OSA log likelihoods that the temporal models tend to perform better than MGLASSO on the test data immediately following the training set. To illustrate this, we divide the test set into six consecutive, equally-sized, windows. For TGLASSO, we take the OSA log likelihoods computed above, and record the mean value in each window. We do the same for MGLASSO. In Figure 3.3, we plot the difference between the TGLASSO and MGLASSO means in each of the six windows. The plot shows that TGLASSO outperforms MGLASSO in the block immediately following the training data, but this superiority is lost over time. This observation motivates our second experiment, in which we train multiple models on temporal windows of data. A further motivation is that the multi-window scenario is more realistic: in practice, a financial model is likely to be used to predict a few time steps ahead, before retraining as new data are acquired.

Table 3.1: Bias weights learned for the TGLASSO transition model. For each previous state, the next state is modelled as a multinomial logit. The values shown here are the intercept parameters. The diagonal is positive and other values are negative, indicating that the latent state tends to persist in time – unless the side information is strong enough to overcomes the bias.

	Next State		
Previous State	1	2	3
1	2.1913	-1.1407	-1.0506
2	-2.9042	3.2156	-0.3114
3	-2.1607	-0.4496	2.6103

3.3.2.2 Interpretation of Latent States

Inspection of the bias parameters of the transition models for TGLASSO and CopT-GLASSO indicate a strong tendency for the latent state to persist between adjacent time steps. The biases learned for TGLASSO are shown in Table 3.1 (for a typical training run). To compare state persistence between MGLASSO and TGLASSO, we compute for each model the responsibility of each state at each time step, and compute the difference in these responsibilities between adjacent time steps. The sum of the absolute differences for MGLASSO is 407, and for TGLASSO it is 266. So, as expected, the latent state changes less frequently for TGLASSO in this experiment.

The latent states can be interpreted by examining the responsibilities. Figure 3.4 (Left) shows that state 1 of TGLASSO is used to model the period around the market crash of late 2008, while states 2 and 3 correspond approximately to periods of rising trend and falling trend respectively. Notice that the bias weights in Table 3.1 indicate that transitions to state 1 are less common than transitions to the other states – as expected if state 1 is interpreted as an extreme scenario. Figure 3.4 (Right) illustrates the precision matrices $\mathbf{\Lambda}_m$ learned for states 2 and 3, with the stocks sorted according to market sectors. The lower triangle corresponds to state 2 (rising market) and the upper triangle to state 3 (falling market). The structures are similar to those seen in Section 2.5.3: stocks within the same market sector tend to be more strongly coupled, and dependencies are stronger when the market is falling.

Similar patterns of responsibilities and structures were observed for MGLASSO

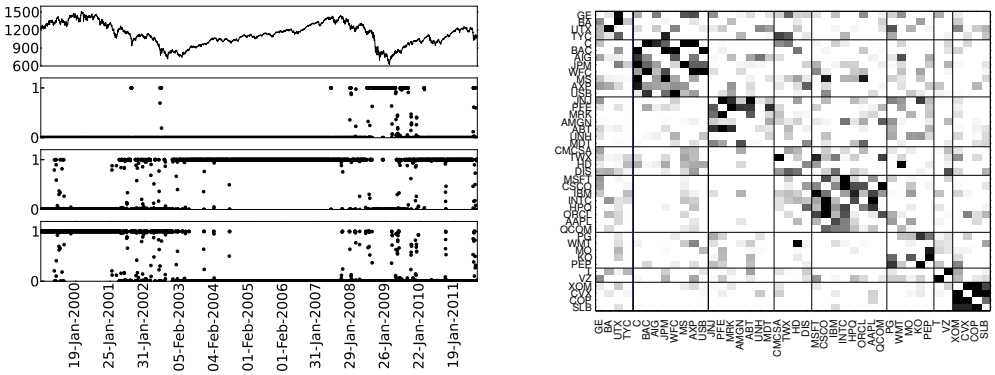


Figure 3.4: (Left) *Top row*: Value of the S&P 500 index. *Rows 2,3,4*: Posterior probability in the TGLASSO model of hidden states 1,2, and 3 respectively, given the observed data. (Right) Visualisation of learned precision matrices in TGLASSO. The lower left and upper right triangles correspond to the rising and falling states respectively. Absolute values are plotted, and the dynamic range is chosen to highlight differences between the two states. Note the greater degree of coupling in the upper right (when the market is falling). Lines group stocks by market sector; it is clear that assets in the same sector tend to be coupled more strongly.

and CopTGLASSO.

3.3.3 Multi-Window Experiment

Motivated by practical problems (in finance, for example) and the result illustrated in Figure 3.3, we repeat the previous experiment but on multiple windows: for each model, we retrain on a series of windows over the data set. Each training window consists of a four-year sequence, with the immediately following three months used as the test sequence. Subsequent windows begin three months after the start of the previous window (and so the windows overlap). We compute OSA log likelihoods at each time step in each test window. The models are compared in Figure 3.2 (Middle). The results are broadly similar to the single-window case – Figure 3.2 (Left) – but CopTGLASSO is now the best performer by a significant margin.

3.3.3.1 The Effects of the Input Features

TGLASSO and MGLASSO perform similarly both here and in the single-window test. Examining the $\mathbf{\Psi}$ parameters, we find that for our choice of input features, most entries

of these matrices were set to zero during training for both MGLASSO and TGLASSO. So the inputs appear not to be predictive of the component means. Examining the parameters $\boldsymbol{\Xi}$, we notice that MGLASSO uses more of the input features than TGLASSO. So we remove the inputs and repeat the experiment to see how that affects each model. (Removing the inputs means that TGLASSO becomes a sparse HMM – as opposed to IO-HMM – and MGLASSO becomes a sparse mixture model.) Figure 3.2 (Right) shows the results. TGLASSO and CopTGLASSO perform much the same as when the inputs are present, but MGLASSO's performance is worse because it is unable to model the non-stationarity. The results suggest that MGLASSO is more sensitive to the presence of good input features: as the inputs become sufficiently predictive of the hidden state, MGLASSO's performance improves. But since (Cop)TGLASSO can model the temporal dependence between states, its performance is less sensitive to the choice of inputs.

TFAC actually performs better when there are no inputs, which we expect is due to overfitting in this unregularised model. The performance of MRCE and CopMRCE is unchanged when the inputs are removed: other models can use the inputs to predict the hidden state, but (Cop)MRCE is a single-state model.

3.3.3.2 Interpretation of the Selected Features

We now inspect the input features selected by MGLASSO to see if we can interpret them. Each $\boldsymbol{\Xi}_m$ is a matrix in which the rows correspond to input features. For each feature, we sum the magnitudes of the values in each row of each $\boldsymbol{\Xi}_m$ as a measure of the predictive strength of the input features. Table 3.2 lists the results, with features ranked by this measure of predictive strength. Among the most relevant features, we find the average true range (ATR), a volatility indicator (Vol), Chaikin volatility (ChaikinVol), Bollinger band bounds (BBlo and BBhi), and the implied volatility (VIX). These are all measures of market volatility. It makes sense that these would score high: as volatility is the standard criterion for measuring risk, it is natural to expect that volatility-related features will be predictive of the market state (Cuthbertson, 1996). Another group of technical indicators that rank high on the list contains measures of market momentum: that is, to what extent the market is rising or falling – the so-called "bull" and "bear" markets. This group includes the Chaikin accumulation / distribution line (ChaikinAD), the detrended price oscillator (DPO), the Williams AD line (WilliamsAD), and the triple smoothed exponential oscillator (TRIX).

Features that are less useful in predicting the hidden state include indicators that

Table 3.2: Input features ranked according to predictive strength, measured by the sum of magnitudes of the associated transition-model parameters. Technical indicators measuring market volatility or momentum are strongly predictive of the market state. Less useful in this experiment are indicators that average over very long or short periods, and those that attempt to predict the onset of a trend.

Rank	**Technical Indicator**	**Predictive Strength**
1	ATR	38.7404
2	ChaikinAD	26.3262
3	BBlo	18.9525
4	DPO	11.9678
5	Vol	11.0775
6	WilliamsAD	10.2598
7	TRIX	9.6246
8	ChaikinVol	9.2545
9	BBhi	8.0977
10	VIX	6.1606
11	AroonDn	4.9219
12	DX	4.4171
13	AroonUp	3.6304
14	KST	3.4232
15	MFI	3.3311
16	VHF	3.2905
17	DVI	3.2771
18	OBV	3.1303
19	ChaikinMF	2.9679
20	ADX	2.3636
21	TDI	2.0067
22	MACD	1.4260
23	CCI	0.8702
24	NIKKEI	0.4893
25	EMV	0.2866
26	DOW	0.2537
27	BBpct	0.1451
28	SP500	0.0945
29	FTSE	0.0635
30	CLV	0.0198

average over a very long period – such as the DV intermediate oscillator (DVI) – or consider too short a period – such as the close location value (CLV). Indicators that try to predict the start and end of trends are also less useful, such as the commodity channel index (CCI), trend detection index (TDI), and the vertical horizontal filter (VHF). This is intuitive, as we would not expect very long or very short-term indicators to be useful for predicting weekly returns.

3.3.4 Computational Costs

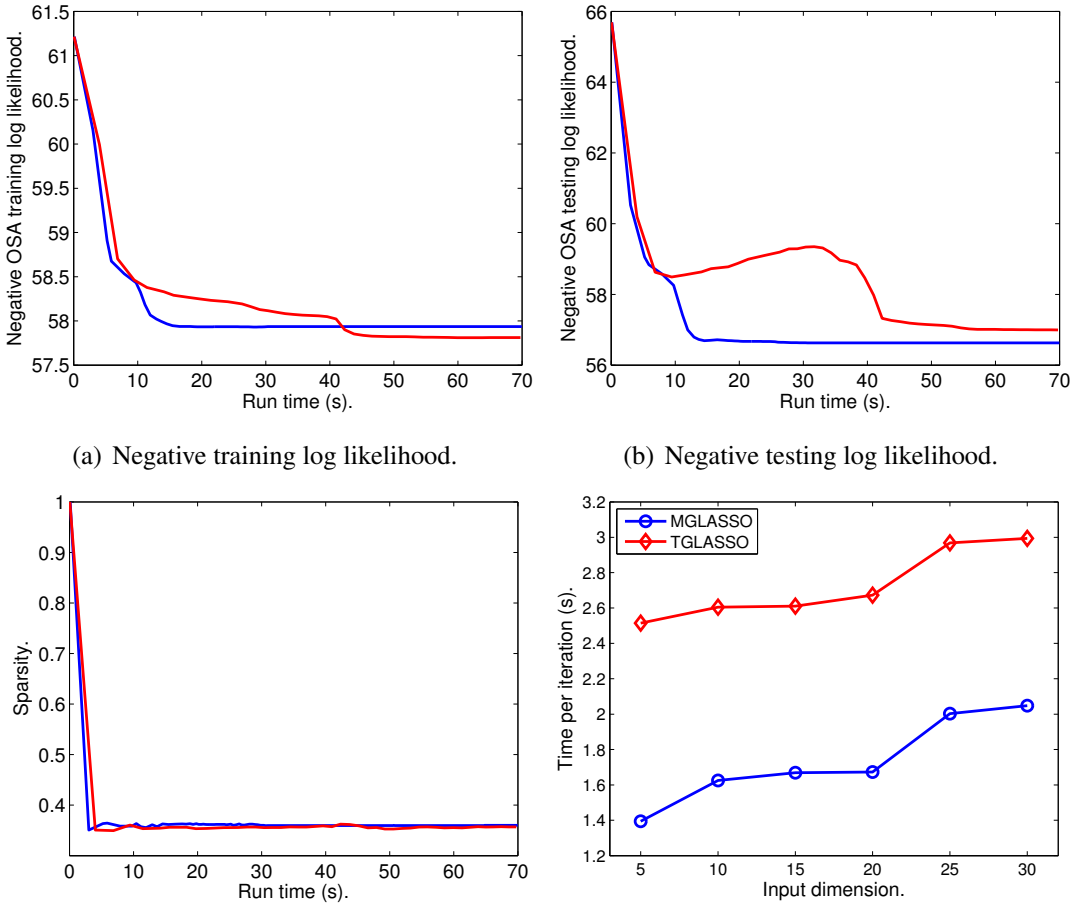
The time complexity of an IO-HMM is linear in N (the length of the training sequence[4]), and quadratic in M (Bengio and Frasconi, 1996). Thus, MGLASSO and TGLASSO both have computational cost linear in N, but MGLASSO is linear in M while TGLASSO is quadratic in M. With our choice of mixing model for MGLASSO and transition model for TGLASSO, both have an E step that is linear in C. The M steps of these models are the same; computational costs of MGLASSO and related non-temporal models are discussed in Section 2.5.5.

We compare empirically the convergence of MGLASSO and TGLASSO, using a similar methodology to Section 2.5.5.1. We run each model on the single-window data set used in Section 3.3.2, and record the one-step-ahead log likelihood of the training and testing data after each iteration of training. We also measure the sparsity[5] of each component's precision matrix in the emission model, and record the mean at each iteration. Figure 3.5(a-c) shows the results. Initially, the negative log likelihoods fall at the same rate for both models. However, log likelihoods for MGLASSO stabilise more quickly than for TGLASSO. The former appears to have converged after about 20 seconds (or 20 iterations), while TGLASSO appears to take around 50 seconds (or 30 iterations). The mean sparsity of each model's emissions drops quickly in the first few seconds, then changes little after that.

Given our observations in Section 3.3.3.1 on the sensitivity of MGLASSO and TGLASSO to the inputs, it is important to examine how training times vary with the dimension of the input vector $\boldsymbol{x}$. We randomise the order of the input features, then train each model on the full data set using only the first C input dimensions, where C varies from 5 to 30. Figure 3.5(d) shows the results. In this experiment, run times for both

[4] Recall that, for ease of exposition, we consider the training data to consist of a single sequence. With multiple sequences, the time complexity is linear in the sum of lengths of all training sequences.

[5] Recall that we define sparsity as the fraction of off-diagonal elements that are non-zero (so a lower value is more sparse).



(a) Negative training log likelihood.

(b) Negative testing log likelihood.

(c) Sparsity.

(d) Run time vs input dimension.

Figure 3.5: Subfigures (a–c) show the evolution of the negative training log likelihood, negative testing log likelihood, and mean sparsity of the emission components as the models MGLASSO (blue line) and TGLASSO (red line) are trained. MGLASSO appears to have converged after about 20 seconds, TGLASSO after about 50 seconds. Subfigure (d) shows the average time per iteration as the dimensionality of the input is varied. MGLASSO with all 30 input dimensions is faster than any of the TGLASSO models.

models appear to scale approximately linearly with input dimension. MGLASSO with all 30 inputs runs more quickly than TGLASSO with only 5 inputs. Since MGLASSO and TGLASSO exhibit similar predictive performance provided that MGLASSO may utilise the inputs – see Figure 3.2 – the temporal connections in the TGLASSO model appear to bring little benefit for a substantial increase in training time in this experiment.

Recall that there are only 3 latent states ($M = 3$) in this experiment. Due to the linear versus quadratic scaling in M, the difference in training times between MGLASSO and TGLASSO may grow as more states are added. However, when applied to other problems, it may be that the temporal connections produce better performance for TGLASSO than MGLASSO – for example, if the inputs provide little information about the latent state. So it is possible that the performance penalty incurred by TGLASSO is acceptable in some cases.

In Section 2.5.5, we studied the computational costs of the CopMSLICE family of models. In particular, we examined the difference in training time between MSLICE and CopMSLICE (of which MGLASSO and CopMGLASSO are special cases). We expect the difference in training time between TGLASSO and CopTGLASSO to behave similarly, because evaluation and updating of the marginal functions is independent of the presence or absence of latent-state connections. We therefore omit CopTGLASSO from the experiments in this section.

3.4 Conclusions and Future Work

In this chapter, we introduced the IO-HMMs TGLASSO and CopTGLASSO – extensions of the MGLASSO and CopMGLASSO models introduced in Chapter 2, in which the discrete latent state depends on the state at the previous time step. We demonstrated that these models outperformed an existing baseline: an IO-HMM with factored emissions. In our experiments, MGLASSO and TGLASSO initially showed similar predictive performance, but when the input features were removed, the performance of MGLASSO degraded by more than that of TGLASSO. So we conclude that the market indices and technical indicators used as covariates in our experiment were sufficiently predictive of the latent state as to make the temporal connections unnecessary. This illustrates that with well-designed input features, the simpler, faster, MGLASSO model may be used in preference to TGLASSO, which is more expensive to train. However, we also conclude that the temporal connections may improve

performance when the side information is absent or weak.

The experiments of this chapter included a higher-dimensional input vector than the experiments of Section 2.5. We examined the inputs used by a trained MGLASSO model, and found them to be explainable. This suggests that the family of models introduced in this and the previous chapter may be useful for feature selection in decision support applications where interpretation of the learned models is necessary.

The future work discussed for CopMSLICE in Section 2.6 applies to TGLASSO and CopTGLASSO too. Adding latent variables to these models – to get TSLICE and CopTSLICE respectively – ought to be technically straightforward. But we suspect the resulting learning algorithms will be slow, perhaps to the extent of being impractical for some applications.

We would like to extend TGLASSO and CopTGLASSO by allowing the precision matrices to depend on their values at the previous time step. This might be useful in finance, for example, where we would expect dependence structures to change gradually over time: if the prices of two stocks are coupled, they are likely to remain coupled for multiple consecutive days. The prices may become decoupled by a business event, such as the two companies severing a business relationship, whence the absence of any mutual price dependence would be expected to persist for some time after. Such a process may be better modelled by allowing the precision matrices to depend on their previous values. However, we expect that introducing these temporal connections will not be straightforward: we may encounter similar problems as when we tried to introduce direct dependence of the precision matrices on the side information, as discussed in Section 2.6.

Chapter 4

Sparse Bayesian Gaussian Graphical Models

In Chapters 2 and 3, we investigated various sparse Gaussian models, using optimisation methods to estimate parameters. The methods in those chapters were typically based on the combination of L1 penalties and the maximum *a posteriori* (MAP) estimator to induce sparsity in the parameters. However, a Bayesian approach may be more principled, and preferable in some scenarios. Bayesian methods typically make use of the full posterior distribution – as opposed to a point estimate – which can make prediction more accurate. The posterior may also be more interpretable than a point estimate, since the latter does not indicate the degree of confidence in that value. Bayesian methods can make it easy to incorporate prior knowledge into a model, since the prior distribution is often represented explicitly.

A perceived drawback of the Bayesian approach is that it is often considered to be much slower than optimisation methods. However, Mohamed et al. (2012) recently compared the $L1$ approach with Bayesian methods based on the "spike-and-slab" prior, focussing on unsupervised linear latent variable models. They found that the Bayesian methods could outperform $L1$, even when both were constrained by the same time budget. Here, we address the question of whether a Bayesian method can outperform the $L1$ optimisation approach to infer sparse precision matrices.

The GWishart distribution is a generalisation of the Wishart that requires the matrix random variable to respect the structure of a given graph – see Section 4.1.2. A class of sparse Bayesian Gaussian graphical models (GGMs) based on the GWishart has been under development in parallel with the graphical lasso. These models have wide practical application. For example, Dobra et al. (2010) use them in a variable selection

problem, studying the determinants of macroeconomic growth; Lenkoski and Dobra (2011) use them in a regression setting to predict the volume of calls in a call centre; Dobra et al. (2011) apply sparse GGMs to model cancer mortality rates in the United States; Wang and Li (2012) use them to evaluate mutual fund performance; and Cheng and Lenkoski (2012) embed a GGM within a hierarchical Bayesian model to develop a stochastic volatility model for multivariate heteroskedastic financial data.

Inference in models based on sparse Bayesian GGMs is often limited by the efficiency with which the GWishart distribution can be sampled. Wang and Li (2012) demonstrate that the block Gibbs sampler is the current state of the art for this task. A more efficient sampler would make inference in GWishart-based models faster, and could make practical the use of more complex, higher-dimensional models. Hamiltonian Monte Carlo (HMC) samplers (see section 4.1.4) can facilitate fast mixing in distributions where the random variables are strongly coupled. Furthermore, they naturally take advantage of sparsity: the bottleneck in HMC is often the computation of the energy gradient with respect to the distribution parameters, and fewer parameters means fewer gradients to evaluate. We develop an HMC approach to sample from the GWishart distribution; in our experiments, it significantly outperforms the block Gibbs sampler in most cases.

This chapter expands on our work in (Orchard et al., 2013). The chapter is organised as follows. In the background material of Section 4.1, we describe the GWishart distribution and review current methods for drawing samples from it. We describe a spike-and-slab Gaussian model incorporating the GWishart, and discuss recent methods for performing inference in that model. We then review Hamiltonian Monte Carlo. In Section 4.2, we describe our contributions to improving sampling from the GWishart, beginning with a method for choosing the covering set of cliques in block Gibbs. We then describe our HMC approach, and in particular our choice of step size, trajectory length, and mass matrix – which are key to good performance. In the experiments in Section 4.3, we compare our new samplers to the existing block Gibbs approach across a range of dimensions, data sizes, and graphs of varying sparsity, demonstrating greatly improved efficiency in most cases. We then utilise the HMC sampler within a particular sparse GGM, and compare this model with the graphical lasso. We show that, on a real-world data set, the GGM can outperform graphical lasso, even when the methods are restricted to the same time budget. Conclusions and future work in Section 4.4 wrap up the chapter.

4.1 Background

4.1.1 The Wishart Distribution

Before discussing the GWishart, we briefly review the Wishart. Throughout this chapter, we use p to denote the dimensionality of the observed variables. (There are no latent variables or covariates in the models of this chapter, except in the discussion of future work). The Wishart $W(d, \boldsymbol{D})$ is a distribution over symmetric, non-negative definite matrices, with density[1]

$$p(\boldsymbol{\Lambda}) = \frac{1}{2^{\frac{dp}{2}} (\det \boldsymbol{D})^{-\frac{d}{2}} \Gamma\left(\frac{d}{2}\right)} (\det \boldsymbol{\Lambda})^{\frac{d-p-1}{2}} \exp\left[-\frac{1}{2} \operatorname{tr}(\boldsymbol{D}\boldsymbol{\Lambda})\right], \tag{4.1}$$

where $d > p-1$ is the degrees of freedom, $\boldsymbol{D}$ is a $p \times p$ positive-definite scale matrix, and Γ is the multivariate gamma function.

The Wishart distribution is the conjugate prior for the precision matrix of a multivariate Gaussian, and so finds common use in Bayesian statistics. Let $\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{-1}\right)$, and let the prior on $\boldsymbol{\Lambda}$ be $W(d, \boldsymbol{D})$. If N data points are observed, contained in the matrix $\boldsymbol{Y} \in \mathbb{R}^{N \times p}$, then the posterior distribution of $\boldsymbol{\Lambda}$ is $W(d+N, \boldsymbol{D} + \boldsymbol{Y}^T\boldsymbol{Y})$.

Sampling from the Wishart distribution is straightforward. Let $\boldsymbol{T}$ and $\boldsymbol{\Phi}$ denote the Cholesky decompositions of $\boldsymbol{D}^{-1}$ and $\boldsymbol{\Lambda}$ respectively, and define $\boldsymbol{\Psi}$ as follows:

$$\boldsymbol{D}^{-1} = \boldsymbol{T}^T\boldsymbol{T}; \;\; \boldsymbol{\Lambda} = \boldsymbol{\Phi}^T\boldsymbol{\Phi}; \;\; \boldsymbol{\Psi} = \boldsymbol{\Phi}\boldsymbol{T}^{-1}. \tag{4.2}$$

Then, samples of $\boldsymbol{\Lambda}$ can be generated by drawing samples of the upper triangular matrix $\boldsymbol{\Psi}$, which is easy because the elements of $\boldsymbol{\Psi}$ are independently distributed:

$$\Psi_{ii} \sim \chi^2(d - i + 1); \tag{4.3}$$

$$\Psi_{ij} \sim \mathcal{N}(0, 1), \; i < j; \tag{4.4}$$

where $\chi^2(k)$ is the chi-squared distribution with k degrees of freedom.

4.1.2 The GWishart Distribution

The GWishart distribution $W_G(b, \boldsymbol{D})$ generalises the Wishart by introducing a graph $G = (V, E)$, and requiring that any draw $\boldsymbol{\Lambda}$ respect the graph structure. That is, if an

[1] The Wishart distribution is usually defined in terms of a scale matrix $\boldsymbol{V} \equiv \boldsymbol{D}^{-1}$. We use $\boldsymbol{D}$ for consistency with the GWishart definition in Section 4.1.2.

edge is missing in the graph, the associated element of $\mathbf{\Lambda}$ must be zero. Precisely, the density is:

$$p(\mathbf{\Lambda}|\boldsymbol{G}) = \frac{1}{I_G(b,\boldsymbol{D})} (\det \mathbf{\Lambda})^{\frac{b-2}{2}} \exp\left[-\frac{1}{2}\operatorname{tr}(\boldsymbol{D}\mathbf{\Lambda})\right] 1_{[\mathbf{\Lambda}\in M^+(G)]}\,, \tag{4.5}$$

where b is the degrees of freedom parameter, $\boldsymbol{D}$ is the scale matrix, $1_{[]}$ is the indicator function, $M^+(G) = \{\mathbf{\Lambda} \in S_{++}$ such that $\Lambda_{ij} = 0$ if $(i,j) \notin E, i \neq j\}$, S_{++} is the cone of positive-definite matrices, and $I_G(b,\boldsymbol{D})$ is the normalisation constant.

Like the Wishart, the GWishart is the conjugate prior for the precision of a multivariate Gaussian. Let $\boldsymbol{y} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{\Lambda}^{-1}\right)$, and let $\boldsymbol{Y} \in \mathbb{R}^{N\times p}$ be a data matrix. The posterior distribution of $\mathbf{\Lambda}$ is easily shown to be $W_G(b+N, \boldsymbol{D}+\boldsymbol{Y}^T\boldsymbol{Y})$.

We introduce some notation for later use. Following Atay-Kayis and Massam (2005), we define

$$\mathcal{V} = \{(i,j), i \leq j \text{ such that either } i = j, i \in V \text{ or } (i,j) \in E\}, \tag{4.6}$$

$$\mathcal{W} = \{(i,j), i,j \in V, i \leq j\}, \tag{4.7}$$

$$\overline{\mathcal{V}} = \mathcal{W} \setminus \mathcal{V}. \tag{4.8}$$

Define $\mathbf{\Lambda}^{\mathcal{V}} = \{\Lambda_{ij} : (i,j) \in \mathcal{V}\}$, and let $\mathbf{\Lambda}_{\mathcal{V}}$ denote a column vector containing the elements of $\mathbf{\Lambda}^{\mathcal{V}}$ in the columnwise order in which they appear in $\mathbf{\Lambda}$.

4.1.2.1 Inference in the GWishart

Unlike for the Wishart, computing the normalisation constant $I_G(b,\boldsymbol{D})$ of the GWishart distribution is not straightforward. Lenkoski and Dobra (2011) showed empirically that a Gaussian approximation to the GWishart becomes more accurate as the dimensionality p increases. Therefore, in higher dimensions, a Laplace approximation to $I_G(b,\boldsymbol{D})$ may be justified. Atay-Kayis and Massam (2005) developed an approximate method in which they use the decomposition (4.2) and write the normalisation constant as the expectation of a function of the random matrix $\mathbf{\Psi}$. They approximate this expectation by drawing samples of the free elements $\mathbf{\Psi}^{\mathcal{V}}$. Elements of $\mathbf{\Psi}^{\mathcal{V}}$ are independently distributed – according to a chi-squared distribution for the diagonal elements, and a Gaussian for the off-diagonal elements. The elements of $\mathbf{\Psi}^{\overline{\mathcal{V}}}$ are not free: each one is a function of the elements of $\mathbf{\Psi}$ preceeding it in row-wise order. Thus, completing the matrix $\mathbf{\Psi}$ is an iterative operation, and this step is the bottleneck in computing the approximation of $I_G(b,\boldsymbol{D})$.

Since an exact $I_G(b,\boldsymbol{D})$ is not available, approximate inference is necessary. Various sampling procedures have been invented. The method of Atay-Kayis and Massam

(2005) for approximating $I_G(b, \boldsymbol{D})$ is straightforwardly adapted to drawing samples of $\boldsymbol{\Lambda}$: draw each $\boldsymbol{\Psi}$ as above, then compute $\boldsymbol{\Lambda}$ using (4.2). Each such $\boldsymbol{\Lambda}$ is an exact sample from $W_G(b, \boldsymbol{D})$. As in the normalisation constant approximation, the efficiency of this sampler is limited by the matrix completion operation. Exact samplers were also invented by Wang and Carvalho (2010), who developed a rejection sampler, and Mitsakakis et al. (2011), who developed a Metropolis-Hastings (MH) method. Dobra et al. (2011) demonstrated efficiency gains over these approaches using a random-walk Markov Chain Monte Carlo (MCMC) scheme in which each element of $\boldsymbol{\Lambda}^{\mathcal{V}}$ is updated conditionally on the other elements.

However, Wang and Li (2012) demonstrated that a block Gibbs sampler (Piccioni, 2000) convincingly outperformed all other existing methods. We describe it in the next section.

4.1.2.2 Sampling the GWishart with Block Gibbs

The GWishart block Gibbs sampler (Piccioni, 2000; Wang and Li, 2012) is based on the fact that a block of $\boldsymbol{\Lambda}$ corresponding to a clique in G can be sampled conditional on the rest of $\boldsymbol{\Lambda}$ by sampling from a Wishart. Thus, given a set of cliques that cover $\boldsymbol{\Lambda}^{\mathcal{V}}$, a sampler for the GWishart can be constructed by iterating over the covering set and conditionally sampling each block of $\boldsymbol{\Lambda}$.

More precisely, the block Gibbs algorithm is as follows. First, construct a set $I = \{I_k : 1 \leq k \leq K\}$ where $I_k \subset V$ such that all I_k are cliques, and $\cup_{I_k \in I} \boldsymbol{\Lambda}_{I_k, I_k} = \boldsymbol{\Lambda}^{\mathcal{V}}$. Initialise the Markov chain at some $\boldsymbol{\Lambda}$. Then, generate the next sample in the chain as follows: iterate over k, and at each step draw $\boldsymbol{A} \sim W(b, \boldsymbol{D}_{I_k, I_k})$, and set

$$\boldsymbol{\Lambda}_{I_k, I_k} = \boldsymbol{A} + \boldsymbol{\Lambda}_{I_k, V \setminus I_k} \left(\boldsymbol{\Lambda}_{V \setminus I_k, V \setminus I_k}\right)^{-1} \boldsymbol{\Lambda}_{V \setminus I_k, I_k}. \tag{4.9}$$

The choice of covering set I can have a significant effect on the performance of the sampler, and the optimal selection method probably depends on G. Wang and Li (2012) considered two choices for I: (1) The maximal cliques; (2) All pairs of nodes connected by an edge, plus all isolated nodes. It was found that maximal cliques gave better performance than the edgewise covering set in all the models considered. However, it seems likely that the set of all maximal cliques will be a suboptimal choice in many models. First, finding the maximal cliques is NP-hard, so this method may scale poorly. Second, the number of maximal cliques may be much greater than that required to cover $\boldsymbol{\Lambda}^{\mathcal{V}}$; a smaller covering set may trade off a little mixing quality for a

significant speed up. In Section 4.2.1, we describe a heuristic for choosing a smaller set of maximal cliques that still cover $\mathbf{\Lambda}^{\mathcal{V}}$.

A sampler for the GWishart was recently proposed by Lenkoski (2013) that is closely related to block Gibbs. Unlike block Gibbs, however, each sample is drawn independently. In this algorithm, T samples are drawn from $W_G(b, \boldsymbol{D})$ by repeating the following process T times. First, sample $\mathbf{\Lambda}^*$ from the Wishart $W(b, \boldsymbol{D})$, and compute the inverse $\mathbf{\Sigma} = (\mathbf{\Lambda}^*)^{-1}$. Then, set $\mathbf{\Lambda}^{(0)} = \boldsymbol{I}$, and construct a sequence $\left(\mathbf{\Lambda}^{(1)}, \mathbf{\Lambda}^{(2)}, \ldots\right)$, where $\mathbf{\Lambda}^{(t)}$ is computed from $\mathbf{\Lambda}^{(t-1)}$ by iterating over a covering set of cliques I, and applying Equation (4.9) for each clique I_k with $\boldsymbol{A} = \mathbf{\Sigma}^{-1}_{I_k, I_k}$. These deterministic updates cause $\mathbf{\Lambda}^{(t)}$ to converge to a matrix $\mathbf{\Lambda}$ (where $\mathbf{\Lambda}$ depends on $\mathbf{\Sigma}$). Lenkoski (2013) shows that $\mathbf{\Lambda}$ is a valid sample from $W_G(b, \boldsymbol{D})$. In practice, the chain must be terminated after a finite number of iterations. But if run for long enough such that $\mathbf{\Lambda}^{(t)}$ has converged to $\mathbf{\Lambda}$ to machine precision, then the sampler is exact.

4.1.3 A Spike-and-Slab Gaussian Graphical Model

The following spike-and-slab Gaussian graphical model (GGM) utilises the GWishart in a prior over sparse precisions of a Gaussian distribution:

$$\boldsymbol{G} \sim P(\boldsymbol{G}), \tag{4.10}$$

$$\mathbf{\Lambda} \sim W_G(b, \boldsymbol{D}), \tag{4.11}$$

$$\boldsymbol{y} \sim \mathcal{N}\left(\boldsymbol{0}, \mathbf{\Lambda}^{-1}\right). \tag{4.12}$$

$P(\boldsymbol{G})$ is an arbitrary distribution on graphs; we describe some of the choices appearing in the literature in Section 4.1.3.2. This model has received considerable attention in recent years; see, for example (Dobra et al., 2011; Rodriguez et al., 2011; Dobra and Lenkoski, 2011; Wang and Li, 2012).

4.1.3.1 Inference

Bayesian inference in this model typically involves sampling. One of the most efficient existing methods (Wang and Li, 2012), which we refer to as WL, iterates two steps. The first is to draw a sample of $\mathbf{\Lambda}$ given $\boldsymbol{G}$ and $\boldsymbol{Y}$, which of course is a sample from the GWishart distribution $W_G\left(b+N, \boldsymbol{D}+\boldsymbol{Y}^T\boldsymbol{Y}\right)$. The second step is to resample the graph $\boldsymbol{G}$ given the new $\mathbf{\Lambda}$. WL does this via a reversible-jump algorithm: they propose to flip a single edge G_{ij}, which requires that Λ_{ij} and Λ_{jj} be resampled from their conditional distributions. This move is then accepted or rejected according to Metropolis-Hastings.

However, the basic form of this algorithm involves computing a ratio of GWishart normalisation constants, which is computationally intensive. To avoid this, WL uses the double-Metropolis-Hastings method (Liang, 2010), which is an approximate version of the exchange algorithm (Murray et al., 2006). The exchange algorithm is designed to sample from distributions with problematic normalisation constants, and involves drawing exact samples of an auxiliary variable. In the double-MH algorithm, the auxiliary variable is updated using a sequence of Metropolis-Hastings kernels, which may reduce computation time. Note, however, that the double-MH algorithm does not result in a valid sampler, but an approximation, and so the WL algorithm for sampling the GGM is also an approximation.

The WL procedure – with the GWishart sampled using our HMC method to be described in Section 4.2.2 – is summarised in Algorithm 6. The functions that we name *ProposalRate*, *AcceptanceRate*, and *ResampleEdge* contain many of the details for the sampling of $\boldsymbol{G}|\boldsymbol{\Lambda}$. We do not describe these details here, but refer the reader to Wang and Li (2012).

Cheng and Lenkoski (2012) propose a modification to the WL algorithm to reduce the time required to compute a conditional Bayes factor (CBF) required by WL. They do this by working with the Cholesky decomposition of the precision matrix, and permuting its rows and columns to make the CBF computation faster.

Lenkoski (2013) addresses the issue that the use of double-MH within WL makes the WL sampler approximate. Lenkoski (2013) uses his exact sampler for the GWishart (see Section 4.1.2.2) to propose an exact, double-reversible-jump, sampler for the sparse GGM that is based on the exchange algorithm. Lenkoski (2013) does not make experimental comparisons of this sampler with WL, but it seems likely that the price of exactness is a greater computational cost: each sample drawn by the exact GWishart sampler requires the convergence of the sequence $\boldsymbol{\Lambda}^{(t)}$ to $\boldsymbol{\Lambda}$ (see Section 4.1.2.2), whereas the MCMC sampler used within WL need only make a single update.

Another recent approach to sampling in this sparse GGM model is due to Mohammadi and Wit (2012) who use birth-death MCMC (BDMCMC) (Stephens, 2000) to make changes to the graph. As in WL, the issue of computing a ratio of normalisation constants appears – here, when computing the death rate. In low dimensions, Mohammadi and Wit (2012) approximate the normalisation constants using the Monte Carlo method of Atay-Kayis and Massam (2005). For the high-dimensional case, they approximate the ratio by setting it equal to one, and support this choice by performing an

experiment in which the ratio appears to converge to one as dimensionality increases. To sample the GWishart, they use block Gibbs.

The efficiency of WL depends strongly on the efficiency of the GWishart sampler. We expect this is the case for BDMCMC too. Having introduced more efficient methods for sampling the GWishart in Sections 4.2.1 and 4.2.2, we use them within the WL algorithm to do inference in the GGM described by Equations (4.10 – 4.12). We compare this Bayesian model with the optimisation-based graphical lasso in Section 4.3.4.

4.1.3.2 Graph Priors

Various choices of the graph prior in (4.10) have been studied in the literature; we mention a few of them here. The uniform prior is often used – (Roverato, 2002), (Dobra and Lenkoski, 2011), (Wang and Li, 2012), for example – which keeps computation simple. Wong et al. (2003) use a prior that gives more weight to graphs whose potential precision matrices fill a large volume of the positive-definite cone. Jones et al. (2005) encourage sparsity using a Bernoulli prior on each edge of the graph; this is the approach we take in our experiments in Section 4.3. Mohammadi and Wit (2012) use a truncated Poisson distribution on the edge degree.

The fact that the graph prior is explicit in the GGM model makes it more flexible for including prior information than the graphical lasso described in Section 2.1.2. For example, in the GGM, we can make the prior probability of including an edge dependent on whether or not another set of edges is included. This cannot be done in the graphical lasso. There, the probability that $\Lambda_{ij} = 0$ depends on Γ_{ij} (the strength of the applied $L1$ penalty), but Γ_{ij} is constant: it cannot depend on the other elements of $\mathbf{\Lambda}$. An extension of the graphical lasso – perhaps including group lasso penalties (Yuan and Lin, 2006) – may improve its flexibility, but in the Bayesian GGM, any prior knowledge about the graph is easily incorporated. We note, however, one restriction imposed by the WL and BDMCMC samplers, or indeed any sampler that switches a single edge at a time: we cannot use a graph prior that requires a set of edges be all included or all omitted. With such a prior, the samplers would never be able to switch between the two states. This is not a problem with the model, of course, and samplers developed in the future may lift this restriction.

4.1.4 Hamiltonian Monte Carlo

Simple MCMC samplers that exhibit random walk behaviour – such as a Gibbs sampler, or random-walk Metropolis-Hastings – may mix poorly, especially when some of the random variables are strongly correlated. Hamiltonian Monte Carlo (HMC) (see, for example, (Neal, 1993, 2010)), otherwise known as hybrid Monte Carlo, is designed to mix more rapidly by exploiting an analogy between the negative log density to be sampled and the potential energy of a physical system. Let $\underline{\boldsymbol{y}}$ be a random vector with density

$$p(\boldsymbol{y}) = Z^{-1} \exp(-E(\boldsymbol{y})), \tag{4.13}$$

where Z is a normalisation constant. The intuition is that if $E(\boldsymbol{y})$ is identified as a potential energy function, a particle moving in that potential according to physical laws would be accelerated towards regions of lower energy (higher density), and the particle's momentum would introduce some persistence to the direction of motion, thus avoiding random walk behaviour.

HMC introduces an auxiliary random vector $\underline{\boldsymbol{p}} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{M})$. Since $\underline{\boldsymbol{y}}$ and $\underline{\boldsymbol{p}}$ are independent, their joint density is

$$p(\boldsymbol{y}, \boldsymbol{p}) = Z'^{-1} \exp\left(-E(\boldsymbol{y})\right) \exp\left(-\frac{1}{2}\boldsymbol{p}^T \boldsymbol{M}^{-1} \boldsymbol{p}\right), \tag{4.14}$$

where Z' combines Z and the Gaussian normalisation constant. Define

$$H(\boldsymbol{y}, \boldsymbol{p}) = E(\boldsymbol{y}) + \frac{1}{2}\boldsymbol{p}^T \boldsymbol{M}^{-1} \boldsymbol{p}, \tag{4.15}$$

so that (4.14) may be written in a similar form to (4.13). Notice that if we can draw samples from the joint $(\boldsymbol{y}, \boldsymbol{p})$, we can sample from the marginal of $\boldsymbol{y}$ simply by discarding the values of $\boldsymbol{p}$.

In HMC, we interpret $\boldsymbol{y}$ as a position vector, and $\boldsymbol{p}$ as a corresponding momentum vector. Classical mechanics may be formulated in terms of a Hamiltonian $H(\boldsymbol{y}, \boldsymbol{p})$ and Hamilton's equations:

$$\frac{dy_i}{dt} = \frac{\partial H}{\partial p_i}, \tag{4.16}$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial y_i}. \tag{4.17}$$

The Hamiltonian represents the total energy of the system, and in HMC it is defined according to (4.15). The two terms on the right hand side of (4.15) are now interpreted

as potential and kinetic energy respectively, and $\boldsymbol{M}$ is interpreted as the mass matrix. With this Hamiltonian, (4.16 – 4.17) become

$$\dot{\boldsymbol{y}} = \boldsymbol{M}^{-1}\boldsymbol{p}, \tag{4.18}$$

$$\dot{\boldsymbol{p}} = -\nabla E(\boldsymbol{y}). \tag{4.19}$$

It can be shown that if the dynamics defined by these equations can be simulated exactly, then alternating the following two steps constitutes a valid MCMC sampler:

1. Draw $\boldsymbol{p}$ from $\mathcal{N}(\boldsymbol{0}, \boldsymbol{M})$;

2. Simulate the dynamics for some fixed time, and record the final value of $(\boldsymbol{y}, \boldsymbol{p})$.

If the dynamics are simulated for some time t, this defines a transformation $\mathcal{D}_t$ from the phase space $(\boldsymbol{y}, \boldsymbol{p})$ to itself. The proof that the above is a valid sampler rests on the following properties of this map.

- Reversibility. $\mathcal{D}_t$ has an inverse $\mathcal{D}_{-t}$ obtained by running time backwards (negating the right hand sides of (4.18 – 4.19)).

- Conservation of energy. The Hamiltonian is invariant under $\mathcal{D}_t$.

- Symplecticness. The Jacobian of $\mathcal{D}_t$ is a symplectic matrix. This implies volume preservation: volumes in phase space are mapped to equal-sized volumes by $\mathcal{D}_t$.

But Hamiltonian dynamics cannot be simulated exactly, and a method is required to approximately integrate (4.18 – 4.19). The leapfrog integrator is typically used. Given a step size ε and current state $(\boldsymbol{y}_0, \boldsymbol{p}_0) = (\boldsymbol{y}(t), \boldsymbol{p}(t))$, the leapfrog algorithm generates $(\boldsymbol{y}_1, \boldsymbol{p}_1) = (\boldsymbol{y}(t + L\varepsilon), \boldsymbol{p}(t + L\varepsilon))$ by iterating the following updates for L steps:

$$\boldsymbol{p}\left(\tau + \frac{\varepsilon}{2}\right) = \boldsymbol{p}(\tau) - \frac{\varepsilon}{2}\nabla E\left(\boldsymbol{y}(\tau)\right), \tag{4.20}$$

$$\boldsymbol{y}(\tau + \varepsilon) = \boldsymbol{y}(\tau) + \varepsilon \boldsymbol{M}^{-1}\boldsymbol{p}\left(\tau + \frac{\varepsilon}{2}\right), \tag{4.21}$$

$$\boldsymbol{p}(\tau + \varepsilon) = \boldsymbol{p}\left(\tau + \frac{\varepsilon}{2}\right) - \frac{\varepsilon}{2}\nabla E\left(\boldsymbol{y}(\tau + \varepsilon)\right). \tag{4.22}$$

This integrator is reversible and symplectic, but it does not conserve the Hamiltonian. To correct for this, and maintain a valid sampler, a Metropolis-Hastings step is introduced. The proposed value $(\boldsymbol{y}_1, \boldsymbol{p}_1)$ is accepted with probability

$$\min\{1, \exp[H(\boldsymbol{y}_0, \boldsymbol{p}_0) - H(\boldsymbol{y}_1, \boldsymbol{p}_1)]\}, \tag{4.23}$$

and otherwise rejected.

The parameters ε, L, and $\boldsymbol{M}$ must be chosen manually by the user. Poor choices can have a dramatic effect on the performance of the sampler. The typical practice is to perform preliminary runs with different choices of parameters, and examine metrics such as the acceptance rate and autocorrelation. See Neal (2010) for details on using HMC in practice. Here, we make some observations relevant to our particular use of HMC in Section 4.2.2. First, it is legal to draw (ε, L) from some joint distribution at each time step. Using a distribution can be helpful because different values may perform better in different regions of phase space (but the distribution itself must not depend on the current state).

The mass matrix $\boldsymbol{M}$ is the covariance of momentum vector $\boldsymbol{p}$, and must therefore be positive semi-definite. With the kinetic energy defined as $\boldsymbol{p}^T \boldsymbol{M}^{-1} \boldsymbol{p}$, the mass may also be identified as the inertia of the hypothetical particle. That is, it describes the particle's resistance to acceleration in each direction. It is preferable to move slowly in directions in which the energy gradient changes rapidly because the leapfrog approximation can then lead to a high probability of the proposal being rejected. Of course, moving too slowly means a low mixing rate. So, intuitively, we want the mass matrix to be adapted to the shape of the local energy surface. In standard HMC, $\boldsymbol{M}$ cannot depend on the current state, so a mass must be chosen that will work reasonably well in all regions of the state space. Neal (2010) suggests setting $\boldsymbol{M} = \boldsymbol{W}^{-1}$, where $\boldsymbol{W}$ is an estimate of the covariance of $\boldsymbol{y}$. Alternatively, one could use a method such as Riemann manifold HMC (Girolami and Calderhead, 2011) which does allow the mass matrix to depend on the current state – but at a considerable cost in computation time.

The choice of initial value for the Markov chain depends on the distribution $p(\boldsymbol{y})$ to be sampled. Considerations are the same as those for any MCMC method. Typically, one would use some reasonable value such as the mode of $p(\boldsymbol{y})$ (if known), and then run the chain for a "burn-in" period to achieve adequate mixing before samples are recorded.

The HMC procedure is summarised in algorithm 5.

4.2 Improved Sampling in the GWishart

In this section, we describe our contributions to sampling from the GWishart distribution with greater efficiency.

Algorithm 5 The HMC Sampler

$\boldsymbol{y}$: Initial state
E: Energy function
$\boldsymbol{M}$: Mass matrix
$P_{\varepsilon L}$: Joint distribution of step size ε and number of steps L
T: Number of samples required
function HMC($\boldsymbol{y}, E, \boldsymbol{M}, P_{\varepsilon L}, T$)
  for $t \leftarrow 1 : T$ **do**
    Draw $\boldsymbol{p} \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{M})$
    Draw $(\varepsilon, L) \sim P_{\varepsilon L}$
    $\boldsymbol{y}^*, \boldsymbol{p}^* \leftarrow \boldsymbol{y}, \boldsymbol{p}$
    for $l \leftarrow 1 : L$ **do**
      $\boldsymbol{y}^*, \boldsymbol{p}^* \leftarrow$ LEAPFROG($\boldsymbol{y}^*, \boldsymbol{p}^*, E, \boldsymbol{M}, \varepsilon$)
    end for
    if RANDOM$(0,1) <$ exp$[H(\boldsymbol{y}, \boldsymbol{p}) - H(\boldsymbol{y}^*, \boldsymbol{p}^*)]$ **then**
      $\boldsymbol{y}, \boldsymbol{p} \leftarrow \boldsymbol{y}^*, \boldsymbol{p}^*$
    end if
  end for
end function
function LEAPFROG($\boldsymbol{y}, \boldsymbol{p}, E, \boldsymbol{M}, \varepsilon$)
  $\boldsymbol{p} \leftarrow \boldsymbol{p} - \frac{\varepsilon}{2} \nabla E(\boldsymbol{y})$
  $\boldsymbol{y} \leftarrow \boldsymbol{y} + \varepsilon \boldsymbol{M}^{-1} \boldsymbol{p}$
  $\boldsymbol{p} \leftarrow \boldsymbol{p} - \frac{\varepsilon}{2} \nabla E(\boldsymbol{y})$
  return $\boldsymbol{y}, \boldsymbol{p}$
end function

4.2.1 Choosing the Covering Set in Block Gibbs

As discussed in Section 4.1.2.2, the choice of covering set I affects both the run time and the quality of mixing. We investigate the use of a simple heuristic to build the set I. Our goals are to build large cliques to facilitate mixing, but to reduce run time by building the set quickly and keeping the number of cliques small. Note the contrast to the two methods investigated by Wang and Li (2012): generating the full set of maximal cliques may be slow because it is an NP-hard problem, and may result in a large set; the set of edges and unconnected nodes may also be a large set, and the lack of larger cliques may hinder mixing.

Our procedure is as follows.

1. Generate a permutation π of the integers $(1,...,p)$.
2. Permute the rows and columns of $\boldsymbol{G}$ according to π.
3. Iterate over the entries of $\boldsymbol{G}$. For each G_{ij} not yet included in a clique:
   (a) Create a new clique $C = \{i, j\}$;
   (b) Grow C by considering nodes in the order π, and adding them greedily.

We permute the nodes to avoid creating a bias due to labelling. Rather than finding all maximal cliques, this algorithm quickly finds a small covering set of maximal cliques, which should improve the efficiency of the block Gibbs sampler in high dimensional models. We investigate this experimentally in Section 4.3.2.

4.2.2 Sampling the GWishart with HMC

Here we develop an HMC method to sample from the GWishart distribution. The first issue to deal with is that the GWishart $W_G(b, \boldsymbol{D})$ is defined over positive-definite matrices $\boldsymbol{\Lambda} \in M^+(G)$, so when running HMC, we must ensure that $\boldsymbol{\Lambda}$ remains within the positive-definite cone S_{++}. Our first approach was to use the Cholesky decomposition in Equation (4.2), and apply HMC to the free variables $\boldsymbol{\Psi}^{\mathcal{V}}$. (See Section 4.1.2 for the definition of $\boldsymbol{\Psi}^{\mathcal{V}}$). The energy function becomes

$$E\left(\boldsymbol{\Psi}^{\mathcal{V}}\right) = -\sum_{i=1}^{p} (b + \nu_i - 1) \log \Psi_{ii} + \frac{1}{2} \sum_{1 \le i \le j \le p} \Psi_{ij}^2, \tag{4.24}$$

where $\nu_i \equiv |\{j : j > i, G_{ij} = 1\}|$. There are no constraints on $\boldsymbol{\Psi}$: any $\boldsymbol{\Psi}$ produces a positive-definite $\boldsymbol{\Lambda}$. The problem with this representation is that the non-free elements

$\boldsymbol{\Psi}^{\overline{\mathcal{V}}}$ may depend on all elements of $\boldsymbol{\Psi}$ that precede them in a row-wise order. This means that, given $\boldsymbol{\Psi}^{\mathcal{V}}$, the matrix $\boldsymbol{\Psi}$ must be completed iteratively. For HMC, the energy gradients – derived in Appendix D – depend on the gradients $\frac{\partial \Psi_{rs}}{\partial \Psi_{ij}}$. For each Ψ_{ij}, these must also be evaluated in row-wise order, which makes HMC very slow in this representation, as we show empirically in Section 4.3.2.

Applying HMC in the space of $\boldsymbol{\Lambda}$, the energy function is

$$E(\boldsymbol{\Lambda}) = \frac{1}{2}\left[\operatorname{tr}(\boldsymbol{D}\boldsymbol{\Lambda}) - (b-2)\log\det\boldsymbol{\Lambda}\right], \tag{4.25}$$

and its gradient is

$$\frac{\partial E}{\partial \boldsymbol{\Lambda}} = \frac{1}{2}\left[2\boldsymbol{D} - \boldsymbol{D}\odot\boldsymbol{I} - (b-2)\left(2\boldsymbol{\Sigma} - \boldsymbol{\Sigma}\odot\boldsymbol{I}\right)\right], \tag{4.26}$$

where $\odot$ is the Hadamard product operator; see Appendix D. A naive approach to maintaining positive-definiteness is simply to set the energy to infinity outside S_{++}. However, this may lead to a high rejection rate if the approximate dynamics often lead to proposals outside the cone. When $b > 2$, the energy approaches infinity at the boundary, so the chain would remain in S_{++} if the dynamics could be simulated exactly. But in practice, the leapfrog method can overshoot the boundary. For $b \leq 2$, the situation is worse as even the exact dynamics lead the chain outside S_{++}.

Another idea is to reflect the simulated path off the constraint boundary. If the constraints were independent bounds on each variable, this would be straightforward. But for the positive-definite constraint, it is non-trivial to find where the path crosses the boundary, or to find the tangent plane at that point.

Fortunately, we find that judicious choices of the step size and trajectory length are sufficient to achieve good performance. We want the chain to be able to escape the regions near the S_{++} boundary – where accurate simulation of the dynamics is required – but we also want fast mixing. Intuitively, it seems that a distribution is required that concentrates much of its mass near the mean, but results in the occasional draw of a small value.

We choose a fixed target trajectory length $L = \max(1, \lfloor \beta/\varepsilon \rceil)$, where β is a user-defined parameter, so that when a small ε is chosen, the chain still moves a long distance. With this fixed trajectory length, the distribution of ε cannot have too much mass near zero, or the sampler will be slow. This rules out the exponential distribution, for example. We find empirically that a $\Gamma(2,\alpha)$ distribution works well, provided that the mass matrix is well-chosen (see Section 4.2.2.1) and the degrees of freedom parameter b is greater than 10 or so. This is almost always the case for a posterior GWishart

because b increases with the number of data points. But this HMC approach may not be the best choice for GWishart priors with small b.

The parameters (α, β) may be chosen by performing preliminary runs (as is typical with HMC). The time to generate each sample is reduced with a larger α (greater step size). However, the acceptance rate usually drops as α is increased. We follow the advice of Neal (2010) and aim for an acceptance rate of around 65%. Our practical experience suggests that as b (the GWishart degrees of freedom parameter) becomes smaller, α must be reduced to maintain this acceptance rate. The acceptance rate seems particularly sensitive to α when b is very small – less than 10, say. We speculate that this is because more of the distribution's mass becomes concentrated near the boundary of the positive-definite cone when b is reduced, raising the chance that a trajectory will leave the cone and get rejected. A smaller α is required to compensate. For β, a smaller value (shorter trajectory) means that each sample is generated more quickly, but a larger value may reduce autocorrelations in the sequence of samples. One way to balance these effects is to aim for a high effective sample size (ESS, defined in Equation 4.27) per second, while varying β during preliminary runs. The acceptance rate must also be maintained, because β affects that too.

4.2.2.1 The Mass Matrix

To run HMC (see Algorithm 5) on the GWishart, we arrange elements in the upper triangle of $\boldsymbol{\Lambda}$ into a vector $\boldsymbol{\Lambda}_{\mathcal{V}}$, in the column-wise order they appear in $\boldsymbol{\Lambda}$. There may be strong correlations between the elements of $\boldsymbol{\Lambda}_{\mathcal{V}}$, so we find that the mass matrix strongly influences the performance of HMC. We take the approach of estimating the covariance $\boldsymbol{W}$ of this random vector and set $\boldsymbol{M} = \boldsymbol{W}^{-1}$.

In estimating the covariance, we need to consider both speed and accuracy. If we wish to take many samples from the same GWishart, it may be reasonable to spend more time estimating $\boldsymbol{W}$. But the GWishart is often a component of a larger model such as the GGM described in Section 4.1.3. In that case, the GWishart parameters $(b, \boldsymbol{D}, \boldsymbol{G})$, and therefore $\boldsymbol{W}$ may have changed each time HMC is required to generate a new sample. If it needs to be computed at each iteration, estimation of $\boldsymbol{W}$ will need to run quickly.

We considered the following methods for estimating $\boldsymbol{W}$, which we compare experimentally in Section 4.3.3.

1. Set $\boldsymbol{W}$ to the identity matrix.

2. Perform a short block Gibbs run and set $\boldsymbol{W}$ to the empirical covariance.

3. Compute a Laplace approximation to the distribution of $\boldsymbol{\Lambda}_{\mathcal{V}}$, and set $\boldsymbol{W}$ to its covariance.

4. Assume $\boldsymbol{\Lambda} \sim W(b, \boldsymbol{D})$, draw samples from the Wishart, and compute the empirical precision $\boldsymbol{K}$ of the vector $\boldsymbol{\Lambda}_{\mathcal{W}}$. Now make the approximation that $\boldsymbol{\Lambda}_{\mathcal{W}}$ is Gaussian-distributed: the conditional precision given that $\boldsymbol{\Lambda}_{\overline{\mathcal{V}}} = \mathbf{0}$ is simply $\boldsymbol{K}_{\mathcal{V},\mathcal{V}}$. We set $\boldsymbol{W} = \boldsymbol{K}^{-1}_{\mathcal{V},\mathcal{V}}$ (so that $\boldsymbol{M} = \boldsymbol{K}_{\mathcal{V},\mathcal{V}}$). Notice that (provided b and $\boldsymbol{D}$ remain constant) $\boldsymbol{M}$ can be recomputed when the graph changes without drawing any further samples.

Both 3 and 4 utilise Gaussian approximations. Lenkoski and Dobra (2011) showed empirically that this approximation becomes more accurate as b increases.

Algorithm 6 shows how to use the HMC sampler for the GWishart with method 4 for computing the mass matrix, within the WL sampler for the GGM described in Section 4.1.3. The WL algorithm requires samples from both the prior and posterior, so we precompute precision matrices $\boldsymbol{K}_0$ and $\boldsymbol{K}_N$ for each case by sampling the associated Wishart distributions. Notice that whenever the graph changes, computing the mass is as straightforward as taking a subset of the rows and columns of $\boldsymbol{K}_0$ or $\boldsymbol{K}_N$. In Algorithm 6, the same distribution $P_{\varepsilon L}$ for the HMC step size and step number is used for both the prior and posterior. Typically, these would be different distributions, chosen via preliminary runs.

4.3 Evaluation

4.3.1 Verification of Correctness

First, we verify the correctness of our implementations of the GWishart samplers. We do this by a drawing a large set of samples from the same GWishart distribution with each sampler, and comparing their summary statistics. We would expect that, as the number of samples increases, each sampler's statistics should tend towards the same values.

We generate a random sparse graph $\boldsymbol{G}$ with 10 vertices; draw a random precision matrix $\boldsymbol{\Lambda}$ from a GWishart distribution conditioned on $\boldsymbol{G}$; and then draw 50 data points from a Gaussian with precision $\boldsymbol{\Lambda}$. We draw 100000 samples from the posterior GWishart using the following samplers.

Algorithm 6 The WL Algorithm, Incorporating the HMC GWishart Sampler

Given

$b_0, \boldsymbol{D}_0$ ▷ GWishart prior parameters

$\boldsymbol{Y}$ ▷ $N \times p$ data matrix

$\boldsymbol{G}, \boldsymbol{\Lambda}$ ▷ Initial state of the Markov chain

$P_{\varepsilon L}$ ▷ Joint distribution of HMC step size and number of steps

T ▷ Number of samples required

end Given

$b_N \leftarrow b_0 + N$; $\boldsymbol{D}_N \leftarrow \boldsymbol{D}_0 + \boldsymbol{Y}^T\boldsymbol{Y}$ ▷ GWishart posterior parameters

$\boldsymbol{K}_0 \leftarrow$ ESTIMATEWISHARTPRECISION$(b_0, \boldsymbol{D}_0)$

$\boldsymbol{K}_N \leftarrow$ ESTIMATEWISHARTPRECISION$(b_N, \boldsymbol{D}_N)$

$\boldsymbol{M}_N \leftarrow (\boldsymbol{K}_N)_{\mathcal{V},\mathcal{V}}$

for $t \leftarrow 1 : T$ **do**

  $P_G \leftarrow$ SAMPLEHYPER$(\boldsymbol{G})$ ▷ Sample hyperparameters of the graph prior

  for each possible edge (i, j) **do**

    $\boldsymbol{G}' \leftarrow \boldsymbol{G}$

    Flip edge (i, j) of the proposal $\boldsymbol{G}'$

    $R \leftarrow$ PROPOSALRATE$(P_G, \boldsymbol{G}, \boldsymbol{G}', \boldsymbol{\Lambda}, b_N, \boldsymbol{D}_N)$

    if RANDOM$(0,1) < R$ **then**

      $\boldsymbol{\Lambda}' \leftarrow$ RESAMPLEEDGE$(i, j, \boldsymbol{\Lambda}, b_0, \boldsymbol{D}_0)$

      $\boldsymbol{M}_0 \leftarrow (\boldsymbol{K}_0)_{\mathcal{V}',\mathcal{V}'}$

      $E \leftarrow$ GWENERGY$(\boldsymbol{G}', b_0, \boldsymbol{D}_0)$ ▷ Energy function for the prior

      $\boldsymbol{\Lambda}'_{\mathcal{V}'} \leftarrow$ HMC$(\boldsymbol{\Lambda}'_{\mathcal{V}'}, E, \boldsymbol{M}_0, P_{\varepsilon L}, 1)$ ▷ See Algorithm 5

      $R \leftarrow$ ACCEPTANCERATE$(i, j, \boldsymbol{\Lambda}', b_0, \boldsymbol{D}_0)$

      if RANDOM$(0,1) < R$ **then**

        $\boldsymbol{G} \leftarrow \boldsymbol{G}'$

        $\boldsymbol{M}_N \leftarrow (\boldsymbol{K}_N)_{\mathcal{V},\mathcal{V}}$

        $\boldsymbol{\Lambda} \leftarrow$ RESAMPLEEDGE$(i, j, \boldsymbol{\Lambda}, b_N, \boldsymbol{D}_N)$

      end if

    end if

    $E \leftarrow$ GWENERGY$(\boldsymbol{G}, b_N, \boldsymbol{D}_N)$ ▷ Energy function for the posterior

    $\boldsymbol{\Lambda}_{\mathcal{V}} \leftarrow$ HMC$(\boldsymbol{\Lambda}_{\mathcal{V}}, E, \boldsymbol{M}_N, P_{\varepsilon L}, 1)$ ▷ See Algorithm 5

  end for

end for

1. HMC: Our Hamiltonian Monte Carlo GWishart sampler. The step size and trajectory length are chosen as described in Section 4.2.2. The mass matrix is computed by taking preliminary samples from a Wishart – method 4 in Section 4.2.2.1.

2. BG-HCC: Block Gibbs in which the covering set is generated using our heuristic clique cover algorithm; see Section 4.2.1.

3. BG-MC: Block Gibbs in which the covering set consists of all maximal cliques.

4. DLR: The random-walk Metropolis-Hastings sampler of Dobra et al. (2011).

The covering sets for BG-MC and BG-HCC are different on the graph used in this experiment.

Table 4.1 shows the mean computed with the set of samples drawn by each of the four samplers. The means of the samples drawn by HMC, BG-HCC, and BG-MC are all very similar. The DLR mean is a little further away, but this is expected because DLR mixes more slowly than BG-MC (Wang and Li, 2012). The results are consistent with correct implementations of the samplers.

We also computed the variance using each set of samples. We do not show them, but the values for each sampler were close, in a similar way to the means. We repeated the experiment on different randomly drawn graphs, and for different numbers of dimensions. The similarity of the summary statistics in each case lead us to believe that the samplers are correctly implemented.

4.3.2 Comparing the HMC and Block Gibbs Samplers

We compare HMC and block Gibbs on synthetically generated data, testing the effects of dimensionality, data size, and sparsity on the efficiency of these samplers. Each test case corresponds to a setting of the model dimensionality p; a sparsity parameter s where $0 \leq s \leq 1$; and the ratio N/q, where N is the number of data points and $q = q(s)$ is the expected number of free variables.

Each test case is composed of 10 runs. In each run, a graph G is drawn by sampling each edge from $Bern(s)$, and a precision matrix $\mathbf{\Lambda}$ is drawn from $W_G(1, p\boldsymbol{I}_p)$ by taking the 1000^{th} sample from a block Gibbs run. The N data points are then drawn from $\mathcal{N}\left(\mathbf{0}, \mathbf{\Lambda}^{-1}\right)$.

Table 4.1: Mean precision matrices computed by averaging the 100000 samples drawn by 4 different GWishart samplers. (Only the upper triangles are shown because the precision is symmetric). The means computed by HMC, BG-HCC, and BG-MC are very similar; DLR is a little further away, but we expect DLR to converge more slowly than the other samplers. The results suggest that the samplers are correctly implemented.

Sampler	**Mean Precision**									
HMC	0.7604	0.1367	0	0	0.3997	0.1743	0	0	0.1838	0.0816
		0.6280	−0.0486	−0.1865	−0.0046	0	−0.1822	0.0361	0	0.2077
			0.1018	0.1342	0	0	0	0	0	0
				1.1180	0	0.0218	0	−0.3076	0	0
					0.2670	0.0760	0	0	0	0
						0.3982	0	−0.3099	0.0746	0.3490
							0.6691	−0.1667	0	0
								0.6351	0	−0.1011
									0.5071	0.0479
										0.6348
BG-HCC	0.7607	0.1368	0	0	0.3999	0.1743	0	0	0.1838	0.0814
		0.6278	−0.0486	−0.1868	−0.0044	0	−0.1822	0.0361	0	0.2075
			0.1016	0.1342	0	0	0	0	0	0
				1.1170	0	0.0218	0	−0.3070	0	0
					0.2671	0.0759	0	0	0	0
						0.3978	0	−0.3094	0.0747	0.3489
							0.6690	−0.1666	0	0
								0.6340	0	−0.1011
									0.5069	0.0480
										0.6346
BG-MC	0.7606	0.1365	0	0	0.3999	0.1744	0	0	0.1837	0.0816
		0.6280	−0.0486	−0.1869	−0.0048	0	−0.1821	0.0363	0	0.2079
			0.1018	0.1342	0	0	0	0	0	0
				1.1180	0	0.0222	0	−0.3083	0	0
					0.2671	0.0761	0	0	0	0
						0.3981	0	−0.3100	0.0745	0.3488
							0.6685	−0.1666	0	0
								0.6353	0	−0.1012
									0.507	0.0478
										0.6345
DLR	0.7641	0.1389	0	0	0.4006	0.1751	0	0	0.1848	0.0818
		0.6325	−0.0487	−0.1885	−0.0038	0	−0.1824	0.0356	0	0.2082
			0.1016	0.1343	0	0	0	0	0	0
				1.116	0	0.0229	0	−0.3074	0	0
					0.2668	0.0763	0	0	0	0
						0.3980	0	−0.3097	0.0743	0.3488
							0.6686	−0.1664	0	0
								0.6344	0	−0.1017
									0.5101	0.0469
										0.6337

We sample from the posterior of each test run using HMC in which the mass matrix is computed by sampling from a (fully connected) Wishart distribution and then conditioning on the missing edges as described in Section 4.2.2.1. (We experiment with other methods of generating the mass in Section 4.3.3). We compare this to BG-MC and BG-HCC (the two variants of block Gibbs defined in Section 4.3.1). For all samplers, $\mathbf{\Lambda}$ is initialised to the identity matrix. We ran 100 iterations of burn-in, and then gathered the following 10000 samples.

We compare the samplers by effective sample size (ESS) which is defined as:

$$\text{ESS} = T\left(1+2\sum_{t=1}^{\infty}\rho(t)\right)^{-1}, \tag{4.27}$$

where T is the number of samples drawn and $\rho(t)$ is the autocorrelation at lag t. We used the initial monotone sequence estimator (Geyer, 1992) to estimate the autocorrelations. Table 4.2 shows the results. Times do not include the time taken to compute the index sets in block Gibbs, or to compute the mass matrix in HMC. When $\boldsymbol{G}$ is fixed as in this experiment, these times are negligible. (But if the sampler is to be part of a joint sampler for $(\boldsymbol{G},\mathbf{\Lambda})$, then they are significant, as discussed in Sections 4.3.3 and 4.3.4).

We do not have results for the HMC sampler in the Cholesky representation because it was extremely slow. We ran it on the first test case in Table 4.2, where $p = 10$, $\frac{N}{q} = 5$, and $s = 0.5$. It took 53 minutes to generate 10000 samples, resulting in an ESS/sec of 3.13. For comparison, HMC in the original space took around 20 seconds to generate 10000 samples from the same distribution.

There are missing entries for BG-MC at dimensionalities 75 and 100: we abandoned those tests because they were taking an extremely long time. We consider BG-MC to be impractical for high-dimensional problems. BG-HCC is more efficient than BG-MC in this scenario, and also when the sparsity is such that BG-MC has to work with a large number of maximal cliques.

Table 4.2 shows that BG-MC is best for low dimensional models, but when $p \geq 25$, HMC is significantly more efficient. The effect of the data size is different for block Gibbs and HMC. Block Gibbs tends to improve as data size decreases; HMC tends to improve with more data, we expect because the Gaussian approximation used when computing the mass matrix becomes more accurate. As the graph becomes sparser (s decreases), HMC improves because there are fewer variables to simulate. The block Gibbs methods tend to prefer either very sparse or very dense graphs, which is ex-

Table 4.2: Comparison by ESS/sec on synthetic data of HMC and two block Gibbs methods: BG-MC, where the covering set consists of all maximal cliques; and BG-HCC, our heuristic clique cover algorithm. Block Gibbs is best only for very low-dimensional models. HMC is orders of magnitude faster in higher dimensions. Numbers in brackets are standard errors computed over 10 runs.

Test		**BG-MC**		**BG-HCC**		**HMC**	
Dimension	p	ESS	ESS/sec	ESS	ESS/sec	ESS	ESS/sec
	10	7697 (1161)	**878 (136)**	7086 (1173)	529 (100)	**10000 (0)**	514 (10)
$N/q = 5$	25	4087 (1051)	39.9 (12.3)	1525 (625)	20.0 (8.2)	**10000 (0)**	**244 (4)**
$s = 0.5$	50	3037 (693)	1.59 (0.5)	521 (240)	1.26 (0.58)	**9977 (74)**	**61.8 (3.6)**
	75	–	–	188 (81)	0.151 (0.068)	**9999 (1)**	**16.6 (0.6)**
	100	–	–	150 (76)	0.0624 (0.0360)	**8836 (317)**	**3.62 (0.18)**
Data size	N/q	ESS	ESS/sec	ESS	ESS/sec	ESS	ESS/sec
	0.2	4435 (439)	26.3 (4.1)	1910 (425)	24.8 (5.3)	**7704 (431)**	**83.9 (5.5)**
$p = 25$	1	4060 (1158)	24.6 (8.4)	1905 (877)	26.2 (14.8)	**9990 (16)**	**235 (34)**
$s = 0.5$	5	3809 (1084)	23.1 (7.2)	1681 (640)	23.2 (9.9)	**10000 (0)**	**295 (4)**
	25	3534 (733)	20.8 (4.6)	1254 (453)	16.8 (6.2)	**9929 (212)**	**324 (10)**
	100	3020 (898)	18.8 (5.9)	1185 (601)	15.7 (8.4)	**9554 (1128)**	**314 (38)**
Sparsity	s	ESS	ESS/sec	ESS	ESS/sec	ESS	ESS/sec
	0.1	8221 (1376)	147 (34)	8221 (1431)	146 (35)	**10000 (0)**	**316 (19)**
$p = 25$	0.25	3177 (767)	33.7 (8.7)	2700 (763)	32.2 (9.5)	**10000 (0)**	**254 (6)**
$N/q = 5$	0.5	3089 (1423)	14.4 (6.8)	1108 (842)	12.6 (10.2)	**9995 (13)**	**239 (7)**
	0.75	7439 (776)	14.2 (1.7)	1933 (450)	27.1 (5.2)	**10000 (0)**	**214 (6)**
	0.9	9426 (179)	21.1 (12.7)	4995 (1221)	96.3 (34.4)	**10000 (0)**	**205 (21)**

Table 4.3: Comparison of methods for computing the HMC mass matrix. The preliminary sampling methods result in the best ESS/sec. In addition, the preliminary Wishart method computes the mass more rapidly than the preliminary GWishart, which may be important if the mass needs to be updated often. Numbers in brackets are standard errors computed over 10 runs.

	Identity	**Preliminary GWishart**	**Laplace**	**Preliminary Wishart**
Time to compute M (sec)	0 (0)	91.6 (13.3)	27.1 (23.1)	2.15 (0.04)
Sampling time (sec)	5560 (316)	21.7 (0.2)	18.3 (0.8)	21.8 (0.3)
ESS	2348 (726)	10000 (0)	669 (470)	10000 (0)
ESS/sec	0.425 (0.138)	461 (3)	37.0 (25.9)	460 (7)

pected because these cases will usually produce fewer cliques than a moderate level of sparsity.

In summary, we note the following key points from our tests.

1. For block Gibbs, using BG-HCC is preferable to BG-MC in high dimensions, or when the level of sparsity is unfavourable to BG-MC.

2. Except for low-dimensional problems, HMC performs significantly better than both block Gibbs methods.

4.3.3 Comparing Methods for Computing the Mass Matrix

We compared the methods of computing the mass matrix described in Section 4.2.2.1. We generated 10 runs of a single test case as in Section 4.3.2. The parameters were: $p = 25, \frac{N}{q} = 5, s = 0.5$. We sampled the distributions using HMC with each of the mass matrix methods. For the two methods requiring preliminary samples, we drew 20000 points. For the Laplace approximation, we found the mode numerically by gradient ascent. The results are shown in Table 4.3.

The identity matrix mass makes HMC highly inefficient. The long sampling time with the identity matrix mass is the result of small step sizes and a comparatively large trajectory length. As in Section 4.3.2, these parameters were chosen manually using preliminary runs, attempting to maximise ESS/sec. We found that small step sizes were

necessary to avoid a high rejection rate – but the trajectory length cannot be reduced too much or the chain will not move far and the ESS will be reduced.

The Laplace approximation also performed quite poorly. In terms of ESS/sec, a preliminary sampling run from the GWishart, and from the Wishart (followed by conditioning on missing edges), gave similar results. However, the preliminary run is considerably more expensive with the GWishart. If the HMC sampler is to be embedded in a joint sampler for $(\boldsymbol{G}, \boldsymbol{\Lambda})$, the preliminary GWishart run needs to be repeated each time the graph changes, which is clearly impractical. But the precision $\boldsymbol{K}$ computed from the Wishart samples remains valid as the graph changes: the new mass can be obtained simply by removing those rows and columns from $\boldsymbol{K}$ that correspond to missing edges.

4.3.4 Comparing the Bayesian GGM and the Graphical Lasso

Having demonstrated the advantages of using an HMC sampler over block Gibbs for the GWishart, we now employ this method to sample from the sparse Bayesian GGM described in Section 4.1.3. We embed our HMC sampler within the WL sampler – see Algorithm 6. We compare this Bayesian model with the optimisation-based graphical lasso (GLASSO) described in Section 2.1.2.

4.3.4.1 Time-Constrained Performance Comparison

We begin with an experiment designed to compare the performance of the sparse GGM against GLASSO when both methods are constrained to the same time budget. We use the financial data set described in Section 2.5.1, but here we select 35 stocks and 1000 days covering the period April 2005 to March 2009. We use the first 500 days as the training data, and the remaining 500 as the test set. We preprocess the data set by subtracting the mean of the training data, and scale such that the empirical precision of the training set has all ones along the main diagonal.

We apply the Bayesian GGM and the graphical lasso to this data. To set the penalty parameter γ for GLASSO, we do 5-fold cross-validation over 100 equally spaced values. We train a model with the best-performing γ and compute the log likelihood of the test data.

For the Bayesian GGM, we use the WL method (with embedded HMC) to sample $(\boldsymbol{G}, \boldsymbol{\Lambda})$ jointly. We choose the graph prior such that each edge is independently

Bernoulli-distributed:

$$p(\boldsymbol{G}|s) = \prod_{i<j} s^{G_{ij}}(1-s)^{(1-G_{ij})}. \tag{4.28}$$

The parameter s controls the sparsity of the graph, and so is analogous to the $L1$ penalty in GLASSO. But in the Bayesian model, there is no need to cross-validate s: we choose a $Beta(1,1)$ distribution as the prior for s (since it is conjugate to the Bernoulli distribution), and introduce a sampling step to resample s given the current $\boldsymbol{G}$. In this experiment, we set the parameters $(b, \boldsymbol{D})$ for the GWishart prior manually, but we found that varying these parameters had little effect on the generalisation performance of the model. We report results for the prior $W_G(b, d\boldsymbol{I}_p)$, where the degrees of freedom $d \equiv b + p - 1 = p + 10$. To initialise the chain, we set $\boldsymbol{\Lambda}$ to the empirical precision computed on the training data.

The WL sampler requires samples from both the GWishart prior and posterior, and its performance depends strongly on the efficiency of this component. We tested both BG-HCC and HMC for this task. BG-MC is far too slow to use within the WL sampler: when we tried it, WL could not complete a single iteration in the time it took to complete cross-validation on the graphical lasso. Most of the time is spent recomputing the maximal cliques each time the graph changes; but there are many maximal cliques, so lots of time is spent sampling too. For HMC, our results do not include the time taken to find good settings of the step parameters (α, β). We set these values by adjusting them such that the acceptance rate on preliminary runs is around 65% – as suggested by Neal (2010) – and such that the ESS is high. In practice, little adjustment is needed because our choice of mass matrix means that similar step parameters can be used for a wide variety of GWishart distributions.

To make a comparison with graphical lasso, we approximate the expected test log likelihood

$$\mathbb{E}\left[\log p\left(\boldsymbol{Y}_{(test)}|\boldsymbol{\Lambda}\right)|\boldsymbol{Y}_{(train)}\right] \approx \frac{1}{T}\sum_{t=1}^{T} \log p\left(\boldsymbol{Y}_{(test)}|\boldsymbol{\Lambda}^{(t)}\right), \tag{4.29}$$

using the samples $\left\{\boldsymbol{\Lambda}^{(t)}\right\}_{t=1}^{T}$ drawn from the posterior $p\left(\boldsymbol{\Lambda}|\boldsymbol{Y}_{(train)}\right)$. We compute this expectation after including each additional sample; Figure 4.1 shows how the value evolves over time for both the HMC and BG-HCC versions of the joint sampler for a typical run. The samplers do not quite agree because neither has yet converged. But right from the start, they perform significantly better than graphical lasso, which takes a few minutes to register its result (including the time required for cross-validation). At the time GLASSO finishes, the test log likelihood scores are:

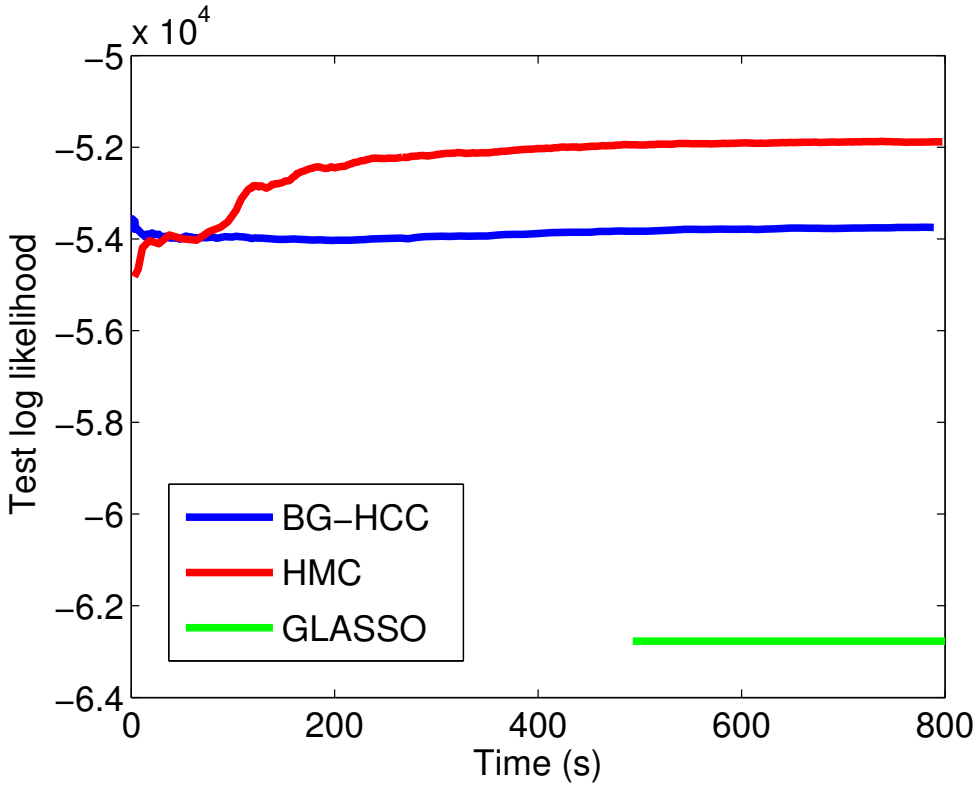


Figure 4.1: Estimated test log likelihoods over time. The Bayesian methods result in significantly higher test log likelihoods – good estimates of which are obtained after only a few samples.

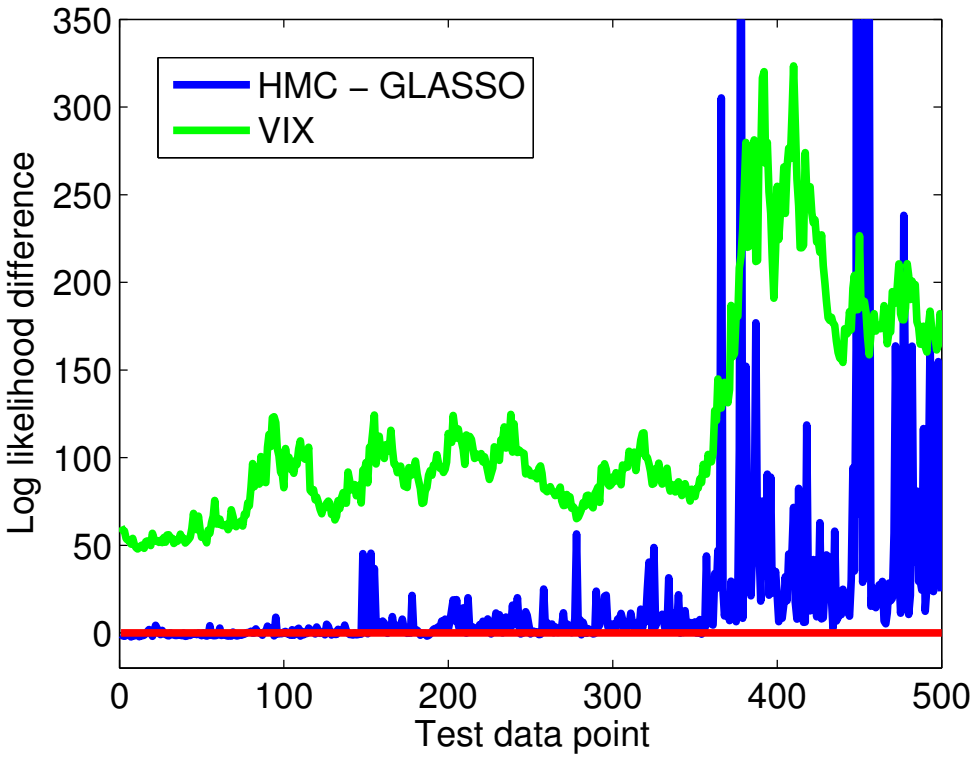


Figure 4.2: For each point in the test data set, the difference in log likelihood between the Bayesian GGM and graphical lasso is plotted in blue. The VIX volatility index, arbitrarily scaled to fit, is in green. The Bayesian method performs better over the whole test set, but tends to strongly outperform graphical lasso when market volatility is high.

- HMC $= (-5.19 \pm 0.33) \times 10^4$;
- BG-HCC $= (-5.38 \pm 0.35) \times 10^4$;
- GLASSO $= (-6.28 \pm 0.51) \times 10^4$.

For comparison, the test log likelihood under a Gaussian model with the empirical precision of the training data is $(-7.20 \pm 0.70) \times 10^4$.

Figure 4.2 offers an explanation for the better performance of the Bayesian model. It plots the difference in log likelihood between the HMC and GLASSO methods for each point in the test set. The VIX index – which is a measure of market volatility – is overlaid. The graph shows that the Bayesian GGM fares particularly well against the graphical lasso when the market is more volatile. This seems likely to be a result of the Bayesian methods making use of the full posterior, rather than just using the MAP solution. If the graph were fully-connected, the prior on $\mathbf{\Lambda}$ would be Wishart, and so the marginal distribution of $\boldsymbol{y}$ would be a multivariate Student's t. We would

therefore expect the Bayesian GGM model to be more robust to extreme values than the Gaussian model trained by GLASSO – and such values are more likely with increased market volatility.

4.3.4.2 Computational Cost Comparison

In Section 4.3.4.1, we saw that the sparse GGM model can perform better than the graphical lasso, even when both are constrained to the same time budget. But this result was observed on a single data set, with a fixed number of dimensions. It is natural to question how this result scales with the dimensionality of the data. We study this issue in the current section. For the GGM, we sample the GWishart with HMC and compute the mass matrix using a preliminary sampling run from the associated Wishart distribution; see Section 4.2.2.1.

We begin with some theoretical observations. For the graphical lasso, there are many algorithms in the literature, and these may scale differently. As a representative example, Friedman et al. (2008) state that for their algorithm, the "computation time is $O\left(p^3\right)$ for dense problems, and considerably less than that for sparse problems". For the GGM, we observe empirically that the bottleneck in high dimensions is the inversion of the mass matrix, which has cost $O\left(p^6\right)$. During each iteration of the GGM sampler, the WL method proposes to flip each of the $O\left(p^2\right)$ edges while holding the others fixed. In our implementation, we select and invert a new mass matrix for HMC each time the graph changes, so the worst case cost for the sparse GGM sampler is $O\left(p^8\right)$. In practice, some edges remain unmodified, and a sparse graph means that the mass matrix to be inverted typically has dimension much less than p.

Since the the graphical lasso computation time scales better than the GGM in theory, we would expect to see the result of Section 4.3.4.1 – that the GGM outperformed GLASSO for the same time budget – reversed on a higher-dimensional problem. We now perform an experiment to verify this, and to find out which method is preferred for different problem sizes.

We again use the 1000 points of FTSE data as in Section 4.3.4.1. From this, we create eight data sets of dimension $10, 20, \ldots, 80$: we randomly permute the dimensions, and take the first 10 for the first data set, the first 20 for the second, and so on. We use the same 500 points of test data as in the previous experiment. For the training data, we increase the number of data points linearly with the dimensionality p: of the 500 data points used in Section 4.3.4.1, we use the fraction $\frac{p}{80}$ immediately prior to the test data in temporal order. The data are preprocessed as before.

For GLASSO, we cross-validate as in the previous section. For the GGM, we use independent Bernoulli priors with $s = 0.5$ on the graph edges; see Equation (4.28). We choose the GWishart prior $W_G\left(b, \frac{7}{4}d\boldsymbol{I}\right)$, where $b = 10$ and $d = b + p - 1$ is the degrees of freedom. The number $\frac{7}{4}$ is chosen to make the test log likelihoods of GLASSO and the GGM similar for illustrative purposes. As we saw in the previous section, the GGM has an advantage over GLASSO where the test data are distributed differently than the training data, so by decreasing (increasing) the mean of the prior precision, we can make the GGM perform better (worse) relative to GLASSO. But in this section, we are interested primarily in comparing the computation time of the two methods.

The GGM sampler runs faster as the graph sparsity increases. If we were to initialise the chain with a dense graph, the early iterations would run slowly. Since our Bernoulli prior on the edges has $s = 0.5$, for a fair comparison with GLASSO we choose an initial precision in which half of the off-diagonal elements are zero. To construct this initial value, we take the empirical precision of the training data, then set the half of the off-diagonal elements with smallest absolute value to zero (and verify that the resulting matrix remains positive-definite).

For the HMC step size and trajectory length parameters α and β (see Section 4.2.2), we do no preliminary runs to adapt these parameters to each data set: we use the values $(\alpha = 0.02, \beta = 0.5)$ when sampling the prior, and $(\alpha = 0.2, \beta = 0.5)$ when sampling the posterior, for all eight data sets. These produce reasonable acceptance rates (at least 70%) in all cases.

As in the previous experiment, we estimate the test log likelihood from the GGM samples using Equation (4.29), and compute this after each additional sample. Figure 4.3 shows how this value evolves over time for each of the eight data sets, and facilitates a comparison with GLASSO. We reiterate that the test log likelihood scores are of less interest here than the time taken to arrive at these results. The figure shows that when the number of dimensions is less than or equal to 30 (the first three graphs), the GGM estimate of test log likelihood has stabilised long before GLASSO returns a result. For dimensions 40 to 60, the GGM estimate appears to be approaching stability around the time GLASSO returns a result. With 70 and 80 dimensions, the GGM sampler has only generated a few samples when GLASSO has finished.

In conclusion, focussing on speed alone, the GGM appears to be the better choice when the number of dimensions is 30 or less. Up to 60 dimensions, either method is a reasonable choice. If there are more dimensions than 60, GLASSO seems the best option. This should be seen as a rough guide: we note that with different choices of

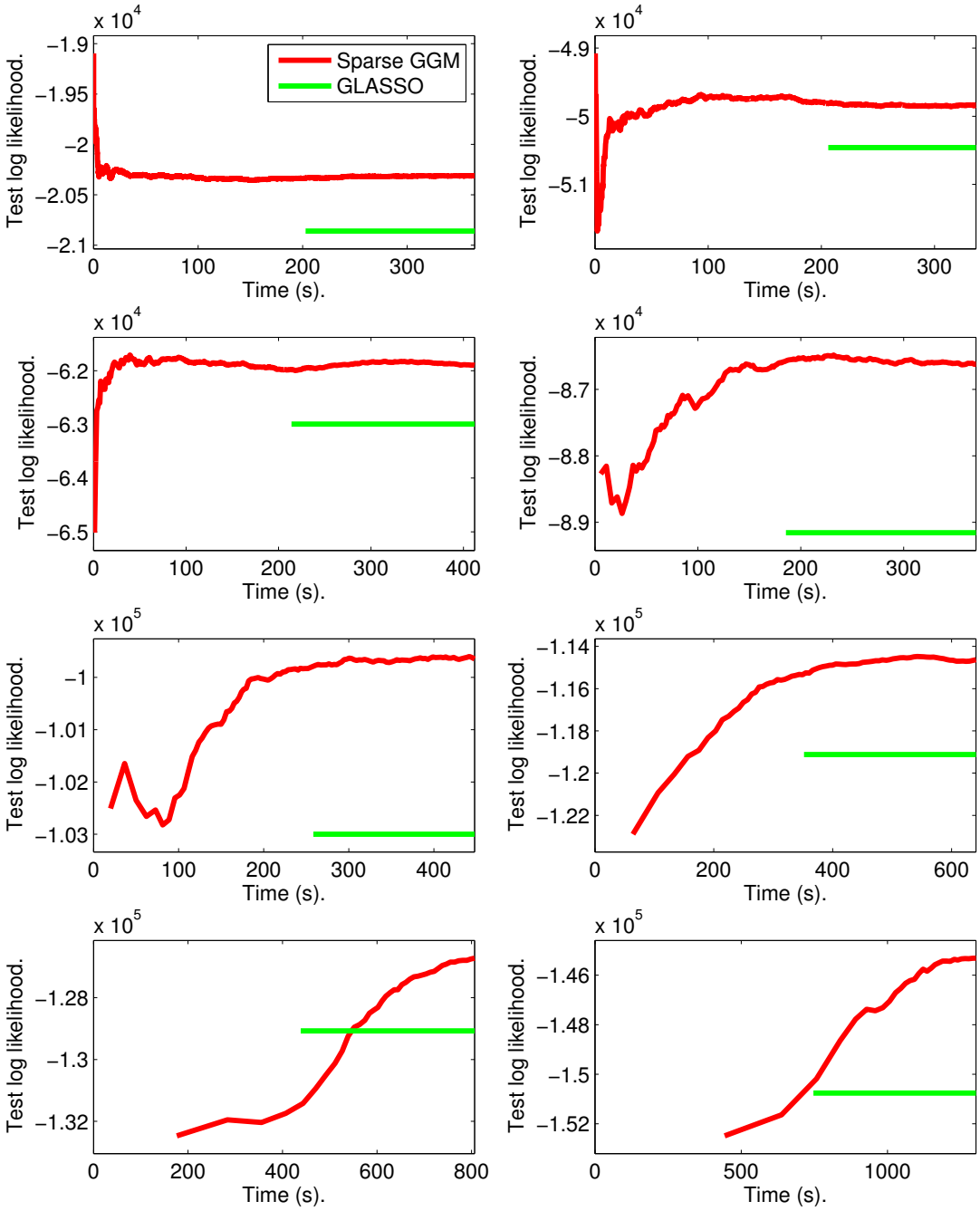


Figure 4.3: Comparison of test log likelihoods over time between the sparse Bayesian GGM and the graphical lasso. Left to right, top to bottom, the graphs compare these methods on data sets of dimension $10, 20, ..., 80$. The test log likelihood from the GGM has stabilised before GLASSO produces a result in the cases of 10 to 30 dimensions. For 40 to 60 dimensions, the GGM estimate is nearing convergence, and may still be useful in preference to GLASSO in a practical situation. In 70 dimensions and above, the GGM sampler has only managed a few iterations before GLASSO returns its result, and so GLASSO is likely preferable to the GGM here.

parameters, a different data set, or different implementations of the two methods, the relative performance may be different. Additionally, there are many factors to consider other than computational performance when choosing a data analysis method.

4.4 Conclusions and Future Work

In this chapter, we developed a Hamiltonian Monte Carlo sampler for the GWishart distribution and demonstrated its increased efficiency over the block Gibbs sampler. Our HMC method, together with our computation of the mass matrix from preliminary Wishart samples, is suitable for embedding into a joint sampler of the graph and precision in a sparse Gaussian model. We also described a way to choose the covering set in the block Gibbs sampler that reduced run time and made it more practical to use block Gibbs within a joint sampler.

We then compared a sparse Bayesian GGM model based on the GWishart distribution with the graphical lasso estimator on a real-world data set. We found that the Bayesian model performed better in terms of test log likelihood, even when the models were constrained to the same time budget. The better performance of the Bayesian model appeared due to its use of the full posterior – as opposed to the graphical lasso's MAP solution. We went on to compare the efficiency of the two methods over a range of data dimensions. We found that the GGM outperformed GLASSO up to 30 dimensions, and remained viable up to about 60 dimensions on our data set. Beyond that, GLASSO was the more efficient method. We interpret this result as a rough guide to making a choice between the two methods: we expect the relative efficiency to depend on various factors incuding the data set, the chosen parameters of each method, and their implementations. Furthermore, there are factors other than efficiency to consider.

If inference in the sparse Bayesian GGM can be performed adequately quickly, then it may be a better choice of model in some situations than the graphical lasso. We discussed in Section 4.1.3.2 that the explicit graph prior in the Bayesian model allows certain types of prior information to be encoded more precisely than in the graphical lasso. And of course, GLASSO provides only the MAP solution, whereas the Bayesian model provides a full posterior. This may be important for some applications; for example, when attempting to reconstruct a metabolic reaction network, as in the work by Krumsiek et al. (2011), one may wish to obtain a measure of confidence in the existence of an edge in the graph. Sampling the GGM, we can count the samples in which the edge is present; GLASSO gives only a binary answer.

Future work could involve an investigation of better ways to select the step size and trajectory length distributions. This may involve an attempt to understand how the optimal choices are affected by the parameters of the GWishart distribution. We could also investigate how to select a mass matrix that is more effective or more efficient to compute. We discuss additional future work in the following subsections.

4.4.1 Comparing the WL and BDMCMC Samplers

It would be interesting to compare the WL sampler (based on reversible jump MCMC) with the BDMCMC sampler (based on birth-death MCMC) – see Section 4.1.3. In their BDMCMC work, Mohammadi and Wit (2012) approximate a ratio of GWishart normalisation constants by one. Wang and Li (2012) use a double Metropolis-Hastings method to compute this ratio. It would interesting to see how the approximation affects the trade-off between efficiency and accuracy – both in WL and BDMCMC. We note also that it is the use of double MH in the WL sampler that necessitates drawing samples from the GWishart prior. Our HMC sampler for the GWishart is at its least efficient when sampling from the prior (see Section 4.3.2), so removing the need to sample the prior should further improve efficiency of inference in the GGM.

4.4.2 Sampling the GWishart Hyperparameters

In Section 4.3.4, we set the GWishart prior parameters b and $\boldsymbol{D}$ manually. It would be preferable to put a prior on these parameters and sample them conditioned on $\boldsymbol{\Lambda}$ (in a similar way to our treatment of the graph-sparsity hyperparameter s). However, the prior and posterior GWishart distributions that we sample from in Algorithm 6 depend on these hyperparameters, and the mass matrix should change accordingly. But computing the mass matrix is expensive: if we compute the mass by taking preliminary samples from $W(b, \boldsymbol{D})$ (see Section 4.2.2.1), then we would need to draw new samples each time b or $\boldsymbol{D}$ changes.

If the changes to the hyperparameters between each iteration are small, perhaps it will suffice to update the mass infrequently, or even to keep it fixed. Alternatively, an adaptive sampling scheme such as Riemann manifold HMC (RMHMC) (Girolami and Calderhead, 2011) would not require the mass to be manually selected. In RMHMC, the mass matrix is replaced by a metric tensor that is a function of the HMC position variable – $\boldsymbol{\Lambda}$ in our case. This allows the HMC sampler to adapt to the local geometry. For the Bayesian GGM model, the metric tensor would depend on the hyperparameters.

The metric tensor must be computed at each iteration of RMHMC, so making it a function of the hyperparameters should not incur additional cost. But RMHMC is already costly: the inverse of the metric tensor must be evaluated at each iteration, which grows with the cube of the number of parameters. In the GGM, the number of parameters is $O(p^2)$, and so each iteration of RMHMC would be $O(p^6)$ – which will quickly become prohibitive as p grows[2].

[2] We already invert the mass matrix each time the graph changes in our use of HMC within the WL algorithm, which is also $O(p^6)$. But RMHMC inverts the metric tensor at each integration step of each HMC proposal. Since we require multiple HMC runs for each potential change to the graph, using RMHMC would be more expensive.

Chapter 5

Conclusions

Here, we summarise our contributions, and make some concluding remarks.

5.1 Contributions

The key contributions, summarised here by chapter, are as follows.

Chapter 2

- We introduced the sparse latent inverse covariance estimator (SLICE) – an extension of the graphical lasso to incorporate latent variables – and described an EM method to compute it.
- On a real-world financial data set, we demonstrated that SLICE learned a more parsimonious representation of the data than graphical lasso.
- We extended the SLICE model to a mixture of experts named MSLICE, and extended the EM algorithm to learn its parameters.
- We demonstrated that the learned MSLICE mixture components could be interpreted in terms of the side information, and that the learned precision matrices were also somewhat interpretable. Thus, the method may be useful for knowledge discovery.
- To handle non-Gaussian data, we augmented SLICE and MSLICE with Gaussianising functions, resulting in the Gaussian copula model CopSLICE and the mixture of Gaussian copula experts CopMSLICE. We showed how to learn the parameters of these models.

- We demonstrated on financial data that the copula models improved on the Gaussian models in terms of log likelihoods on test data.

Chapter 3

- We introduced the IO-HMM models TGLASSO and CopTGLASSO – extensions of the MGLASSO and CopMGLASSO models of the previous chapter.
- We demonstrated that these models outperformed existing baselines, including an IO-HMM with factored emissions.
- The latent states and Gaussian structures learned by these models were interpretable, backing up our conclusion from Chapter 2 that our models may be useful in knowledge discovery.
- In our experiments, the temporal models brought little benefit over their non-temporal counterparts, especially in the presence of side information. It appears that when the side information is strongly predictive of the latent state, the temporal connections are unnecessary, and only increase the computation time.

Chapter 4

- We developed an HMC sampler for the GWishart distribution, and demonstrated its increased efficiency over the state-of-the-art block Gibbs sampler in most scenarios.
- We demonstrated that the HMC sampler is suitable for embedding into a joint Gaussian graphical model (GGM) in which the graph and precision matrix are to be sampled jointly.
- We described a way to choose the covering set in the block Gibbs sampler to reduce run time and make block Gibbs more practical when used within a GGM joint sampler.
- We compared a sparse Bayesian GGM with the (non-Bayesian) graphical lasso on a real-world data set, and found that the Bayesian model outperformed the graphical lasso even when both methods were constrained to the same time budget.

5.2 Future Directions

Here, we discuss some preliminary investigations and potential future research directions aimed at unifying some of the methods and ideas from Chapters 2, 3, and 4.

5.2.1 Latent-Variable Extensions of the Bayesian GGM

We would like to extend the sparse Bayesian GGM described in Section 4.1.3 by incorporating latent variables, similarly to how we extended graphical lasso to SLICE – see Section 2.2.1. Consider the following model:

$$\boldsymbol{G} \sim P(\boldsymbol{G}) \tag{5.1}$$

$$\boldsymbol{\Lambda} \sim W_G(b, \boldsymbol{D}) \tag{5.2}$$

$$\boldsymbol{u} \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}^{-1}\right), \tag{5.3}$$

where $\boldsymbol{u} = \left(\boldsymbol{y}^T, \boldsymbol{z}^T\right)^T$, and $\boldsymbol{z}$ is a vector of latent variables. Given data matrix $\boldsymbol{Y}$, how can we sample from the posterior $P(\boldsymbol{G}, \boldsymbol{\Lambda}, \boldsymbol{Z}|\boldsymbol{Y})$? Here, we discuss our investigation of this problem to date; but it remains an open issue.

The naive approach would be to extend the WL algorithm with a sampling step for $P(\boldsymbol{Z}|\boldsymbol{\Lambda}, \boldsymbol{Y})$, and sample $\boldsymbol{G}$ and $\boldsymbol{\Lambda}$ as before. The problem with this is that $\boldsymbol{Z}$ and $\boldsymbol{\Lambda}$ are tightly coupled: see the discussion in Murray and Adams (2010), for example. Progress of the Markov chain through the joint space of $(\boldsymbol{\Lambda}, \boldsymbol{Z})$ may be slow if $P(\boldsymbol{Z}|\boldsymbol{\Lambda}, \boldsymbol{Y})$ and $P(\boldsymbol{\Lambda}|\boldsymbol{G}, \boldsymbol{Z}, \boldsymbol{Y})$ are sampled alternately.

If samples of $\boldsymbol{Z}$ are not required, we may instead integrate out the latent variables, and run HMC to sample the joint precision $\boldsymbol{\Lambda}$ from $P(\boldsymbol{\Lambda}|\boldsymbol{G}, \boldsymbol{Y})$. We implemented this sampler, but we found that mixing was very slow. This seems to be because the latent variables couple the elements of $\boldsymbol{\Lambda}$ such that the local geometry varies considerably over the space of $\boldsymbol{\Lambda}$, making it difficult to find a suitable mass matrix. We illustrate this by running the following experiment. We generated a random graph on 25 nodes by drawing from the distribution (4.28) with $s = 0.5$; drew a ground truth precision by sampling the GWishart prior; then drew 1000 samples from the Gaussian with this precision. We retained only the first 20 dimensions from these samples, considering the remaining 5 dimensions to be latent variables. These samples constituted the 1000×20 data matrix $\boldsymbol{Y}$. We then ran the naive sampler to draw 200000 samples from the posterior $p(\boldsymbol{\Lambda}|\boldsymbol{G}, \boldsymbol{Y})$. In Figure 5.1, we plot the sampled values for 3 pairs of elements of $\boldsymbol{\Lambda}$, chosen manually for illustrative purposes. In each subfigure, we can see long,

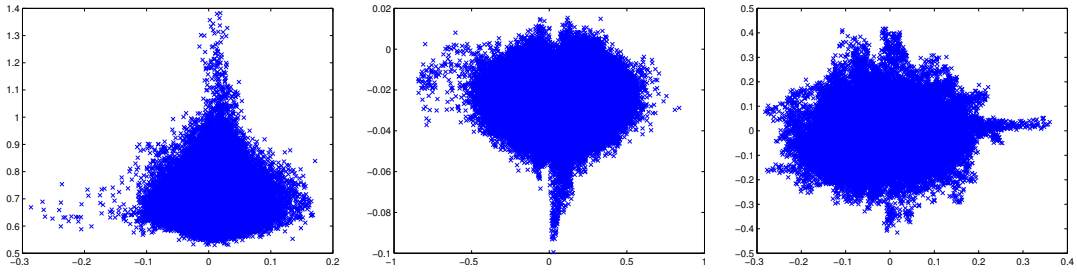


Figure 5.1: Illustration of variation in the local geometry in $\mathbf{\Lambda}$-space in the presence of latent variables. Each subfigure plots sampled values of a pair of elements of $\mathbf{\Lambda}$. In each plot, we see variations in local geometry that may be problematic for HMC. For example, a mass matrix that performs well in one of the large central regions may not perform well in one of the long thin regions.

narrow regions in the distribution. The optimal mass matrix for an HMC sampler may be very different in these regions than in the larger central regions.

RMHMC would allow the mass to vary over the space of $\mathbf{\Lambda}$, which may allow the Markov chain to explore more rapidly. But, as discussed in Section 4.4.2, RMHMC is $O(p^6)$, where p here is the sum of the observed and latent dimensions.

Whichever method is used to sample from $p(\mathbf{\Lambda}|\boldsymbol{G},\boldsymbol{Y})$, there is an additional problem: the distribution $P(\boldsymbol{G},\mathbf{\Lambda}|\boldsymbol{Y})$ is, in general, multimodal. We investigated tempered transitions (Neal, 1996) as a way of solving this problem. The tempered transitions method requires that we specify a sequence of distributions interpolating between the distribution from which we wish to sample, and some distribution in which sampling is easier. The Markov chain moves through the sequence until the simpler distribution is reached, and then moves back to the original distribution. The idea is that large moves can be made in the simpler distribution, making the Markov chain more likely to move between modes. For the GGM, we observed that, when the graph is fully connected, a Metropolis-Hastings sampler can rapidly explore the distribution of $\mathbf{\Lambda}$ given $\mathbf{\Sigma}_{yy}$ – because with fixed $\mathbf{\Sigma}_{yy}$, the likelihood of any two values of $\mathbf{\Lambda}$ is the same. Our idea was to exploit this by using tempered transitions with J interpolating distributions for the graph prior defined in Equation (4.28). We defined a sequence of J interpolating distributions by changing the sparsity parameter s according to the sequence $S = (s_0, s_1, ..., s_J)$ where $s_i < s_k$ if $i < k$, s_0 is the sparsity of the original graph prior, and $s_J = 1$. This sequence was defined to make the graph progressively more dense. When the graph was fully dense, we sampled $\mathbf{\Lambda}|\mathbf{\Sigma}_{yy}$ using Metropolis-Hastings.

However, we were not able to make this method work well. We found the following

problems.

1. The number of interpolating distributions J had to be large to avoid a high rejection rate. For example, with $p = 15$, we needed J to be around 500. This made the tempered transition steps slow.

2. Given J, we found it difficult to find a sequence S that kept the rejection rate low.

3. The method scales poorly with dimension: as p increases, the number of possible edges increases as p^2, and it is more likely that at least one will be missing when s_J is reached, which results in a rejection of the tempered transition.

We also tried the Wang-Landau adaptive simulated tempering algorithm (AST) (Wang and Landau, 2001) and a coupled AST algorithm (Salakhutdinov, 2010), but we still found speed and scaling to be problematic in each case.

It may be that with a different application of these methods – such as a different choice of sequence S – they may help with the multimodality problem. Also, as explained above, we do not yet have a good sampler for $P(\boldsymbol{\Lambda}|\boldsymbol{G}, \boldsymbol{Y})$ when latent variables are present. This sampler is used within the tempering methods, so perhaps they would perform better if we had such a sampler. The sampler may also make tempered transitions useful with $s_J < 1$. At present, our MH sampler for $\boldsymbol{\Lambda}|\boldsymbol{\Sigma}_{yy}$ relies on the graph being fully connected; if it is not, defining a proposal that keeps $\boldsymbol{\Sigma}_{yy}$ constant (or nearly constant) is more difficult.

5.2.2 Conditional Mixtures and Copulas

Another possible direction is to extend the Bayesian GGM model in a similar way to our extensions of SLICE in Chapters 2 and 3; that is, by introducing side information, multiple mixture components, non-Gaussian marginals, and temporal dependencies. Some of these extensions already exist in the literature, and we briefly review them now. Future work may combine these approaches – with some modifications – into a Bayesian model somewhat akin to CopMSLICE without the latent variables.

Rodriguez et al. (2011) study a Dirichlet process mixture of GGMs, a more general species sampling model, and an infinite hidden Markov GGM. They limit the models to decomposable graphs, which simplifies inference because the GWishart normalisation constant can be computed in closed form (Roverato, 2002). The limitation to decomposable graphs could probably be dropped at a cost of increased computation time. The normalisation constants would no longer be computable in closed form –

but we could probably use the WL sampler with HMC for the GWishart to sample within each component of the mixture.

To extend the mixture to a mixture of experts, we might incorporate a technique from Rasmussen and Ghahramani (2001). They modify the Dirichlet process to make it input-dependent by using a kernel-based local estimate of the occupation number of each component. In their approach, the experts are Gaussian processes, but it should be possible to make them sparse GGMs.

Dobra and Lenkoski (2011) introduce a copula extension of the Bayesian GGM. Their model has a graph $\boldsymbol{G}$ and precision $\boldsymbol{\Lambda}$ drawn from (4.10) and (4.11) respectively. The rest of the model is:

$$\boldsymbol{y} \sim \mathcal{N}\left(\mathbf{0}, \boldsymbol{\Lambda}^{-1}\right), \tag{5.4}$$

$$y'_v = y_v \left(\boldsymbol{\Lambda}^{-1}\right)_{vv}^{-\frac{1}{2}}, \tag{5.5}$$

$$\widetilde{y}_v = F_v^{-1}\left(\Phi\left(y'_v\right)\right), \tag{5.6}$$

where F_v is the marginal cdf of $\widetilde{y}_v$, and Φ is the standard Gaussian cdf. They make no assumptions about the form of F_v. Instead, they follow Hoff (2007), treating the F_v as nuisance parameters. Inference is performed by mapping a data matrix $\widetilde{\boldsymbol{Y}}$ into the set of $\boldsymbol{Y}'$ consistent with the non-decreasing property of every F_v. Resampling each Y'_{nv} conditioned on all other variables becomes a draw from a truncated Gaussian. They resample $\boldsymbol{G}$ using reversible jump MCMC, and resample $\boldsymbol{\Lambda}$ by sequentially updating the elements of its Cholesky decomposition by Metropolis-Hastings. Samplers developed since – such as the WL algorithm with our HMC sampler for the GWishart – would probably improve the efficiency of this step.

5.3 Concluding Remarks

Structure learning is an important problem that spans multiple tasks across multiple domains. Discovering relationships between variables may be posed as a structure-learning problem. If predictions are required, the choice of model structure may have a strong influence on the accuracy of those predictions, and on the ability of the model to use the available data efficiently. But sufficient prior knowledge may not be available to make clear the optimal choice of structure. In such cases, it would be useful to learn a model structure from data.

There has been much recent progress on learning the structure of Gaussian models. One thread of this research is focussed on optimisation-based methods, such as the graphical lasso; see Section 2.1.2. Another thread has focussed on Bayesian models built around the GWishart distribution; see Section 4.1.2. In this thesis, we have contributed to the field of Gaussian structure learning, addressing problems in each of these threads of research.

We began with the observation that, in many practical problems, latent variables confound the relationships between the observed variables, which may lead to a dense marginal structure among those observed variables. However, the relationships in the joint space of observed and latent variables may be more sparse. This motivated our introduction of latent variables into a Gaussian model, imposing sparsity in the joint space to learn the structure, resulting in our SLICE method; see Section 2.2.1. As we postulated, our experiments indicate that SLICE can learn more parsimonious models than the graphical lasso. This could be useful for knowledge discovery or decision support. For example, if SLICE is trained on financial returns data as in Section 2.5, the latent variables may capture intra-sector or market-wide effects; it may then be easier to spot residual couplings that are not explained by the latent factors. However, multimodality could present a problem: the EM algorithm for SLICE can result in very different structures depending on the initial conditions. If SLICE is used for knowledge discovery, some prior knowledge of the desired structure or the latent variables present may be required: the prior knowledge may be encoded into the $L1$ penalties to lessen the problem of multiple modes. Without sufficient prior knowledge, SLICE may result in an undesirable structure. Future work may attempt to address this problem. Alternatively, the sparse Bayesian models discussed in Chapter 4 result in a full posterior distribution, and so, in principle, each mode could be explored by sampling. However, as discussed in Section 5.2.1, we do not yet have a sampler for a sparse latent Gaussian model that mixes sufficiently quickly for practical use.

We noted that, in some scenarios, the structure underlying the data may vary between groups of data points. For example, financial assets' prices may become more strongly correlated under extreme market conditions (Preis et al., 2012). This motivated the MSLICE model, our generalisation of SLICE with multiple components and side information. In our experiments with financial data, we showed that the components could be interpreted in terms of the side information – a volatility index in that case. In some applications – including the modelling of financial returns – the number of components will not be known *a priori*. We set this parameter by cross-validation

in MSLICE, but it would be preferable to learn it simultaneously with the component structures. Although we studied a Bayesian model with just a single-component, the extension to a mixture of experts with a variable number of components may be easier in that case, given the existing work on Dirichlet process mixtures; see the discussion in Section 5.2.2.

Incorporating side information was particularly important in our experiments: it led to greater predictive accuracy, and the selected features could be understood intuitively. Furthermore, our results suggest that in cases where rich side information is available, an input-output HMM may not perform any better than a mixture of experts. This is good news because it means that practitioners may get good results while working with a simple model that is fast to train. However, in cases where side information is limited or unavailable, a temporal model may still be the better choice. Extending MSLICE to a matrix-variate model akin to those of Zhang and Schneider (2010) or Kalaitzis et al. (2013) may find a fruitful middle ground: a matrix-variate model would capture correlations between the variables at a fixed time, and also correlations between different time points. Such an extension may require less computation time than our IO-HMM models, and would naturally accommodate a higher-order Markov chain.

We consider the modification of our models to allow a precision matrix to depend directly on the side information to be an important topic of future work. A mixture of experts like MSLICE assumes that the data points fall into groups, with each group possessing a different underlying structure. But in practice, there may be small differences within each group. For example, we saw in our experiment in Section 2.5.3 that MSLICE learned to assign one component to each of the low, medium, and high-volatility market states. But within, say, a low-volatility period, two companies may begin a business arrangement that changes their share price correlation. If the precision matrices were to depend on the covariates (and the new business arrangement were included in the covariate vector), this change in the low-volatility structure might be captured. The key barriers to introducing dependence of the precision matrices on the covariates – as discussed in Section 2.6.1 – are ensuring that the precision matrices are both sparse and positive-definite for any value of the covariate vector, and avoiding the computation of a determinant for every point in the data set.

Since many real-world data – including financial data – are non-Gaussian, we augmented the components of MSLICE with Gaussianising functions, resulting in a mixture of Gaussian copulas that we named CopMSLICE. As expected, this improved per-

formance in our experiments with financial data, albeit with a trade-off in computation time. Non-Gaussianity in the marginals broadens the applicability of the model, but CopMSLICE still assumes that the dependence structures can be accurately modelled by multivariate Gaussians. Learning non-Gaussian dependence structures is beyond the scope of the methods discussed in this thesis, all of which rest on the assumption that edges in the Markov random field of a model (or component) correspond to non-zero entries in its precision matrix – which is true for the Gaussian, but not true in general.

For sparse Gaussian models based on the GWishart distribution, sampling from the GWishart is often the bottleneck. We addressed this problem first by modifying the state-of-the-art block Gibbs sampler, and then by developing a Hamiltonian Monte Carlo (HMC) sampler for this distribution. We demonstrated that the HMC sampler is much more efficient than block Gibbs under most conditions. This increase in the efficiency of inference could allow GWishart-based models to be applied to higher-dimensional problems. It may also help to make practical more complex models based on the GWishart distribution: we have already seen with CopMSLICE how the addition of latent variables, multiple components, side information, and copulas improved performance – no doubt the Bayesian models would benefit from similar extensions. (As noted above, however, we are not yet able to extend the HMC sampler to the latent-variable case.)

In our experiments, the sparse Bayesian GGM model outperformed the graphical lasso when both were constrained to the same time budget. Yet the graphical lasso has received far more study in the machine learning literature than sparse Bayesian GGMs. The Bayesian approach has some advantages. First, it results in a full posterior distribution, in contrast to MAP methods such as the graphical lasso which result in a single structure. This may be important if, for example, a measure of uncertainty is required in the presence or absence of an edge, or if it is useful to explore a range of the most likely structures. Further, the graph prior is explicit in the Bayesian GGM described in Section 4.1.3; in the optimisation methods, the prior is implicit in the choice of $L1$ penalties. An explicit prior is more flexible – allowing the practitioner to express correlations between edges, for example – and it may allow prior knowledge to be encoded more easily. For these reasons, we believe that the Bayesian approach to sparse Gaussian modelling is worthy of greater attention. Perhaps one reason it is less popular than the optimisation approach is that the Bayesian method is harder to use for the practitioner. Graphical lasso results in a single structure, while the Bayesian GGM

results in a set of samples from the posterior distribution. The Bayesian model would have to be appropriately packaged for greater ease of use by practitioners outside the machine learning field.

Appendices

Appendix A

CopMSLICE EM Derivation

Here, we derive the EM learning algorithm for the CopMSLICE model (of which SLICE and MSLICE are special cases). Let $\boldsymbol{x}$ denote a vector of covariates (or side information); $\widetilde{\boldsymbol{x}} = \left(1, \boldsymbol{x}^T\right)^T$ the covariate vector augmented with a unit element; $\widetilde{\boldsymbol{y}}$ a vector of observed variables; $\boldsymbol{y}$ a deterministic transformation of $\widetilde{\boldsymbol{y}}$; and $\boldsymbol{z}$ a vector of latent variables. The model is a conditional mixture with M components. Let $\boldsymbol{w}$ indicate which component of the mixture is responsible for generating $\widetilde{\boldsymbol{y}}$; that is, $w_m = 1$ for some m, and $w_k = 0$ for all $k \neq m$. We use $n = 1, \ldots, N$ to index data points, $c = 1, \ldots, C$ to index dimensions of $\boldsymbol{x}$, $v = 1, \ldots, V$ to index dimensions of $\boldsymbol{y}$ or $\widetilde{\boldsymbol{y}}$, $h = 1, \ldots, H$ to index dimensions of $\boldsymbol{z}$, and $m = 1, \ldots, M$ to index mixture components. To keep the notation as simple as possible, we assume that the latent vector $\boldsymbol{z}$ has the same dimensionality irrespective of the component that generates it. Extending the model to have a different number of latent factors in each component is straightforward. Define $\boldsymbol{u} = \left(\boldsymbol{y}^T, \boldsymbol{z}^T\right)^T$. The generative model is as follows:

$$\boldsymbol{w} \sim P(\boldsymbol{w}|\boldsymbol{x}; \boldsymbol{\Xi}), \tag{A.1}$$

$$(\boldsymbol{u}|w_m = 1) \sim \mathcal{N}\left(\boldsymbol{0}, \boldsymbol{\Lambda}_m^{-1}\right), \tag{A.2}$$

$$(\widetilde{y}_v|w_m = 1) = f_{mv}^{-1}(y_v) + \left(\boldsymbol{\Psi}_m^T\right)_{v:} \widetilde{\boldsymbol{x}}. \tag{A.3}$$

See Figure 2.5. $\boldsymbol{\Xi}$ parameterises the distribution of the mixing variable $\boldsymbol{w}$. Matrix $\boldsymbol{\Psi}_m$ parameterises the conditional mean of component m. The functions f_{mv} can be thought of as Gaussianising functions: we define $f_{mv}(\cdot) \equiv \Phi^{-1}\left(F_{mv}(\cdot\,; \boldsymbol{\zeta}_{mv})\right)$, where Φ is the Gaussian cdf, and F_{mv} is a univariate cdf parameterised by $\boldsymbol{\zeta}_{mv}$. It is required that F_{mv} – and therefore f_{mv} – is monotonically increasing and differentiable. We also define $\boldsymbol{\theta}_m^f = (\boldsymbol{\Psi}_m, \boldsymbol{\zeta}_m)$; that is, $\boldsymbol{\theta}_m^f$ parameterises the mapping $\widetilde{\boldsymbol{y}} \mapsto \boldsymbol{y}$ when component m is active.

From Equations (A.2) and (A.3), it is clear that each mixture component has a nonparanormal distribution (see Section 2.1.5) when the latent variables $\boldsymbol{z}$ are either conditioned on or marginalised. We can write

$$p\left(\widetilde{\boldsymbol{y}}|\boldsymbol{x}, w_m = 1, \boldsymbol{z}\right) = p\left(\boldsymbol{y}|w_m = 1, \boldsymbol{z}\right) \prod_{v=1}^{V} f'_{mv}\left(\widetilde{y}_v - \left(\boldsymbol{\Psi}_m^T\right)_{v:} \widetilde{\boldsymbol{x}}\right). \tag{A.4}$$

Define $\widetilde{\boldsymbol{Y}}$ to be the data matrix such that each row is a data point; that is, $\widetilde{\boldsymbol{Y}}_{n:} = \widetilde{\boldsymbol{y}}_n^T$. Define $\boldsymbol{X}$, $\widetilde{\boldsymbol{X}}$, $\boldsymbol{W}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, and $\boldsymbol{U}$ similarly. Let $\boldsymbol{\theta} = (\boldsymbol{\Xi}, \boldsymbol{\Lambda}, \boldsymbol{\Psi}, \boldsymbol{\zeta})$ denote the set of all parameters. We wish to learn $\boldsymbol{\theta}$ by maximising the (penalised) log likelihood $\log p(\widetilde{\boldsymbol{Y}}|\boldsymbol{X})$. We do this via EM, so we first write down the joint density:

$$\begin{aligned}
&p(\widetilde{\boldsymbol{Y}}, \boldsymbol{Z}, \boldsymbol{W}|\boldsymbol{X}) \\
&= \prod_{n=1}^{N} \prod_{m=1}^{M} \left[p\left(w_{nm} = 1|\boldsymbol{x}_n\right) p\left(\boldsymbol{z}_n|w_{nm} = 1\right) p\left(\widetilde{\boldsymbol{y}}_n|\boldsymbol{x}_n, w_{nm} = 1, \boldsymbol{z}_n\right)\right]^{w_{nm}} &\text{(A.5)} \\
&= \prod_{n=1}^{N} \prod_{m=1}^{M} \left[p\left(w_{nm} = 1|\boldsymbol{x}_n\right) p\left(\boldsymbol{u}_n|w_{nm} = 1\right) \prod_{v=1}^{V} f'_{mv}\left(\widetilde{y}_{nv} - \left(\boldsymbol{\Psi}_m^T\right)_{v:} \widetilde{\boldsymbol{x}}_n\right)\right]^{w_{nm}}. &\text{(A.6)}
\end{aligned}$$

We iteratively maximise the objective

$$Q\left(\boldsymbol{\theta}^{(t)}\right) = \mathbb{E}\left[\mathcal{L}\left(\boldsymbol{\theta}^{(t)}\right)|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] - \gamma\left(\boldsymbol{\theta}^{(t)}\right), \tag{A.7}$$

where $\mathcal{L}\left(\boldsymbol{\theta}\right)$ is the log likelihood,

$$\begin{aligned}
\mathcal{L}\left(\boldsymbol{\theta}\right) = \sum_{n,m} w_{nm} \Bigg[&\log p\left(w_{nm} = 1|\boldsymbol{x}_n\right) \\
&+ \log p\left(\boldsymbol{u}_n|w_{nm} = 1\right) \\
&+ \sum_{v} \log f'_{mv}\left(\widetilde{y}_{nv} - \mu_{nmv}\right)\Bigg],
\end{aligned} \tag{A.8}$$

$\boldsymbol{\mu}_{nm} \equiv \boldsymbol{\Psi}_m^T \widetilde{\boldsymbol{x}}_n$, and $\gamma(\boldsymbol{\theta})$ is a sum of the $L1$ penalties that we apply to the parameters $\boldsymbol{\theta}$.

In each M step of EM, we do not fully maximise $Q\left(\boldsymbol{\theta}\right)$. Instead, we partially maximise by performing one maximisation for each of $\boldsymbol{\Xi}$, $\boldsymbol{\Lambda}$, $\boldsymbol{\Psi}$, and $\boldsymbol{\zeta}$ conditioned on the other parameters. In the following sections, we describe how to find the expectations, and how to maximise, for each of the parameter groups.

A.1 The Mixing Model Parameters

Here, we wish to maximise $Q\left(\boldsymbol{\theta}\right)$ with respect to $\boldsymbol{\Xi}$, while fixing the current values of $(\boldsymbol{\Lambda}, \boldsymbol{\Psi}, \boldsymbol{\zeta})$. Only the first term in Equation (A.8) involves $\boldsymbol{\Xi}$, so the objective here is

$$Q_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right) \equiv \mathbb{E}\left[\sum_{n,m} w_{nm} \log p\left(w_{nm} = 1|\boldsymbol{x}_n; \boldsymbol{\Xi}^{(t)}\right)|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] - \gamma_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right), \tag{A.9}$$

where the expectation is over w_{nm}. Define

$$\overline{w}_{nm} \equiv \mathbb{E}\left[w_{nm}|\boldsymbol{X},\widetilde{\boldsymbol{Y}},\boldsymbol{\theta}^{(t-1)}\right] \tag{A.10}$$

$$= \mathbb{E}\left[w_{nm}|\boldsymbol{x}_n,\widetilde{\boldsymbol{y}}_n,\boldsymbol{\theta}^{(t-1)}\right] \tag{A.11}$$

$$= p\left(w_{nm}=1|\boldsymbol{x}_n,\widetilde{\boldsymbol{y}}_n;\boldsymbol{\theta}^{(t-1)}\right) \tag{A.12}$$

$$= \frac{p\left(\widetilde{\boldsymbol{y}}_n|w_{nm}=1,\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)p\left(w_{nm}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}^{(t-1)}\right)}{p\left(\widetilde{\boldsymbol{y}}_n|\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)}, \tag{A.13}$$

where $p\left(\widetilde{\boldsymbol{y}}_n|\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)=\sum_{j=1}^{M}p\left(\widetilde{\boldsymbol{y}}_n|w_{nj}=1,\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)p\left(w_{nj}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}^{(t-1)}\right)$. The likelihood $p\left(\widetilde{\boldsymbol{y}}_n|w_{nj}=1,\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)=\int p\left(\widetilde{\boldsymbol{y}}_n,\boldsymbol{z}_n|w_{nj}=1,\boldsymbol{x}_n;\boldsymbol{\theta}^{(t-1)}\right)d\boldsymbol{z}_n$ is a non-paranormal density. Equation (A.13) is used to compute $\overline{w}_{nm}$ in the E step. The M-step objective then becomes

$$Q_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right)=\sum_{n,m}\overline{w}_{nm}\log p\left(w_{nm}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}^{(t)}\right)-\gamma_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right). \tag{A.14}$$

At this point, we must choose a form for the distribution of $\boldsymbol{w}$ given $\boldsymbol{x}$. We worked with a multinomial logit model:

$$p\left(w_{nm}=1|\boldsymbol{x}_n;\boldsymbol{\Xi}\right)=\frac{\exp\left(\boldsymbol{\xi}_m^T\widetilde{\boldsymbol{x}}_n\right)}{\sum_{j=1}^{M}\exp\left(\boldsymbol{\xi}_j^T\widetilde{\boldsymbol{x}}_n\right)}, \tag{A.15}$$

where $\boldsymbol{\xi}_m=\left(\boldsymbol{\Xi}_{m:}\right)^T$. The values in the first column of $\boldsymbol{\Xi}$ – which multiply the unit element of $\widetilde{\boldsymbol{x}}$ – can be thought of as a bias. When training this model, we apply $L1$ penalties to the elements of $\boldsymbol{\Xi}$, except for the bias:

$$\gamma_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right)=\sum_{m=1}^{M}\sum_{c=1}^{C}\left(\Gamma_{\Xi}\right)_{mc}\left|\Xi_{m(c+1)}^{(t)}\right|, \tag{A.16}$$

where $\boldsymbol{\Gamma}_{\Xi}\in\mathbb{R}_{+}^{M\times C}$ is a penalty matrix.

The objective function is now

$$Q_{\Xi}\left(\boldsymbol{\Xi}^{(t)}\right)=\sum_{n,m}\overline{w}_{nm}\left[\boldsymbol{\xi}_m^{(t)T}\widetilde{\boldsymbol{x}}_n-\log\left\{\sum_{j=1}^{M}\exp\left(\boldsymbol{\xi}_j^{(t)T}\widetilde{\boldsymbol{x}}_n\right)\right\}\right]-\sum_{m,c}\left(\Gamma_{\Xi}\right)_{mc}\left|\Xi_{m(c+1)}^{(t)}\right|. \tag{A.17}$$

The are various ways to maximise this function. We used the L1General package for Matlab (Schmidt et al., 2007).

A.2 The Precision Matrix

Now, consider maximising $Q(\boldsymbol{\theta})$ with respect to $\boldsymbol{\Lambda}$, while holding $(\boldsymbol{\Xi}, \boldsymbol{\Psi}, \boldsymbol{\zeta})$ fixed. Only the second term in Equation (A.8) involves $\boldsymbol{\Lambda}$, so the objective is

$$Q_{\Lambda}\left(\boldsymbol{\Lambda}^{(t)}\right) \equiv \mathbb{E}\left[\sum_{n,m} w_{nm} \log p\left(\boldsymbol{u}_n | w_{nm}=1; \boldsymbol{\Lambda}_m^{(t)}\right) | \boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] - \gamma_{\Lambda}\left(\boldsymbol{\Lambda}^{(t)}\right), \quad \text{(A.18)}$$

where the final term contains an $L1$ penalty for each element of each precision matrix:

$$\gamma_{\Lambda}\left(\boldsymbol{\Lambda}^{(t)}\right) = \sum_{m=1}^{M} \sum_{i=1}^{D} \sum_{j=1}^{D} \left(\Gamma_{\Lambda_m}\right)_{ij} \left| \left(\Lambda_m^{(t)}\right)_{ij} \right|; \quad \text{(A.19)}$$

$\left(\boldsymbol{\Gamma}_{\Lambda_m}\right) \in \mathbb{R}_+^{D \times D}$ is the matrix of penalties for component m; and $D \equiv V + H$ is the cardinality of vector $\boldsymbol{u}$.

Notice that Equation (A.18) can be written as a sum over mixture components m, each term involving a single precision matrix $\boldsymbol{\Lambda}_m$. It will therefore suffice to consider the E and M steps for each component independently. Let $Q_{\Lambda} \equiv \sum_m Q_{\Lambda_m}$, where

$$\begin{aligned} Q_{\Lambda_m}\left(\boldsymbol{\Lambda}^{(t)}\right) \equiv & \sum_n \mathbb{E}\left[w_{nm} \log p\left(\boldsymbol{u}_n | w_{nm}=1; \boldsymbol{\Lambda}_m^{(t)}\right) | \boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ & - \sum_{i,j} \left(\Gamma_{\Lambda_m}\right)_{ij} \left| \left(\Lambda_m^{(t)}\right)_{ij} \right|. \end{aligned} \quad \text{(A.20)}$$

Since we focus on a single component, we drop the m index for clarity.

We examine the E step first – that is, how to compute the expectation in Equation (A.20). This involves a single data point, so we temporarily drop the n index. The vector $\boldsymbol{u}$ is Gaussian-distributed, so the expectation becomes

$$\begin{aligned} & \mathbb{E}\left[w \log p\left(\boldsymbol{u} | w=1; \boldsymbol{\Lambda}\right) | \boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ =& \overline{w} \mathbb{E}\left[\log p\left(\boldsymbol{u} | w=1; \boldsymbol{\Lambda}\right) | \boldsymbol{x}, \widetilde{\boldsymbol{y}}, \boldsymbol{\theta}^{(t-1)}\right] & \text{(A.21)} \\ =& \frac{\overline{w}}{2} \mathbb{E}\left[\log\det \boldsymbol{\Lambda} - \boldsymbol{u}^T \boldsymbol{\Lambda} \boldsymbol{u} - D \log 2\pi\right] & \text{(A.22)} \\ =& \frac{\overline{w}}{2} \left\{\log\det \boldsymbol{\Lambda} - \mathbb{E}\left[\operatorname{tr}\left(\boldsymbol{u}\boldsymbol{u}^T \boldsymbol{\Lambda}\right)\right] - D \log 2\pi\right\}. & \text{(A.23)} \end{aligned}$$

The product $\boldsymbol{u}\boldsymbol{u}^T$ may be partitioned according to the partition of $\boldsymbol{u}$ into $\boldsymbol{y}$ and $\boldsymbol{z}$, so that the trace term may be written

$$\mathbb{E}\left[\operatorname{tr}\left(\boldsymbol{u}\boldsymbol{u}^T \boldsymbol{\Lambda}\right)\right] = \operatorname{tr}\left\{ \begin{pmatrix} \mathbb{E}\left[\boldsymbol{y}\boldsymbol{y}^T\right] & \mathbb{E}\left[\boldsymbol{y}\boldsymbol{z}^T\right] \\ \mathbb{E}\left[\boldsymbol{z}\boldsymbol{y}^T\right] & \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^T\right] \end{pmatrix} \boldsymbol{\Lambda} \right\}. \quad \text{(A.24)}$$

We consider each of the expectations in turn. The first is straightforward: because $\boldsymbol{y} = f\left(\widetilde{\boldsymbol{y}} - \left(\boldsymbol{\Psi}^{(t)}\right)^T \widetilde{\boldsymbol{x}}; \boldsymbol{\zeta}^{(t)}\right)$ is not a random variable, $\mathbb{E}\left[\boldsymbol{y}\boldsymbol{y}^T\right] = \boldsymbol{y}\boldsymbol{y}^T$. The lower left block is the transpose of the upper right:

$$\mathbb{E}\left[\boldsymbol{z}\boldsymbol{y}^T\right]^T = \mathbb{E}\left[\boldsymbol{y}\boldsymbol{z}^T\right] = \boldsymbol{y}\mathbb{E}\left[\boldsymbol{z}|w=1,\boldsymbol{y},\boldsymbol{\Lambda}^{(t-1)}\right]^T = \boldsymbol{y}\bar{\boldsymbol{z}}^T, \tag{A.25}$$

where $\bar{\boldsymbol{z}} \equiv \mathbb{E}\left[\boldsymbol{z}|w=1,\boldsymbol{y},\boldsymbol{\Lambda}^{(t-1)}\right] = -\left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1}\boldsymbol{\Lambda}_{zy}^{(t-1)}\boldsymbol{y}$ is the mean of a conditional Gaussian. The expectation of the lower-right block may be written

$$\begin{aligned}
\mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^T\right] &= \mathbb{E}\left[\boldsymbol{z}\boldsymbol{z}^T|w=1,\boldsymbol{y},\boldsymbol{\Lambda}^{(t-1)}\right] && \text{(A.26)}\\
&= \mathrm{Cov}\left(\boldsymbol{z}|w=1,\boldsymbol{y},\boldsymbol{\Lambda}^{(t-1)}\right) + \bar{\boldsymbol{z}}\bar{\boldsymbol{z}}^T && \text{(A.27)}\\
&= \left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1} + \bar{\boldsymbol{z}}\bar{\boldsymbol{z}}^T. && \text{(A.28)}
\end{aligned}$$

Finally, we substitute Equations (A.25, A.28) into (A.24), replace the n indices, and form the weighted sum over n:

$$\begin{aligned}
\sum_n \overline{w}_n \mathbb{E}\left[\mathrm{tr}\left(\boldsymbol{u}_n\boldsymbol{u}_n^T\boldsymbol{\Lambda}\right)\right] &= \sum_n \mathrm{tr}\left\{\begin{pmatrix} \overline{w}_n\boldsymbol{y}_n\boldsymbol{y}_n^T & \overline{w}_n\boldsymbol{y}_n\bar{\boldsymbol{z}}_n^T \\ \overline{w}_n\bar{\boldsymbol{z}}_n\boldsymbol{y}_n^T & \overline{w}_n\left[\left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1} + \bar{\boldsymbol{z}}_n\bar{\boldsymbol{z}}_n^T\right] \end{pmatrix}\boldsymbol{\Lambda}\right\} && \text{(A.29)}\\
&= \left(\sum_n \overline{w}_n\right)\mathrm{tr}\left\{\boldsymbol{S}\boldsymbol{\Lambda}\right\}, && \text{(A.30)}
\end{aligned}$$

where we define $\boldsymbol{S}$ such that

$$\begin{aligned}
\boldsymbol{S}_{yy} &= \left(\sum_n \overline{w}_n\right)^{-1}\boldsymbol{Y}^T\overline{\boldsymbol{W}}\boldsymbol{Y}, && \text{(A.31)}\\
\boldsymbol{S}_{zy}^T = \boldsymbol{S}_{yz} &= \left(\sum_n \overline{w}_n\right)^{-1}\boldsymbol{Y}^T\overline{\boldsymbol{W}}\,\overline{\boldsymbol{Z}}, && \text{(A.32)}\\
\boldsymbol{S}_{zz} &= \left(\boldsymbol{\Lambda}_{zz}^{(t-1)}\right)^{-1} + \left(\sum_n \overline{w}_n\right)^{-1}\overline{\boldsymbol{Z}}^T\overline{\boldsymbol{W}}\,\overline{\boldsymbol{Z}}, && \text{(A.33)}
\end{aligned}$$

and $\overline{\boldsymbol{W}}$ is a diagonal matrix such that $\mathrm{diag}\left(\overline{\boldsymbol{W}}\right) = (\overline{w}_1,\ldots,\overline{w}_N)^T$. The primary task of the E step is to compute the matrix $\boldsymbol{S}$.

We turn now to the M step. The task is to find a $\boldsymbol{\Lambda}_m$ that maximises Q_{Λ_m}. Constants in Q_{Λ_m} do not affect the result, so we drop the final term in Equation (A.23) before substituting into (A.20), and define the new objective

$$Q'_{\Lambda_m}(\boldsymbol{\Lambda}_m) \equiv \frac{N_m}{2}\left\{\log\det\boldsymbol{\Lambda}_m - \mathrm{tr}\left(\boldsymbol{S}_m\boldsymbol{\Lambda}_m\right)\right\} - \sum_{i,j}\left(\Gamma_{\Lambda_m}\right)_{ij}\left|\left(\Lambda_m\right)_{ij}\right|, \tag{A.34}$$

where $N_m = \sum_n \overline{w}_{nm}$ is the expected number of data points for which component m is responsible. This is the graphical lasso objective; see Section 2.1.2. But, as discussed

in Section 2.2.1, it is necessary to constrain $\mathbf{\Lambda}_m$. We fix the diagonal of $(\mathbf{\Lambda}_m)_{zz}$ to unity. So we estimate $\mathbf{\Lambda}_m$ as follows:

$$\widehat{\mathbf{\Lambda}}_m = \underset{\mathbf{\Lambda}_m \succeq 0 \,:\, \mathrm{diag}\left((\mathbf{\Lambda}_m)_{zz}\right)=\mathbf{1}}{\arg\max} Q'_{\Lambda_m}(\mathbf{\Lambda}_m). \tag{A.35}$$

We use one of the packages SDPT3 (Toh et al., 1999); LogdetPPA (Wang et al., 2010); or L1General (Schmidt et al., 2007) to solve this optimisation problem; see Section 2.2.2.

A.3 The Gaussianising Functions

Here, we consider maximising $Q(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}^f = (\mathbf{\Psi}, \boldsymbol{\zeta})$, while holding $(\mathbf{\Xi}, \mathbf{\Lambda})$ fixed. The second and third terms in Equation (A.8) involve $\boldsymbol{\theta}^f$, so the objective is

$$\begin{aligned} Q_{\theta^f}\left(\mathbf{\Psi}^{(t)}, \boldsymbol{\zeta}^{(t)}\right) \equiv{}& \mathbb{E}\left[\sum_{n,m} w_{nm} \log p(\boldsymbol{u}_n | w_{nm}=1) \,|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ &+ \mathbb{E}\left[\sum_{n,m} w_{nm} \sum_{v} \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv}) \,|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ &- \gamma_\Psi\left(\mathbf{\Psi}^{(t)}\right) - \gamma_\zeta\left(\boldsymbol{\zeta}^{(t)}\right), \end{aligned} \tag{A.36}$$

where γ_Ψ consists of an L1 penalty on all parameters $\mathbf{\Psi}^{(t)}$ except for the biases:

$$\gamma_\Psi\left(\mathbf{\Psi}^{(t)}\right) = \sum_{m=1}^{M}\sum_{v=1}^{V}\sum_{c=1}^{C} (\Gamma_{\Psi_m})_{cv} \left|\left(\Psi_m^{(t)}\right)_{(c+1)v}\right|, \tag{A.37}$$

where $(\mathbf{\Gamma}_{\Psi_m}) \in \mathbb{R}_+^{C\times V}$ is the matrix of penalties for $\mathbf{\Psi}_m$. Similarly to Section A.2, we can write the objective $Q_{\theta^f} = \sum_m Q_{\theta_m^f}$, such that each term in the sum involves only the parameters $\boldsymbol{\theta}_m^f$:

$$\begin{aligned} Q_{\theta_m^f}\left(\mathbf{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}\right) \equiv{}& \mathbb{E}\left[\sum_{n} w_{nm} \log p(\boldsymbol{u}_n | w_{nm}=1) \,|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ &+ \mathbb{E}\left[\sum_{n} w_{nm} \sum_{v} \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv}) \,|\boldsymbol{X}, \widetilde{\boldsymbol{Y}}, \boldsymbol{\theta}^{(t-1)}\right] \\ &- \sum_{v=1}^{V}\sum_{c=1}^{C} (\Gamma_{\Psi_m})_{cv} \left|\left(\Psi_m^{(t)}\right)_{(c+1)v}\right| - \gamma_{\zeta_m}\left(\boldsymbol{\zeta}_m^{(t)}\right). \end{aligned} \tag{A.38}$$

The E step involves computing the two expectations. The first was examined in Section A.2. The second is straightforward:

$$\mathbb{E}\left[\sum_{n} w_{nm} \sum_{v} \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv})\right] = \sum_{n} \overline{w}_{nm} \sum_{v} \log f'_{mv}(\widetilde{y}_{nv} - \mu_{nmv}). \tag{A.39}$$

Now consider the M step. Using Equations (A.23, A.30), and dropping the terms not involving $\boldsymbol{\theta}^f$, we define a new objective

$$\begin{aligned} Q'_{\theta_m^f}\left(\boldsymbol{\Psi}_m^{(t)}, \boldsymbol{\zeta}_m^{(t)}\right) \equiv & -\frac{N_m}{2}\left\{\operatorname{tr}\left((\boldsymbol{S}_m)_{yy}(\boldsymbol{\Lambda}_m)_{yy}\right)+2\operatorname{tr}\left((\boldsymbol{S}_m)_{zy}(\boldsymbol{\Lambda}_m)_{yz}\right)\right\} \\ & +\sum_n \overline{w}_{nm}\sum_v \log f'_{mv}(\widetilde{y}_{nv}-\mu_{nmv}) \\ & -\gamma_{\Psi_m}\left(\boldsymbol{\Psi}_m^{(t)}\right)-\gamma_{\zeta_m}\left(\boldsymbol{\zeta}_m^{(t)}\right). \end{aligned} \tag{A.40}$$

First, consider the special case in which each component is Gaussian; that is, f_{mv} is the identity map. This simplifies the objective; in particular, there are no parameters $\boldsymbol{\zeta}_m$. We can define the following objective for $\boldsymbol{\Psi}_m$:

$$\begin{aligned} Q_{\Psi_m}\left(\boldsymbol{\Psi}_m^{(t)}\right) \equiv & -\frac{N_m}{2}\left\{\operatorname{tr}\left((\boldsymbol{S}_m)_{yy}(\boldsymbol{\Lambda}_m)_{yy}\right)+2\operatorname{tr}\left((\boldsymbol{S}_m)_{yz}(\boldsymbol{\Lambda}_m)_{zy}\right)\right\} \\ & -\gamma_{\Psi_m}\left(\boldsymbol{\Psi}_m^{(t)}\right), \end{aligned} \tag{A.41}$$

where

$$(\boldsymbol{S}_m)_{yy} = \frac{1}{N_m}\boldsymbol{Y}_m^T\overline{\boldsymbol{W}}_m\boldsymbol{Y}_m, \tag{A.42}$$

$$(\boldsymbol{S}_m)_{yz} = \frac{1}{N_m}\boldsymbol{Y}_m^T\overline{\boldsymbol{W}}_m\overline{\boldsymbol{Z}}_m, \tag{A.43}$$

and $\boldsymbol{Y}_m = \widetilde{\boldsymbol{Y}} - \widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m^{(t)}$. Dropping further terms that do not involve $\boldsymbol{\Psi}_m^{(t)}$, and dropping the (t) superscript for clarity, the objective becomes

$$\begin{aligned} Q'_{\Psi_m}(\boldsymbol{\Psi}_m) \equiv & \frac{1}{2}\operatorname{tr}\left\{\left(2\boldsymbol{\Psi}_m^T\widetilde{\boldsymbol{X}}^T\overline{\boldsymbol{W}}_m\widetilde{\boldsymbol{Y}}-\boldsymbol{\Psi}_m^T\widetilde{\boldsymbol{X}}^T\overline{\boldsymbol{W}}_m\widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)(\boldsymbol{\Lambda}_m)_{yy}\right\} \\ & +\operatorname{tr}\left\{\boldsymbol{\Psi}_m^T\widetilde{\boldsymbol{X}}^T\overline{\boldsymbol{W}}_m\overline{\boldsymbol{Z}}_m(\boldsymbol{\Lambda}_m)_{zy}\right\}-\gamma_{\Psi_m}(\boldsymbol{\Psi}_m) \end{aligned} \tag{A.44}$$

$$\begin{aligned} = & \operatorname{tr}\left\{\boldsymbol{\Psi}_m^T\widetilde{\boldsymbol{X}}^T\overline{\boldsymbol{W}}_m\left[\overline{\boldsymbol{Z}}_m(\boldsymbol{\Lambda}_m)_{zy}+\left(\widetilde{\boldsymbol{Y}}-\frac{1}{2}\widetilde{\boldsymbol{X}}\boldsymbol{\Psi}_m\right)(\boldsymbol{\Lambda}_m)_{yy}\right]\right\} \\ & -\gamma_{\Psi_m}(\boldsymbol{\Psi}_m). \end{aligned} \tag{A.45}$$

In the absence of penalties, this is a quadratic in $\boldsymbol{\Psi}_m$, so the maximum can be found analytically. With penalties, we use the L1General package (Schmidt et al., 2007) to optimise.

If f_{mv} is not the identity map (so the experts are non-Gaussian), optimising $\boldsymbol{\theta}^f$ is more difficult. If the marginals F_{mv} are simple enough, the gradients of (A.40) with respect to $\boldsymbol{\Psi}_m^{(t)}$ and $\boldsymbol{\zeta}_m^{(t)}$ could be derived, although they would be messy. But in general, the gradients are not available; our optimisation method is described in Section 2.4.2.

Appendix B

The FTSE Data Set

Here we list all the assets comprising the data set described in Section 2.5.1, along with their corresponding market sectors. An asterisk indicates that the asset is included in the reduced data set of 19 companies that was used for many of the experiments.

Symbol	Company Name	Market Sector
AAL*	Anglo American	Basic Materials
ABF*	Associated British Foods	Consumer Goods
ADM	Admiral Group	Financial
AGK	Aggreko	Services
AMEC	AMEC	Basic Materials
ANTO*	Antofagasta	Basic Materials
ARM	ARM Holdings	Technology
AU	Autonomy	Technology
AV	Aviva	Financial
AZN	AstraZeneca	Healthcare
BA	BAE Systems	Industrial Goods
BARC*	Barclays	Financial
BATS*	British American Tobacco	Consumer Goods
BG	BG Group	Basic Materials
BLND	British Land Co	Financial
BLT*	BHP Billiton	Basic Materials
BP	BP	Basic Materials
BRBY	Burberry Group	Services
BSY	British Sky Broadcasting Group	Services

Symbol	**Company Name**	**Market Sector**
BT-A	BT Group	Technology
CCL	Carnival	Consumer Goods
CNA	Centrica	Utilities
CNE	Cairn Energy	Basic Materials
CPG	Compass Group	Services
CPI	Capita	Services
DGE*	Diageo	Consumer Goods
GFS	G4S	Services
GKN	GKN	Consumer Goods
GSK	GlaxoSmithKline	Healthcare
HMSO	Hammerson	Financial
HSBA*	HSBC Holdings	Financial
IAG	International Consolidated Airlines Group	Industrial Goods
IAP	ICAP	Financial
III	3i Group	Financial
IMI	IMI	Industrial Goods
IMT*	Imperial Tobacco Group	Consumer Goods
INVP	Investec	Financial
IPR	International Power	Utilities
ISYS	Invensys	Industrial Goods
ITRK	Intertek Group	Services
ITV	ITV	Services
JMAT	Johnson Matthey	Basic Materials
KGF	Kingfisher	Services
LAND	Land Securities Group	Financial
LGEN	Legal and General Group	Financial
LLOY*	Lloyds Banking Group	Financial
LMI*	Lonmin	Basic Materials
MKS	Marks and Spencer Group	Services
MRW	WM Morrison Supermarkets	Services
NXT	Next	Services
OML	Old Mutual	Financial
PRU	Prudential	Financial
PSON	Pearson	Services

Symbol	Company Name	Market Sector
RBS*	Royal Bank of Scotland Group	Financial
REL	Reed Elsevier	Services
REX	REXAM	Consumer Goods
RIO*	Rio Tinto	Basic Materials
RR	Rolls-Royce Group	Industrial Goods
RRS*	Randgold Resources	Basic Materials
RSA	RSA Insurance Group	Financial
SAB*	SABMiller	Consumer Goods
SBRY	Sainsbury	Services
SDR	Schroders	Financial
SGE	Sage Group	Technology
SHP	Shire	Healthcare
SMIN	Smiths Group	Industrial Goods
SN	Smith and Nephew	Healthcare
SRP	Serco Group	Services
SSE	SSE	Utilities
STAN*	Standard Chartered	Financial
SVT	Severn Trent	Utilities
TLW	Tullow Oil	Basic Materials
TSCO	Tesco	Services
TT	TUI Travel	Consumer Goods
ULVR*	Unilever	Consumer Goods
VED*	Vedanta Resources	Basic Materials
VOD	Vodafone Group	Technology
WEIR	Weir Group	Industrial Goods
WOS	Wolseley	Industrial Goods
WPP	WPP	Services
XTA*	Xstrata	Basic Materials

Appendix C

The S&P 500 Data Set

Here we list all the assets – along with their corresponding market sectors – and technical indicators comprising the data set described in Section 3.3.1.

C.1 Assets

Symbol	Company Name	Market Sector
AAPL	Apple	Information Technology
ABT	Abbott Laboratories	Health Care
AIG	American International Group	Financials
AMGN	Amgen	Health Care
AXP	American Express	Financials
BA	Boeing Company	Industrials
BAC	Bank of America	Financials
C	Citigroup	Financials
CMCSA	Comcast	Consumer Discretionary
COP	ConocoPhillips	Energy
CSCO	Cisco Systems	Information Technology
CVX	Chevron	Energy
DIS	The Walt Disney Company	Consumer Discretionary
GE	General Electric	Industrials
HD	Home Depot	Consumer Discretionary
HPQ	Hewlett-Packard	Information Technology
IBM	International Business Machines	Information Technology
INTC	Intel	Information Technology

Symbol	**Company Name**	**Market Sector**
JNJ	Johnson & Johnson	Health Care
JPM	JP Morgan Chase	Financials
KO	The Coca Cola Company	Consumer Staples
MDT	Medtronic	Health Care
MO	Altria Group	Consumer Staples
MRK	Merck	Health Care
MS	Morgan Stanley	Financials
MSFT	Microsoft	Information Technology
ORCL	Oracle	Information Technology
PEP	PepsiCo	Consumer Staples
PFE	Pfizer	Health Care
PG	Procter & Gamble	Consumer Staples
QCOM	Qualcomm	Information Technology
SLB	Schlumberger	Energy
T	AT & T	Telecommunications Services
TWX	Time Warner	Consumer Discretionary
TYC	Tyco International	Industrials
UNH	United Health Group	Health Care
USB	US Bancorp	Financials
UTX	United Technologies	Industrials
VZ	Verizon Communications	Telecommunications Services
WFC	Wells Fargo	Financials
WMT	Wal-Mart Stores	Consumer Staples
XOM	Exxon Mobil	Energy

C.2 Technical Indicators

Abbreviation	Description
DX	Welles Wilder's direction index
ADX	Welles Wilder's average direction index
AroonUp	Aroon up indicator
AroonDn	Aroon down indicator
ATR	Average true range
BBlo	Bollinger bands low line
BBhi	Bollinger bands high line
BBpct	Bollinger bands percent bandwidth
CCI	Commodity channel index
ChaikinAD	Chaikin accumulation/distribution line
ChaikinVol	Chaikin volatility
CLV	Close location value
ChaikinMF	Chaikin money flow
DPO	Detrended price oscillator
DVI	David Varadi's intermediate oscillator
EMV	Arms' ease of movement value
KST	Know sure thing indicator
MACD	Moving average convergence/divergence oscillator
MFI	Money flow index
OBV	On balance volume
TDI	Trend detection index
TRIX	Triple smoothed exponential oscillator
VHF	Vertical horizontal filter
Vol	Close-to-close volatility
WilliamsAD	Williams accumulation/distribution line

Appendix D

HMC for the GWishart – Derivations

Here we derive the energy function and its derivatives, as required by HMC, for the GWishart distribution. In Section D.1, we consider the standard representation of the GWishart. In Section D.2, we look at the Cholesky representation as discussed in Section 4.2.2.

D.1 The Standard Representation

Recall the GWishart density $W_G(b, \boldsymbol{D})$:

$$p(\boldsymbol{\Lambda}) = \frac{1}{I_G(b, \boldsymbol{D})} (\det \boldsymbol{\Lambda})^{\frac{b-2}{2}} \exp\left[-\frac{1}{2}\operatorname{tr}(\boldsymbol{D}\boldsymbol{\Lambda})\right]. \tag{D.1}$$

The energy associated with this density is defined such that

$$p(\boldsymbol{\Lambda}) = \frac{1}{I_G(b, \boldsymbol{D})} \exp\{-E(\boldsymbol{\Lambda})\}, \tag{D.2}$$

so we have

$$E(\boldsymbol{\Lambda}) = \frac{1}{2}\left[\operatorname{tr}(\boldsymbol{D}\boldsymbol{\Lambda}) - (b-2)\log\det\boldsymbol{\Lambda}\right]. \tag{D.3}$$

The following are standard formulae of matrix calculus. See, for example, (Petersen and Pedersen, 2006). If $\boldsymbol{X}$ is a symmetric matrix, and $\boldsymbol{A}$ is a constant square matrix, then

$$\frac{\partial}{\partial \boldsymbol{X}} \operatorname{tr}(\boldsymbol{A}\boldsymbol{X}) = \boldsymbol{A} + \boldsymbol{A}^T - \boldsymbol{A} \odot \boldsymbol{I}, \tag{D.4}$$

$$\frac{\partial}{\partial \boldsymbol{X}} \log\det\boldsymbol{X} = 2\boldsymbol{X}^{-1} - \boldsymbol{X}^{-1} \odot \boldsymbol{I}. \tag{D.5}$$

The derivative of the energy is therefore

$$\frac{\partial E}{\partial \boldsymbol{\Lambda}} = \frac{1}{2}\left[2\boldsymbol{D} - \boldsymbol{D} \odot \boldsymbol{I} - (b-2)(2\boldsymbol{\Sigma} - \boldsymbol{\Sigma} \odot \boldsymbol{I})\right]. \tag{D.6}$$

D.2 The Cholesky Representation

Let $\boldsymbol{\Lambda}$ denote a GWishart-distributed matrix: $\boldsymbol{\Lambda} \sim W_G(b, \boldsymbol{D})$. Let $\boldsymbol{\Psi}$ denote the upper-triangular matrix formed from the Cholesky decompositions of $\boldsymbol{D}^{-1}$ and $\boldsymbol{\Lambda}$ as follows:

$$\boldsymbol{D}^{-1} = \boldsymbol{T}^T \boldsymbol{T}; \tag{D.7}$$

$$\boldsymbol{\Lambda} = \boldsymbol{\Phi}^T \boldsymbol{\Phi}; \tag{D.8}$$

$$\boldsymbol{\Psi} = \boldsymbol{\Phi} \boldsymbol{T}^{-1}. \tag{D.9}$$

From Atay-Kayis and Massam (2005), we know that:

- $\boldsymbol{\Psi}^{\mathcal{V}}$ are free variables, while $\boldsymbol{\Psi}^{\overline{\mathcal{V}}}$ are not free.
- If $\Psi_{rs} \in \boldsymbol{\Psi}^{\overline{\mathcal{V}}}$, it can be written as a function of the free elements as follows:

$$\begin{aligned} \Psi_{rs} = &\sum_{n=r}^{s-1} \left(-\Psi_{rn} T_{<ns]}\right) \\ &- \sum_{m=1}^{r-1} \left(\frac{\Psi_{mr} + \sum_{n=m}^{r-1} \Psi_{mn} T_{<nr]}}{\Psi_{rr}} \right) \left(\Psi_{ms} + \sum_{n=m}^{s-1} \Psi_{mn} T_{<ns]} \right), \end{aligned} \tag{D.10}$$

  where $T_{<mn]} \equiv \frac{T_{mn}}{T_{nn}}$. Define the row-wise order relation $<_r$ as follows: $(m,n) <_r (r,s)$ if $m < r$, or if $m = r$ and $n < s$. Notice that $\Psi_{rs} \in \boldsymbol{\Psi}^{\overline{\mathcal{V}}}$ depends only on $\Psi_{mn} <_r \Psi_{rs}$. The non-free elements must therefore be evaluated iteratively in row-wise order.

- The free elements have density

$$p\left(\boldsymbol{\Psi}^{\mathcal{V}}\right) \propto \prod_{i=1}^{p} \Psi_{ii}^{b+\nu_i-1} \exp\left(-\frac{1}{2} \sum_{1 \le i \le j \le p} \Psi_{ij}^2 \right), \tag{D.11}$$

  where $\nu_i \equiv |\{j : j > i, G_{ij} = 1\}|$.

The density in Equation (D.11) may be written in terms of an energy $E\left(\boldsymbol{\Psi}^{\mathcal{V}}\right)$ as follows:

$$p\left(\boldsymbol{\Psi}^{\mathcal{V}}\right) = \frac{1}{I_G(b, \boldsymbol{D})} \exp\left\{ -E\left(\boldsymbol{\Psi}^{\mathcal{V}}\right) \right\}. \tag{D.12}$$

For HMC, additive constants in the energy do not change the algorithm, so we drop them and redefine the energy:

$$E\left(\boldsymbol{\Psi}^{\mathcal{V}}\right) = -\sum_{i=1}^{p} (b + \nu_i - 1) \log \Psi_{ii} + \frac{1}{2} \sum_{1 \le i \le j \le p} \Psi_{ij}^2. \tag{D.13}$$

The energy gradients required for HMC are

$$\frac{\partial E}{\partial \Psi_{ii}} = \sum_{(r,s)\in\overline{\mathcal{V}}} \Psi_{rs}\frac{\partial \Psi_{rs}}{\partial \Psi_{ii}} - \frac{b+\nu_i-1}{\Psi_{ii}}, \tag{D.14}$$

$$\frac{\partial E}{\partial \Psi_{ij}} = \sum_{(r,s)\in\overline{\mathcal{V}}} \Psi_{rs}\frac{\partial \Psi_{rs}}{\partial \Psi_{ij}} + \Psi_{ij}, \quad \text{for } i<j. \tag{D.15}$$

We therefore require the derivatives $\frac{\partial \Psi_{rs}}{\partial \Psi_{ij}}$ of the non-free elements of $\mathbf{\Psi}$ with respect to the free elements. If $(r,s) <_r (i,j)$, this derivative is zero. For $(i,j) <_r (r,s)$, we differentiate (D.10) to find

$$\begin{aligned}\frac{\partial \Psi_{rs}}{\partial \Psi_{ij}} = &-\sum_{n=r}^{s-1}\frac{\partial \Psi_{rn}}{\partial \Psi_{ij}}T_{<ns]} \\ &-\sum_{m=1}^{r-1}\left[\left(\frac{\Psi_{mr}+\sum_{n=m}^{r-1}\Psi_{mn}T_{<nr]}}{\Psi_{rr}}\right)\left(\frac{\partial \Psi_{ms}}{\partial \Psi_{ij}}+\sum_{n=m}^{s-1}\frac{\partial \Psi_{mn}}{\partial \Psi_{ij}}T_{<ns]}\right)\right. \\ &\quad +\left(\Psi_{ms}+\sum_{n=m}^{s-1}\Psi_{mn}T_{<ns]}\right)\left\{\frac{1}{\Psi_{rr}}\left(\frac{\partial \Psi_{mr}}{\partial \Psi_{ij}}+\sum_{n=m}^{r-1}\frac{\partial \Psi_{mn}}{\partial \Psi_{ij}}T_{<nr]}\right)\right. \\ &\quad\quad \left.\left.-\left(\Psi_{mr}+\sum_{n=m}^{r-1}\Psi_{mn}T_{<nr]}\right)\frac{1}{\Psi_{rr}^2}\frac{\partial \Psi_{rr}}{\partial \Psi_{ij}}\right\}\right].\end{aligned} \tag{D.16}$$

For the partial derivatives $\frac{\partial \Psi_{ab}}{\partial \Psi_{ij}}$ that appear on the right hand side, if $(a,b)\in\mathcal{V}$, then

$$\frac{\partial \Psi_{ab}}{\partial \Psi_{ij}} = \begin{cases} 1 & \text{if } (a,b)=(i,j), \\ 0 & \text{if } (a,b)\neq(i,j). \end{cases} \tag{D.17}$$

For $(a,b)\in\overline{\mathcal{V}}$, notice that $(a,b) <_r (r,s)$. Therefore, for each (i,j), Equation (D.16) must be applied iteratively over (r,s) in row-wise order to compute all partial derivatives.

Bibliography

Agakov, F. V., Orchard, P. R., and Storkey, A. (2012). Discriminative Mixtures of Sparse Latent Fields for Risk Management. In *AISTATS*.

Aloui, R., Aïssa, M. S. B., and Nguyen, D. K. (2013). Conditional Dependence Structure between Oil Prices and Exchange Rates: A Copula-GARCH Approach. *Journal of International Money and Finance*, 32:719–738.

Antoniak, C. E. (1974). Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. *The Annals of Statistics*, 2(6):1152–1174.

Atay-Kayis, A. and Massam, H. (2005). A Monte Carlo Method for Computing the Marginal Likelihood in Nondecomposable Gaussian Graphical Models. *Biometrika*, 92(2):317–335.

Balkema, A. and De Haan, L. (1974). Residual Life Time at Great Age. *The Annals of Probability*, 2(5):792–804.

Banerjee, O., El Ghaoui, L., and d'Aspremont, A. (2008). Model Selection through Sparse Maximum Likelihood Estimation for Multivariate Gaussian or Binary Data. *The Journal of Machine Learning Research*, 9:485–516.

Bauwens, L., Laurent, S., and Rombouts, J. V. K. (2006). Multivariate GARCH Models: A Survey. *Journal of Applied Econometrics*, 21(1):79–109.

Bengio, Y. and Frasconi, P. (1995). An Input Output HMM Architecture. In *Advances in Neural Information Processing Systems 7*, pages 427–434. Morgan Kaufmann.

Bengio, Y. and Frasconi, P. (1996). Input-Output HMMs for Sequence Processing. *IEEE Transactions on Neural Networks*, 7(5):1231–1249.

Bengio, Y., Lauzon, V., and Ducharme, R. (2001). Experiments on the Application of IOHMMs to Model Financial Returns Series. *IEEE Transactions on Neural Networks*, 12(1):113–123.

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Bollerslev, T. (1986). Generalized Autoregressive Conditional Heteroskedasticity. *Journal of Econometrics*, 31(3):307 – 327.

Bollerslev, T., Engle, R. F., and Nelson, D. B. (1994). ARCH Models. *Handbook of Econometrics*, 4:2959–3038.

Boubaker, H. and Sghaier, N. (2013). Portfolio Optimization in the Presence of Dependent Financial Returns with Long Memory: A Copula Based Approach. *Journal of Banking and Finance*, 37(2):361–377.

Bradley, J. K. and Guestrin, C. (2010). Learning Tree Conditional Random Fields. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 127–134, Haifa, Israel. Omnipress.

Brechmann, E. C., Czado, C., and Aas, K. (2012). Truncated Regular Vines in High Dimensions with Application to Financial Data. *Canadian Journal of Statistics*, 40(1):68–85.

Carvalho, C. M. and West, M. (2007). Dynamic Matrix-Variate Graphical Models. *Bayesian Analysis*, 2(1):69–97.

Chandrasekaran, V., Parrilo, P. A., and Willsky, A. S. (2010). Latent Variable Graphical Model Selection via Convex Optimization. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 1610–1613. IEEE.

Chen, S., Donoho, D., and Saunders, M. (1998). Atomic Decomposition by Basis Pursuit. *SIAM Journal on Scientific Computing*, 20(1):33–61.

Cheng, J., Levina, E., Wang, P., and Zhu, J. (2012). Sparse Ising Models with Covariates. *ArXiv e-prints*, 1209.6342.

Cheng, Y. and Lenkoski, A. (2012). Hierarchical Gaussian Graphical Models: Beyond Reversible Jump. *Electronic Journal of Statistics*, 6:2309–2331.

Cherubini, U., Luciano, E., and Vecchiato, W. (2004). *Copula Methods in Finance*. Wiley.

Chow, C. and Liu, C. (1968). Approximating Discrete Probability Distributions with Dependence Trees. *IEEE Transactions on Information Theory*, 14(3):462–467.

Cuthbertson, K. (1996). *Quantitative Financial Economics: Stocks, Bonds, and Foreign Exchange*. Wiley.

Czado, C., Brechmann, E., and Gruber, L. (2013). Selection of Vine Copulas. In Jaworski, P., Durante, F., and Härdle, W. K., editors, *Copulae in Mathematical and Quantitative Finance*, pages 17–37. Springer Berlin Heidelberg.

d'Aspremont, A., Banerjee, O., and El Ghaoui, L. (2008). First-Order Methods for Sparse Covariance Selection. *SIAM Journal on Matrix Analysis and Applications*, 30(1):56–66.

Dobra, A., Eicher, T. S., and Lenkoski, A. (2010). Modeling Uncertainty in Macroeconomic Growth Determinants using Gaussian Graphical Models. *Statistical Methodology*, 7(3):292–306.

Dobra, A. and Lenkoski, A. (2011). Copula Gaussian Graphical Models and their Application to Modeling Functional Disability Data. *The Annals of Applied Statistics*, 5(2A):969–993.

Dobra, A., Lenkoski, A., and Rodriguez, A. (2011). Bayesian Inference for General Gaussian Graphical Models with Application to Multivariate Lattice Data. *Journal of the American Statistical Association*, 106(496):1418–1433.

Duchi, J., Gould, S., and Koller, D. (2008). Projected Subgradient Methods for Learning Sparse Gaussians. In *UAI*, pages 145–152.

Efron, B., Hastie, T., Johnstone, I., and Tibshirani, R. (2004). Least Angle Regression. *The Annals of Statistics*, 32(2):407–451.

Elidan, G. (2010). Copula Bayesian Networks. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 559–567.

Elidan, G. (2012a). Copula Network Classifiers (CNCs). In *International Conference on Artificial Intelligence and Statistics*, pages 346–354.

Elidan, G. (2012b). Lightning-Speed Structure Learning of Nonlinear Continuous Networks. In *International Conference on Artificial Intelligence and Statistics*, pages 355–363.

Elidan, G., Nachman, I., and Friedman, N. (2007). "Ideal parent" Structure Learning for Continuous Variable Bayesian Networks. *Journal of Machine Learning Research*, 8:1799–1833.

Engelhardt, B. E. and Stephens, M. (2010). Analysis of Population Structure: a Unifying Framework and Novel Methods based on Sparse Factor Analysis. *PLoS Genetics*, 6(9).

Engle, R. (1982). Autoregressive Conditional Heteroscedasticity with Estimates of the Variance of United Kingdom Inflation. *Econometrica: Journal of the Econometric Society*, 50(4):987–1007.

Ernst, J., Vainas, O., Harbison, C. T., Simon, I., and Bar-Joseph, Z. (2007). Reconstructing Dynamic Regulatory Maps. *Molecular Systems Biology*, 3.

Everitt, B. S. (1984). *An Introduction to Latent Variable Models*. Chapman and Hall, London.

Fama, E. (1965). The Behavior of Stock-Market Prices. *Journal of Business*, 38(1):34–105.

Fan, J. and Li, R. (2001). Variable Selection via Nonconcave Penalized Likelihood and its Oracle Properties. *Journal of the American Statistical Association*, 96(456):1348–1360.

Fleming, J., Kirby, C., and Ostdiek, B. (2008). The Specification of GARCH Models with Stochastic Covariates. *Journal of Futures Markets*, 28(10):911–934.

Frank, I. E. and Friedman, J. H. (1993). A Statistical View of Some Chemometrics Regression Tools. *Technometrics*, 35(2):109–135.

Friedman, J., Hastie, T., and Tibshirani, R. (2008). Sparse Inverse Covariance Estimation with the Graphical Lasso. *Biostatistics*, 9(3):432–441.

Friedman, J., Hastie, T., and Tibshirani, R. (2010a). Applications of the Lasso and Grouped Lasso to the Estimation of Sparse Graphical Models. Technical report, Stanford University.

Friedman, J., Hastie, T., and Tibshirani, R. (2010b). Regularization Paths for Generalized Linear Models Via Coordinate Descent. *Journal of Statistical Software*, 33(1):1–22.

Friedman, N. (1997). Learning Belief Networks in the Presence of Missing Values and Hidden Variables. ICML '97.

Friedman, N., Nachman, I., and Peer, D. (1999). Learning Bayesian Network Structure from Massive Datasets: the "Sparse Candidate" Algorithm. In *UAI*.

Genest, C., Gendron, M., and Bourdeau-Brien, M. (2009). The Advent of Copulas in Finance. *The European Journal of Finance*, 15(7–8):609–618.

Geyer, C. J. (1992). Practical Markov Chain Monte Carlo. *Statistical Science*, 7(4):473–483.

Ghahramani, Z. and Hinton, G. E. (1996). The EM Algorithm for Mixtures of Factor Analyzers. Technical Report CRG-TR-96-1, Department of Computer Science, University of Toronto.

Girolami, M. and Calderhead, B. (2011). Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214.

Han, H. (2010). Asymptotic Properties of GARCH-X Processes. Technical report, National University of Singapore.

Harmeling, S. and Williams, C. K. I. (2011). Greedy Learning of Binary Latent Trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(6):1087–1097.

Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning*. Springer, New York.

Heckerman, D., Meek, C., and Cooper, G. F. (1999). A Bayesian Approach to Causal Discovery. In Glymour, C. and Cooper, G. F., editors, *Computation, Causation, and Discovery*. MIT.

Hoerl, A. E. and Kennard, R. W. (1970). Ridge Regression: Biased Estimation for Nonorthogonal Problems. *Technometrics*, 12(1):55–67.

Hoff, P. D. (2007). Extending the Rank Likelihood for Semiparametric Copula Estimation. *The Annals of Applied Statistics*, 1(1):265–283.

Hsieh, C.-J., Sustik, M. A., Dhillon, I., Ravikumar, P., and Poldrack, R. (2013). BIG & QUIC: Sparse Inverse Covariance Estimation for a Million Variables. In Burges, C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 26*, pages 3165–3173.

Hsieh, C.-J., Sustik, M. A., Dhillon, I. S., and Ravikumar, P. (2011). Sparse Inverse Covariance Matrix Estimation Using Quadratic Approximation. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 2330–2338.

Jacobs, R. A., Jordan, M. I., Nowlan, S. J., and Hinton, G. E. (1991). Adaptive Mixtures of Local Experts. *Neural Computation*, 3:79–87.

Jones, B., Carvalho, C., Dobra, A., Hans, C., Carter, C., and West, M. (2005). Experiments in Stochastic Computation for High-Dimensional Graphical Models. *Statistical Science*, 20(4):388–400.

Kalaitzis, A., Lafferty, J., Lawrence, N., and Zhou, S. (2013). The Bigraphical Lasso. In Dasgupta, S. and McAllester, D., editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1229–1237. JMLR Workshop and Conference Proceedings.

Kalaitzis, A. and Lawrence, N. (2012). Residual Component Analysis: Generalising PCA for more Flexible Inference in Linear-Gaussian Models. In Langford, J. and Pineau, J., editors, *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pages 209–216. Omnipress.

Kemp, C. and Tenenbaum, J. B. (2008). The Discovery of Structural Form. *Proceedings of the National Academy of Sciences*, 105(31):10687–10692.

Krishnapuram, B., Carin, L., Figueiredo, M. A. T., and Hartemink, A. (2005). Sparse Multinomial Logistic Regression: Fast Algorithms and Generalization Bounds. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(6):957–968.

Krumsiek, J., Suhre, K., Illig, T., Adamski, J., and Theis, F. (2011). Gaussian Graphical Modeling Reconstructs Pathway Reactions from High-Throughput Metabolomics Data. *BMC systems biology*, 5(1):21.

Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001). Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289. Morgan Kaufmann.

Lake, B. M. and Tenenbaum, J. B. (2010). Discovering Structure by Learning Sparse Graphs. In *Proceedings of the 33rd Annual Cognitive Science Conference*.

Lawrence, N. (2005). Probabilistic Non-Linear Principal Component Analysis with Gaussian Process Latent Variable Models. *Journal of Machine Learning Research*, 6:1783–1816.

Ledoit, O. and Wolf, M. (2003). Improved Estimation of the Covariance Matrix of Stock Returns with an Application to Portfolio Selection. *Journal of Empirical Finance*, 10(5):603–621.

Lenkoski, A. (2013). A Direct Sampler for G-Wishart Variates. *Stat*, 2(1):119–128.

Lenkoski, A. and Dobra, A. (2011). Computational Aspects Related to Inference in Gaussian Graphical Models with the G-Wishart Prior. *Journal of Computational and Graphical Statistics*, 20(1):140–157.

Levina, E., Rothman, A., and Zhu, J. (2008). Sparse Estimation of Large Covariance Matrices via a Nested Lasso Penalty. *The Annals of Applied Statistics*, 2(1):245–263.

Liang, F. (2010). A Double Metropolis-Hastings Sampler for Spatial Models with Intractable Normalizing Constants. *Journal of Statistical Computation and Simulation*, 80(9):1007–1022.

Liu, C. and Rubin, D. B. (1994). The ECME Algorithm: A Simple Extension of EM and ECM with Faster Monotone Convergence. *Biometrika*, 81(4):633–648.

Liu, H., Chen, X., Lafferty, J., and Wasserman, L. (2010). Graph-Valued Regression. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1423–1431.

Liu, H., Lafferty, J., and Wasserman, L. (2009). The Nonparanormal: Semiparametric Estimation of High Dimensional Undirected Graphs. *J. Mach. Learn. Res.*, 10:2295–2328.

Low, R. K. Y., Alcock, J., Faff, R., and Brailsford, T. (2013). Canonical Vine Copulas in the Context of Modern Portfolio Management: Are They Worth It? *Journal of Banking and Finance*, 37(8):3085–3099.

Maathius, M. and Kalisch, M. Buhlmann, P. (2009). Estimating High-Dimensional Intervention Effects from Observation Data. *The Annals of Statistics*, 37:3133–3164.

Mallat, S. (2008). *A Wavelet Tour of Signal Processing: The Sparse Way*. Academic Press.

Meinshausen, N. and Bühlmann, P. (2006). High-Dimensional Graphs and Variable Selection with the Lasso. *The Annals of Statistics*, 34(3):1436–1462.

Mitsakakis, N., Massam, H., and D Escobar, M. (2011). A Metropolis-Hastings Based Method for Sampling from the G-Wishart Distribution in Gaussian Graphical Models. *Electronic Journal of Statistics*, 5:18–30.

Mohamed, S., Heller, K., and Ghahramani, Z. (2012). Bayesian and L1 Approaches for Sparse Unsupervised Learning. In Langford, J. and Pineau, J., editors, *Proceedings of the 29th International Conference on Machine Learning*, ICML '12, pages 751–758, New York, NY, USA. Omnipress.

Mohammadi, A. and Wit, E. (2012). Gaussian Graphical Model Determination based on Birth-Death MCMC Inference. *ArXiv e-prints*, 1210.5371.

Murray, I. and Adams, R. (2010). Slice Sampling Covariance Hyperparameters of Latent Gaussian Models. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 1732–1740.

Murray, I., Ghahramani, Z., and MacKay, D. (2006). MCMC for Doubly-Intractable Distributions. In *Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 359–366. AUAI Press.

Neal, R. M. (1993). Probabilistic Inference Using Markov Chain Monte Carlo Methods. Technical Report CRG-TR-93-1, University of Toronto.

Neal, R. M. (1996). Sampling from Multimodal Distributions using Tempered Transitions. *Statistics and Computing*, 6(4):353–366.

Neal, R. M. (2010). MCMC using Hamiltonian Dynamics in S. Brooks, A. Gelman, G. Jones, and X. Meng (Ed.), Handbook of Markov Chain Monte Carlo, Chapman & Hall.

Nelsen, R. B. (2006). *An Introduction to Copulas*. Springer.

Orchard, P. R., Agakov, F. V., and Storkey, A. (2013). Bayesian Inference in Sparse Gaussian Graphical Models. *ArXiv e-prints*, 1309.7311.

Patton, A. J. (2009). Copula-Based Models for Financial Time Series. In Mikosch, T., Kreiß, J.-P., Davis, R. A., and Andersen, T. G., editors, *Handbook of Financial Time Series*, pages 767–785. Springer Berlin Heidelberg.

Petersen, K. B. and Pedersen, M. S. (2006). The Matrix Cookbook.

Piccioni, M. (2000). Independence Structure of Natural Conjugate Densities to Exponential Families and the Gibbs' Sampler. *Scandinavian Journal of Statistics*, 27:111–127.

Pickands, III, J. (1975). Statistical Inference using Extreme Order Statistics. *The Annals of Statistics*, 3(1):119–131.

Preis, T., Kenett, D. Y., Stanley, H. E., Helbing, D., and Ben-Jacob, E. (2012). Quantifying the Behavior of Stock Correlations Under Market Stress. *Scientific Reports*, 2(752).

Rasmussen, C. E. and Ghahramani, Z. (2001). Infinite Mixtures of Gaussian Process Experts. In *Advances in Neural Information Processing Systems 14*, pages 881–888. MIT Press.

Reboredo, J. C. (2011). How Do Crude Oil Prices Co-Move?: A Copula Approach. *Energy Economics*, 33(5):948–955.

Rodriguez, A., Lenkoski, A., and Dobra, A. (2011). Sparse Covariance Estimation in Heterogeneous Samples. *Electronic Journal of Statistics*, 5:981–1014.

Rolfs, B., Rajaratnam, B., Guillot, D., Wong, I., and Maleki, A. (2012). Iterative Thresholding Algorithm for Sparse Inverse Covariance Estimation. In *Advances in Neural Information Processing Systems 25*, pages 1583–1591.

Rothman, A. J., Levina, E., and Zhu, J. (2010). Sparse Multivariate Regression With Covariance Estimation. *Journal of Computational and Graphical Statistics*, 19(4):947–962.

Roverato, A. (2002). Hyper Inverse Wishart Distribution for Non-Decomposable Graphs and its Application to Bayesian Inference for Gaussian Graphical Models. *Scandinavian Journal of Statistics*, 29(3):391–411.

Salakhutdinov, R. (2010). Learning Deep Boltzmann Machines using Adaptive MCMC. In Fürnkranz, J. and Joachims, T., editors, *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 943–950, Haifa, Israel. Omnipress.

Scheinberg, K., Ma, S., and Goldfarb, D. (2010). Sparse Inverse Covariance Selection via Alternating Linearization Methods. In Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2101–2109.

Schmidt, M., Fung, G., and Rosales, R. (2007). Fast Optimization Methods for L1 Regularization: A Comparative Study and Two New Approaches. In Kok, J., Koronacki, J., Mantaras, R., Matwin, S., Mladenic, D., and Skowron, A., editors, *Machine Learning: ECML 2007*, volume 4701 of *Lecture Notes in Computer Science*, pages 286–297. Springer Berlin / Heidelberg.

Schmidt, M., van den Berg, E., Friedlander, M., and Murphy, K. (2009). Optimizing Costly Functions with Simple Constraints: A Limited-Memory Projected Quasi-Newton Algorithm. In *AISTATS*, volume 5, pages 456–463.

Silva, R., Scheines, R., Glymour, C., and Spirtes, P. (2006). Learning the Structure of Linear Latent Variable Models. *JMLR*, 7.

Snelson, E., Ghahramani, Z., and Rasmussen, C. E. (2004). Warped Gaussian Processes. In Thrun, S., Saul, L., and Schölkopf, B., editors, *Advances in Neural Information Processing Systems 16*, pages 337–344. MIT Press.

Sokolovska, N., Lavergne, T., Cappe, O., and Yvon, F. (2010). Efficient Learning of Sparse Conditional Random Fields for Supervised Sequence Labeling. *IEEE Journal of Selected Topics in Signal Processing*, 4(6):953–964.

Städler, N. and Bühlmann, P. (2012). Missing Values: Sparse Inverse Covariance Estimation and an Extension to Sparse Regression. *Statistics and Computing*, 22(1):219–235.

Stephens, M. (2000). Bayesian Analysis of Mixture Models with an Unknown Number of Components – An Alternative to Reversible Jump Methods. *The Annals of Statistics*, 28(1):40–74.

Sucarrat, G., Grønneberg, S., and Escribano, Á. (2013). Estimation and Inference in Univariate and Multivariate Log-GARCH-X Models when the Conditional Density is Unknown.

Sutton, C., McCallum, A., and Rohanimanesh, K. (2007). Dynamic Conditional Random Fields: Factorized Probabilistic Models for Labeling and Segmenting Sequence Data. *Journal of Machine Learning Research*, 8:693–723.

Tibshirani, R. (1996). Regression Shrinkage and Selection via the Lasso. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 58(1):267–288.

Tipping, M. E. and Bishop, C. M. (1999). Probabilistic Principal Component Analysis. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 61(3):611–622.

Toh, K., Todd, M., and Tütüncü, R. (1999). SDPT3 – A Matlab Software Package for Semidefinite Programming. *Optimization Methods and Software*, 11:545–581.

Wang, C., Sun, D., and Toh, K.-C. (2010). Solving Log-Determinant Optimization Problems by a Newton-CG Primal Proximal Point Algorithm. *SIAM Journal on Optimization*, 20:2994–3013.

Wang, F. and Landau, D. P. (2001). Efficient, Multiple-Range Random Walk Algorithm to Calculate the Density of States. *Physical Review Letters*, 86:2050–2053.

Wang, H. and Carvalho, C. M. (2010). Simulation of Hyper-Inverse Wishart Distributions for Non-Decomposable Graphs. *Electronic Journal of Statistics*, 4:1470–1475.

Wang, H. and Li, S. Z. (2012). Efficient Gaussian Graphical Model Determination Under G-Wishart Prior Distributions. *Electronic Journal of Statistics*, 6:168–198.

Wang, Y.-C., Wu, J.-L., and Lai, Y.-H. (2013). A Revisit to the Dependence Structure between the Stock and Foreign Exchange Markets: A Dependence-Switching Copula Approach. *Journal of Banking and Finance*, 37(5):1706–1719.

Wilson, A. and Ghahramani, Z. (2010). Copula Processes. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2460–2468.

Wong, F., Carter, C. K., and Kohn, R. (2003). Efficient Estimation of Covariance Selection Models. *Biometrika*, 90(4):809–830.

Yuan, M. and Lin, Y. (2006). Model Selection and Estimation in Regression with Grouped Variables. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(1):49–67.

Zhang, N. L. (2004). Hierarchical Latent Class Models for Cluster Analysis. *Journal of Machine Learning Research*, 5:697–723.

Zhang, Y. and Schneider, J. (2010). Learning Multiple Tasks with a Sparse Matrix-Normal Penalty. In Lafferty, J., Williams, C. K. I., Shawe-Taylor, J., Zemel, R., and Culotta, A., editors, *Advances in Neural Information Processing Systems 23*, pages 2550–2558.

Zou, H. and Hastie, T. (2005). Regularization and Variable Selection via the Elastic Net. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320.