Imperial College London

Imperial College London

Department of Mathematics

Regularised Jump Models for Regime Identification and Feature Selection

Author: Edward SELIG (CID: 02442425)

A thesis submitted for the degree of

MSc in Mathematics and Finance, 2023-2024

Declaration

The work contained in this thesis is my own work unless otherwise stated.

Acknowledgements

I would like to extend my heartfelt thanks to Dr. Paul Bilokon for his invaluable help and support throughout the writing of this thesis. I'd also like to thank the Fixed Income Quantitative Investments team at Schroders for proposing such an interesting project and providing regular feedback along the way.

Finally, I wish to express my sincere gratitude to all those who offered emotional support throughout my MSc course: the classmates I befriended, my brother Alex, and my parents back home in Sydney, Australia. Words cannot adequately express how grateful I am to you all.

Abstract

A regime modelling framework can be employed to address the complexities of financial markets. Under the framework, market periods are grouped into distinct regimes, each distinguished by similar statistical characteristics. Regimes in financial markets are not directly observable but are often manifested in market and macroeconomic variables. The objective of regime modelling is to accurately identify the active regime from these variables at a point in time, a process known as *regime identification*.

One way to enhance the accuracy of regime identification is to select features that are most responsible for statistical differences between regimes, a process known as *feature selection*. Feature selection is also capable of both enhancing the interpretability of outputs from regime models, and substantially reducing the computational time required to calibrate regime models.

Models based on the Jump Model framework have recently been developed to address the joint problem of regime identification and feature selection. In the following work, we propose a new set of models called *Regularised Jump Models* that are founded upon the Jump Model framework.

These models perform feature selection that is more interpretable than that from the Sparse Jump Model, a model proposed in the literature pertaining to the Jump Model framework. Through a simulation experiment, we find evidence that these new models outperform the Standard and Sparse Jump Models, both in terms of regime identification and feature selection.

Additionally, we show in an empirical study that the new models inform Credit Default Swap (CDS) trading strategies with risk-adjusted returns superior to those that are informed by existing models.

Contents

1 Jump Model Framework **8**
- 1.1 Rationale . . . 8
- 1.2 Notation and Setup . . . 8
- 1.3 Regression Models . . . 9
  - 1.3.1 Single Model . . . 9
  - 1.3.2 K-Models . . . 9
- 1.4 Jump Model . . . 10
  - 1.4.1 Probabilistic Interpretation . . . 11
- 1.5 Algorithms . . . 12
  - 1.5.1 Model Calibration . . . 12
  - 1.5.2 Inference . . . 13

2 Jump Models for Regime Modelling and Feature Selection **16**
- 2.1 Introduction to Clustering . . . 16
- 2.2 Standard Jump Model . . . 17
- 2.3 Sparse Jump Model . . . 18
- 2.4 Continuous Jump Model . . . 19
- 2.5 Regularised Jump Models . . . 20
  - 2.5.1 Regularised K-Means . . . 20
  - 2.5.2 Regularised Jump Model Equation . . . 22
  - 2.5.3 Comparison with Sparse Jump Models . . . 22

3 Calibration of Jump Models **24**
- 3.1 Standard Jump Model Calibration . . . 24
- 3.2 Sparse Jump Model Calibration . . . 25
- 3.3 Regularised Jump Model Calibration . . . 26
- 3.4 Hyperparameter Tuning . . . 28
- 3.5 Simulation Study . . . 29
  - 3.5.1 Setup . . . 30
  - 3.5.2 Model Tuning and Fitting . . . 30
  - 3.5.3 Evaluation of Model Accuracy . . . 31
  - 3.5.4 Results . . . 31
  - 3.5.5 Observations of Results . . . 32

4 Empirical Study **36**
- 4.1 Data Pre-Processing . . . 36
- 4.2 Backtest Methodology . . . 37
  - 4.2.1 CDS Data and Return Estimation . . . 37
  - 4.2.2 Model Training and Online Learning of Market Regimes . . . 37
  - 4.2.3 Trading Strategies . . . 39
- 4.3 Backtest Results . . . 40

4.3.1 Strategy Performance and Benchmarking 41
4.3.2 Feature Selection 44

A Technical Proofs and Supplementary Material **49**
A.1 Parameter Update Equations for Regularised Jump Model Calibration . . . 49
A.2 Introduction to Credit Default Swaps 50
A.2.1 Index CDS 51
A.3 Data Series for Empirical Calibration 52
A.4 Calculation of Performance Metrics 53
A.5 Jump Model State Estimates 54
A.6 Empirical Transition Probability Matrices 55

Bibliography **57**

List of Figures

3.1 Average BAC of Regularised Jump Models as a function of the jump penalty λ, for various combinations of time lengths T and regularisation parameter values γ. 32

4.1 EUR iTraxx Main 5Y Spreads between October 2011 and August 2024. . . 37
4.2 Process for training of Features Dataset. 39
4.3 EUR iTraxx Main 5Y Spreads shaded with state estimates from the Regularised $\mathcal{P}_3$ Jump Model. The states were estimated using the methodology visualised in Figure 4.2. 40
4.4 Total returns and drawdowns of Strategy 1. 42
4.5 Total returns and drawdowns of Strategy 2. 43
4.6 Total returns and drawdowns of Strategy 1 and its benchmark. 44
4.7 Total returns and drawdowns of Strategy 2 and its benchmark. 44
4.8 Feature category weights from the Sparse Jump Model. 46
4.9 Feature category weights from the Regularised $\mathcal{P}_0$ Jump Model. 46
4.10 Feature category weights from the Regularised $\mathcal{P}_1$ Jump Model. 46
4.11 Feature category weights from the Regularised $\mathcal{P}_3$ Jump Model. 47

A.1 Payoff structure of a standard CDS contract. 51
A.2 EUR iTraxx Main 5Y Spreads shaded with empirical state estimates from each Jump Model. 54
A.3 Empirical transition probability matrices of states estimated from the Jump Models (matrix entries in 1.d.p.). 55

List of Tables

3.1 Jump Model hyperparameters. 28
3.2 Average BAC of state sequence (3 d.p.) for each Jump Model, across different values of μ and number of features P. Values in brackets are standard deviations (2 d.p.). 34
3.3 Average BAC of relevant features (3 d.p.) for each Jump Model, across different values of μ and number of features P. Values in brackets are standard deviations (2 d.p.). 35

4.1 Strategy 1 Performance Metrics (2.d.p.). 42
4.2 Strategy 2 Performance Metrics (2.d.p.). 43
4.3 Average weights (1.d.p.) for both strategies and all Jump Models. 43

Introduction

Financial markets are inherently complex due to characteristics such as its high-dimensionality, non-stationarity and low signal-to-noise ratio. The adoption of a regime framework is one approach to addressing such complexities, where a regime is defined as a period of persistent market conditions. The framework assumes that the dynamics of financial markets can be grouped into distinct regimes, each marked by similar statistical properties.

Regimes in financial markets are not directly observable but are often manifested in variables such as the returns of certain asset classes. For instance, equities tend to outperform bonds during a bull market, a regime characterised by rising prices and investor optimism. Conversely, in a bear market, marked by falling prices and investor pessimism, bonds tend to outperform equities.

This stylised fact of financial markets has subsequently motivated academics and industry practitioners to develop latent variable models for observable market processes, where the prevailing market regime is designated the latent variable. These models are called regime-switching models.

The seminal work in regime-switching models was [15] in which a Hidden Markov Model (HMM) was developed and applied to identify expansions and recessions in the US economic cycle. Works such as [15] have demonstrated that financial markets can be parsimoniously described by regime-switching models and that these models can capture stylised behaviours of asset returns such as fat tails, volatility clustering, skewness and time-varying correlations.

The Hidden Markov Model is a popular regime-switching model that relies on specific assumptions of the observable process/processes and the latent state variable influencing it/them. Multiple extensions of the HMM have subsequently been proposed that relax the assumptions of the standard HMM (see [13],[30] and [9] for examples of HMM extensions). More recently, path signatures have been used to detect market regimes ([18] and [17]).

[5] introduced the Jump Model framework, the main area of study. The key idea of the framework is to alternate between minimising a loss function to fit multiple model parameters, and minimising a discrete loss function to determine which set of model parameters is active at each point in time. The paper proves the Hidden Markov Model to be a special case of the Jump Model.

In general, the application of regime-switching models involves the extraction of two quantities for a given observation sequence $\boldsymbol{Y} := (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)^{'} \in \mathbb{R}^{T \times p}$:

State Estimation: The corresponding state sequence $S := (s_1, s_2, \ldots, s_T)$ that best "ex-

plains" the observation sequence $\boldsymbol{Y}$ [24].

Parameter Estimation: The parameters of the conditional distribution $\boldsymbol{Y}_t|s_t,\ t = 1, 2, \ldots T$.

$\boldsymbol{Y}$ is defined as a sequence of T observations, each observation consisting of p features. When modelling high-dimensional datasets, it is often useful to reduce the number of features by determining which features are most relevant, a process known as *feature selection*. The application of feature selection techniques can be useful for identifying features that are either redundant or irrelevant, and can thus be removed without incurring a significant loss of information in the training dataset. It is most common in supervised learning applications such as regression, where labeled training data is available to determine the relevance of features.

However, feature selection is also important for unsupervised learning applications such as clustering and regime-switching models, where the training data is unlabeled because of the hidden nature of market regimes. Feature selection can yield interpretable results by establishing which features are most influential in differences between market regimes. Furthermore, feature selection can also mitigate issues associated with the application of unsupervised learning algorithms to high-dimensional data, an instance of what is commonly known as the "curse of dimensionality".

In Chapter 1, the Jump Model framework is introduced and elaborated. Chapter 2 outlines existing models within the Jump Model framework and proposes a new set of models consistent with the framework; these new models are called *Regularised Jump Models*. The chapter explains how these models are used for joint feature selection, parameter estimation and state-sequence estimation. Henceforth, we label models that are consistent with the Jump Model framework "Jump Models".

Chapter 3 investigates the techniques used to fit Jump Models and tests these techniques in a simulation experiment. In Chapter 4, the Jump Models are backtested in strategies that trade a Credit Default Swap (CDS) contract.

Our original contributions are as follows:

- Proposal of a new set of Jump Models called *Regularised Jump Models*.
- Development of a new hyperparameter tuning method for Jump Models based on *clustering stability*.
- Addition of a step to the online learning algorithm in [21] that updates the parameters of a given Jump Model.
- Calibration of Jump Models onto a dataset containing market and macroeconomic features.

Chapter 1

Jump Model Framework

The chapter introduces the Jump Model framework proposed in [5]. Due to the generality of the framework, a condensed introduction is given by outlining the theory most relevant to regime-switching models.

1.1 Rationale

Multiple statistical models, both supervised and unsupervised, are applied to observations with a timestamp, better known as time-series data. However, when calibrating these models onto time-series datasets, the sequential nature of the dataset is not taken into account.

A simple example of this is linear regression solved by Ordinary Least Squares (OLS). In the minimisation problem $\min_{\beta \in \mathbb{R}^p} \|Y - X\beta\|^2$, where $\|\cdot\|$ is the $\mathcal{L}_2$ norm, $X \in \mathbb{R}^{n \times p}$ is the design matrix and $Y \in \mathbb{R}^n$ is the output vector, each row of X and Y is associated with a data point, yet the solution θ^* is invariant to the order of the rows of X and Y.

On the other hand, as noted by [5], there are many applications in which relevant information is contained not only in data values but also in their temporal order. More specifically, if a data-point's corresponding time stamp is taken into account, one can detect changes in the regime or *mode* that generated the data. Sample applications include video segmentation [10] and speech recognition [24]. These applications are characterised by the dual objectives of fitting multiple models and detecting if any switches between these models have occurred.

In [5], Jump Models are introduced and used to fit a temporal sequence of data, taking into account the ordering of the data. The proposed fitting algorithm alternates between two steps: estimating the parameters of multiple models and estimating the temporal sequence of model activations.

1.2 Notation and Setup

Denote $Y := (y_1, y_2, \ldots, y_T)$ a training sequence of observations where $y_t \in \mathcal{Y}$ for all $t = 1, 2, \ldots, T$. The training sequence is used to build a regression model that provides a prediction $\hat{y}_t$ given some exogenous inputs $x_1, x_2, \ldots, x_t$ and possibly past observations $y_1, y_2, \ldots, y_{t-1}$. We are specifically interested in models where $\hat{y}_t$ is not simply a static function of x_t, but those models that exploit the additional information embedded in the temporal ordering of the data.

[5] shows that the regression model is implicitly defined by the minimisation of a loss function J that depends on the inputs $x_1, x_2, \dots x_t$, past observations $y_1, y_2, \dots y_{t-1}$ and other parameters. The chosen configuration of J determines the regression model. Propositions 1.4.1 and 1.4.2 establish this correspondence.

The quality of the regression model over a time period $t = 1, 2, \dots, T$ is quantified by the *average loss*

$$L = \frac{1}{T} \sum_{t=1}^{T} \ell\left(\hat{y}_t, y_t\right), \tag{1.2.1}$$

where $\ell : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}$ penalises the mismatch between y_t and $\hat{y}_t$, with $\ell(y, y) = 0$ for all $y \in \mathcal{Y}$.

1.3 Regression Models

1.3.1 Single Model

We introduce a model parameter $\theta \in \mathbb{R}^d$, a loss function $\ell : \mathcal{X} \times \mathcal{Y} \times \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ and a regulariser $r : \mathbb{R}^d \to \mathbb{R} \cup \{+\infty\}$ that together define the fitting objective

$$J(X, Y, \theta) = \sum_{t=1}^{T} \ell\left(x_t, y_t, \theta\right) + r(\theta), \tag{1.3.1}$$

where $X = (x_1, x_2, \dots, x_T)$ and $Y = (y_1, y_2, \dots, y_T)$. Additionally, let $x_t \in \mathcal{X} = \mathbb{R}^d, y_t \in \mathcal{Y} = \mathbb{R}$ for $t = 1, 2, \dots, T$. We define the optimal model parameter θ^* as

$$\theta^* = \arg\min_{\theta} J(X, Y, \theta). \tag{1.3.2}$$

By fixing $\theta = \theta^*$ and exploiting the separability of the loss J in 1.3.1, we get the regression model

$$\begin{aligned} \hat{y}_t &= \arg\min_{y \in \mathcal{Y}} J(X, Y, \theta^*) \\ &= \arg\min_{y \in \mathcal{Y}} \ell(X, Y, \theta^*) \\ &=: \psi(x_t) \end{aligned} \tag{1.3.3}$$

where $\psi : \mathcal{X} \to \mathcal{Y}$ is the function defining the regression model. For example, when $\ell(x_t, y, \theta) = \|y - \theta x_t\|_2^2$, we obtain the OLS linear regression model $\psi(x_t) = \theta^T x_t$.

1.3.2 K-Models

We extend the previous model and introduce multiple model parameters $\theta_s \in \mathbb{R}^d, s \in \{1, 2, \dots, K\}$ and a latent *mode* variable s_t that determines which set of model parameters is active at time t. Fitting a K-model on a training dataset (X, Y) entails minimising the fitting loss function

$$J(X, Y, \Theta, S) = \sum_{t=1}^{T} \ell\left(x_t, y_t, \theta_{s_t}\right) + \sum_{k=1}^{K} r\left(\theta_k\right), \tag{1.3.4}$$

with respect to $\Theta = (\theta_1, \theta_2, \ldots, \theta_K)$ and $S = (s_1, s_2, \ldots, s_T)$. The optimal parameters $\Theta^* = (\theta_1^*, \theta_2^*, \ldots, \theta_K^*)$ are used to define the K-model

$$(\hat{y}_t, \hat{s}_t) = \arg\min_{y,s} \ell\left(x_t, y_t, \theta_s^*\right). \tag{1.3.5}$$

The objective function in (1.3.4) is used to estimate the set of optimal model parameters Θ^* using the entire training dataset (X, Y). 1.3.5 infers the output $\hat{y}_t$ and discrete state s_t given the input x_t and Θ^*.

We note that the notational setup given up to this point is inclined towards supervised learning problems such as regression, since it includes an input variable x_t and output variable y_t. Given that the specific application in this work is market regime modelling, an unsupervised learning problem, the input variable x_t and set $\mathcal{X}$ are omitted henceforth.

1.4 Jump Model

The abovementioned models do not take into account the ordering of the data points $(y_1, y_2, \ldots, y_T)$. To address this, a *mode sequence loss*, denoted $\mathcal{L}$, is added to the fitting objective J:

$$J\left(Y, \Theta, S\right) = \sum_{t=1}^{T} \ell\left(y_t, \theta_{s_t}\right) + \sum_{k=1}^{K} r\left(\theta_k\right) + \mathcal{L}\left(S\right), \tag{1.4.1}$$

where $S = (s_0, s_1, \ldots s_T)$ is the mode or state sequence. $\mathcal{L} : \mathcal{K}^{T+1} \to \mathbb{R} \cup \{+\infty\}$ is defined as

$$\mathcal{L}\left(S\right) = \mathcal{L}^{\text{init}}\left(s_0\right) + \sum_{t=1}^{T} \mathcal{L}^{\text{mode}}\left(s_t\right) + \sum_{t=1}^{T} \mathcal{L}^{\text{trans}}\left(s_t, s_{t-1}\right), \tag{1.4.2}$$

where $\mathcal{K} := \{1, 2, \ldots, K\}$, $\mathcal{L}^{\text{init}} : \mathcal{K} \to \mathbb{R} \cup \{+\infty\}$ is the *initial mode cost*, $\mathcal{L}^{\text{mode}} : \mathcal{K} \to \mathbb{R} \cup \{+\infty\}$ is the *mode cost*, and $\mathcal{L}^{\text{trans}} : \mathcal{K}^2 \to \mathbb{R} \cup \{+\infty\}$ is the *mode transition cost*.

The choice of $\mathcal{L}^{\text{init}}, \mathcal{L}^{\text{mode}}$ and $\mathcal{L}^{\text{trans}}$ should strike a balance between fitting the data and incorporating prior assumptions held about the models parametrised by Θ and the mode sequence S.

Firstly, $\mathcal{L}\left(S\right) \equiv 0$ reduces to the K-model introduced in Subsection 1.3.2. If we choose $\mathcal{L}^{\text{trans}}\left(i, j\right) = \lambda$, for all $i \neq j$ and $\mathcal{L}^{\text{mode}}\left(i\right) = \mathcal{L}^{\text{trans}}\left(i, i\right) = 0$, mode transitions are penalised equally by a constant $\lambda \geq 0$.

$\lambda \to \infty$ leads to regression of a single model since mode transitions from s_0 become prohibitively expensive for the fitting loss function J in (1.4.1). $\lambda \to 0$ leads again to the K-model as there is no associated cost of mode transitions.

[5] noted that the choice of a constant transition cost λ leads to multiple solutions for S as indexes i, j can be arbitrarily permuted. The mode loss $\mathcal{L}^{\text{mode}}$ can then be used to break such symmetries. For example, smaller values for s_t will be preferred by making $\mathcal{L}^{\text{mode}}\left(i\right) < \mathcal{L}^{\text{mode}}\left(j\right)$ for $i < j$. The initial mode cost $\mathcal{L}^{\text{init}}$ can be used to incorporate prior knowledge of the initial mode s_0. For example, no prior knowledge can be incorporated by setting $\mathcal{L}^{\text{init}} \equiv 0$.

1.4.1 Probabilistic Interpretation

Let $Y = (y_1, y_2, \ldots, y_T), S = (s_0, s_1, \ldots, s_T)$ and $\Theta = (\theta_1, \theta_2, \ldots, \theta_K)$. A probabilistic interpretation of the loss function defined in (1.4.1) is provided by using the following modelling assumptions:

A1. The mode sequence S and the model parameters Θ are independent:

$$\mathbb{P}(S, \Theta) = \mathbb{P}(S)\,\mathbb{P}(\Theta)$$

A2. The conditional likelihood of Y is given by

$$\mathbb{P}(Y|S, \Theta) = \prod_{t=1}^{T} \mathbb{P}(y_t|S, \Theta) = \prod_{t=1}^{T} \mathbb{P}(y_t|\theta_{s_t}),$$

where $\mathbb{P}(y_t|\theta_{s_t})$ is the likelihood of the outcome y_t given θ_{s_t}.

A3. The priors on the model parameters $\theta_1, \theta_2, \ldots, \theta_K$ are all equal to $\mathbb{P}(\theta)$:

$$\mathbb{P}(\theta_1) = \mathbb{P}(\theta_2) = \cdots = \mathbb{P}(\theta_K) = \mathbb{P}(\theta).$$

Furthermore, the model parameters are independent i.e.,

$$\mathbb{P}(\Theta) = \prod_{k=1}^{K} \mathbb{P}(\theta_k).$$

A4. The probability of being in mode s_t given $s_0, s_1, \ldots, s_{t-1}$ is $\mathbb{P}(s_t|s_{t-1}) = \pi_{s_t, s_{t-1}}$ (Markov property):

A5. The initial mode s_0 has probability $\mathbb{P}(s_0) = \pi_{s_0}$.

Proposition 1.4.1. *Define*

$$\begin{aligned}
\ell(y_t, \theta_{s_t}) &= -\log \mathbb{P}(y_t|\theta_{s_t}) \\
r(\theta_k) &= -\log \mathbb{P}(\theta_k) \\
\mathcal{L}^{trans}(s_t, s_{t-1}) &= -\log \pi_{s_t, s_{t-1}} \\
\mathcal{L}^{init}(s_0) &= -\log \pi_{s_0} \\
\mathcal{L}^{mode}(s_0) &= 0.
\end{aligned}$$

Then minimising $J(Y, \Theta, S)$, as defined in 1.4.1 and 1.4.2, with respect to Θ and S is equivalent to maximising the joint likelihood $\mathbb{P}(Y, S, \Theta)$.

Proof. Proof can be found in [5] Proposition 1. □

Proposition 1.4.2. *Define the probability density functions*

$$\begin{aligned}
\mathbb{P}(y_t|\theta_{s_t}) &= \frac{e^{-\ell(y_t, \theta_{s_t})}}{\nu(\theta_{s_t})}, \\
\mathbb{P}(S, \Theta) &= \frac{\nu(S, \Theta)\, e^{-\mathcal{L}(S) - r(\Theta)}}{\sum_{\bar{S} \in K^{T+1}} \int_{\mathbb{R}^{d \times K}} \nu\left(\bar{S}, \Theta\right) e^{-\mathcal{L}(S) - r(\Theta)} d\Theta},
\end{aligned}$$

where

$$\nu\left(\theta_{s_t}\right) = \int_{\mathcal{Y}} e^{-\ell\left(y,\theta_{s_t}\right)} dy,$$
$$\nu\left(S,\Theta\right) = \prod_{t=1}^{T} \nu\left(\theta_{s_t}\right).$$

Furthermore, assume that the outputs Y are conditionally independent given (S,Θ) i.e. $\mathbb{P}\left(Y|S,\Theta\right) = \prod_{t=1}^{T} \mathbb{P}\left(y_t|\theta_{s_t}\right)$. Then the following identity holds:

$$\arg\min_{S,\Theta} J\left(Y,S,\Theta\right) = \arg\max_{S,\Theta} \log \mathbb{P}\left(Y,S,\Theta\right).$$

Proof. Proof can be found in [5] Proposition 2. □

1.5 Algorithms

1.5.1 Model Calibration

The calibration of the Jump Model onto a training dataset $Y = (y_1, y_2, \ldots, y_T)$ requires the minimisation of $J\left(Y,\Theta,S\right)$ with respect to Θ and S. [5] devises a *coordinate descent* algorithm that alternates between minimisation with respect to Θ, and minimisation with respect to S.

In multivariable calculus, coordinate descent is an optimisation algorithm that optimises a multivariable function along one coordinate at a time, while keeping the other coordinates fixed. More specifically, for a function $f : \mathbb{R}^n \to \mathbb{R}$, starting with an initial guess $\boldsymbol{x}^0 := \left(x_1^0, x_2^0, \ldots, x_n^0\right)$, the $(k+1)^{\text{th}}$ iteration of the algorithm is defined recursively from the k^{th} iteration by solving the single variable optimisation problem

$$x_i^{(k+1)} = \arg\min_{x\in\mathbb{R}} f\left(x_1^{(k+1)}, x_2^{(k+1)}, \ldots, x_{i-1}^{(k+1)}, x, x_{i+1}^{(k)}, \ldots, x_n^{(k)}\right)$$

for each variable x_i of $\boldsymbol{x} := (x_1, x_2, \ldots, x_n)$ from 1 to n.

In the calibration of the Jump Model, the coordinate descent algorithm is performed by minimising $J\left(Y,\Theta,S\right)$ with respect to two sets of variables in alternation: the model parameters Θ and the state sequence S. $J\left(Y,\Theta,S\right)$ is minimised with respect to Θ while keeping S fixed, and then $J\left(Y,\Theta,S\right)$ is minimised with respect to S while keeping Θ fixed.

These two steps are repeated until the state sequence S does not change or the improvement in the value of the objective function J is smaller than a pre-determined tolerance level.

If ℓ and r are convex functions, Step (1.1) in Algorithm (1) can be solved globally using *standard convex programming*. Step (1.2) can be solved by discrete *dynamic programming*. Dynamic programming refers to the technique of determining an optimal decision by breaking it down into a sequence of decision steps such that the decision steps hold a recursive relationship.

Dynamic programming for Step (1.2) in Algorithm (1) is achieved by computing a matrix of costs $V \in \mathbb{R}^{K\times(T+1)}$ and indexes $U \in \mathcal{K} \times \mathbb{R}^T$. The below equations establish

the recursive computations computed backward in time for $t = T-1, T-2, \ldots, 1$:

$$V(s,T) = \mathcal{L}^{\text{mode}}(s) + \ell(y_T, \theta_s), \tag{1.5.1a}$$
$$U_{s,t} = \arg\min_j \{V(j, t+1) + \mathcal{L}^{\text{trans}}(j,s)\}, \tag{1.5.1b}$$
$$V(s,t) = \mathcal{L}^{\text{mode}}(s) + \ell(y_t, \theta_s) + V(U_{s,t}, t+1) + \mathcal{L}^{trans}(U_{s,t}, s), \tag{1.5.1c}$$
$$V(s,0) = \mathcal{L}^{\text{init}}(s) + \min_j \{V(j,1) + \mathcal{L}^{\text{trans}}(j,s)\}. \tag{1.5.1d}$$

Once the two matrices are calculated, the state sequence S is reconstructed forward in time by setting

$$s_0 = \arg\min_j V(j,0),$$
$$s_t = U_{s_{t-1},t}, \ t = 1, 2, \ldots, T.$$

Algorithm 1 Coordinate descent algorithm for jump model calibration in [5]

Input: Training dataset $Y = (y_1, y_2, \ldots, y_T)$, number of modes K and initial mode sequence $S^{(0)} := \left(s_0^{(0)}, s_1^{(0)}, \ldots, s_T^{(0)}\right)$.

Step 1: Iterate for $k = 1, 2, \ldots,$

$$\Theta^{(k)} \leftarrow \arg\min_{\Theta} \left\{ \sum_{t=1}^{T} \ell\left(y_t, \theta_{s_t^{(k-1)}}\right) + \sum_{k=1}^{K} r(\theta_k) \right\}; \tag{1.1}$$

$$S^{(k)} \leftarrow \arg\min_{S} \left\{ \sum_{t=1}^{T} \ell\left(y_t, \theta_{s_t}^{(k)}\right) + \mathcal{L}(S) \right\} \tag{1.2}$$

until $S^{(k)} = S^{(k-1)}$.

Output: Estimated model parameters $\Theta^* = \left(\theta_1^{(k)}, \theta_2^{(k)}, \ldots, \theta_K^{(k)}\right)$ and mode sequence $S^* = \left(s_0^{(k)}, s_1^{(k)}, \ldots, s_T^{(k)}\right)$.

We first note that, because $J(Y, \Theta, S)$ is non-increasing in terms of the number of iterations and that the number of possible state sequences S is finite, Algorithm (1) always terminates in a finite number of steps. However, there is no guarantee that the solution achieved by Algorithm (1) is the global one, as it depends on the initial guess $S^{(0)}$.

[5] suggests running Algorithm (1) multiple times from different random initial sequences and selecting the result that produces the lowest value for $J(Y, \Theta, S)$.

1.5.2 Inference

One-Step Ahead Prediction

Assume that the model parameters Θ^* have been estimated using past observations $\tilde{Y}_{1:t-1} := (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_{t-1})$. The same fitting objective (1.4.1) can be used to estimate y_t and $\hat{S}_{0:t} := (\hat{s}_0, \hat{s}_1, \ldots, \hat{s}_t)$:

$$\left(\hat{y}_t, \hat{S}_{0:t}\right) = \arg\min_{y, S_{0:t}} J_t\left(\tilde{Y}_{1:t-1}, y, \Theta^*, S_{0:t}\right), \ y_t \in \mathcal{Y}_t, \tag{1.5.2}$$

where $\mathcal{Y}_t \subseteq \mathcal{Y}$ is a possible output information set and

$$J_t\left(\tilde{Y}_{1:t-1}, y, \Theta^*, S_{0:t}\right) = \ell\left(y, \theta^*_{s_t}\right) + \sum_{j=1}^{t-1} \ell\left(\tilde{y}_j, \theta^*_{s_j}\right) + \mathcal{L}\left(S_{0:t}\right). \tag{1.5.3}$$

In (1.5.3), $\mathcal{L}\left(S_{0:t}\right)$ is the mode sequence loss for the state sequence $S_{0:t}$ and is given by

$$\mathcal{L}\left(S_{0:t}\right) = \mathcal{L}^{\text{init}}\left(s_0\right) + \sum_{j=1}^{t} \mathcal{L}^{\text{mode}}\left(s_j\right) + \sum_{j=1}^{t} \mathcal{L}^{\text{trans}}\left(s_j, s_{j-1}\right).$$

The algorithm for one-step ahead prediction is given in Algorithm 2. Step 2.1 is again solved by the DP iterations in 1.5.1 over the time span $[0, t]$ with the only difference that in (1.5.1a), the terminal cost is equal to $V\left(s, t\right) = \mathcal{L}^{\text{mode}}\left(s\right) + \min_y\{\ell\left(y, \theta_s\right)\}$ since the last output is determined in Step 2.2.

Algorithm 2 One-step ahead prediction in [5]

Input: Estimated model parameters Θ^* and past observations $\tilde{Y}_{1:t-1} := (\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_{t-1})$.

Step 1: Estimate state sequence $S_{0:t}$:

$$\hat{S}_{0:t} \leftarrow \underset{S_{0:t}}{\arg\min}\Big\{\mathcal{L}\left(S_{0:t}\right) + \sum_{j=1}^{t-1} \ell\left(\tilde{y}_j, \theta^*_{s_j}\right) + \min_{y \in \mathcal{Y}_t} \ell\left(y, \theta^*_{s_t}\right)\Big\}. \tag{2.1}$$

Step 2: Predict y_t:

$$\hat{y}_t \leftarrow \underset{y \in \mathcal{Y}_t}{\arg\min}\, \ell\left(y, \theta^*_{\hat{s}_t}\right). \tag{2.2}$$

Output: Estimated output $\hat{y}_t$ and mode sequence $\hat{S}_{0:t}$.

Recursive Inference and Online Learning

Algorithm 2 estimates the output y_t and the state sequence $S_{0:t} := (s_0, s_1, \ldots, s_t)$ in one run. This is an inefficient algorithm as the computation time of the algorithm grows linearly with the number of time points t. An incremental and more efficient version of the algorithm is shown in Algorithm 3.

In Algorithm 3, the *arrival cost* function $\mathcal{A}_t : \mathcal{K} \to \mathbb{R}$ is introduced and defined by the following recursive relation for $s \in \mathcal{K}$:

$$\begin{aligned}
&\mathcal{A}_0\left(s\right) = \mathcal{L}^{\text{init}}\left(s\right), \\
&\mathcal{A}_t\left(s\right) = \mathcal{L}^{\text{mode}}\left(s\right) + \min_{s' \in \mathcal{K}}\{\ell\left(\tilde{y}_{t-1}, \theta^*_{s'}\right) + \mathcal{A}_{t-1}(s') + \mathcal{L}^{\text{trans}}(s, s')\}.
\end{aligned} \tag{1.5.4}$$

The concept of the arrival cost is important in *online learning*, a method in which data is available in a sequential manner and is used to update the model at each step. Online learning contrasts with *offline learning*, also known as batch learning, the method in which a model is trained on the entire training dataset at once. Algorithms 2 and 3 are instances of offline and online learning respectively.

The arrival cost is the additional cost associated with the arrival of a new data point and the subsequent update of the model based on the new data point. In Algorithm 3, the prior arrival cost $\mathcal{A}_{t-1}$ is stored in memory as a vector of size $|\mathcal{K}|$. The arrival cost is updated using Step 3.1 and the updated arrival cost is then used in Step 3.2 to estimate (y_t, s_t).

We can see that, after replacing s and s' with s_t and s_{t-1} respectively, and expanding the right-hand side of the recursive relation in (1.5.4), we have that

$$\mathcal{A}_t(s_t) = \min_{S_{0:t-1}} \left\{ \sum_{j=1}^{t-1} \ell\left(\tilde{y}_j, \theta^*_{s_j}\right) + \mathcal{L}(S_{0:t}) \right\}. \tag{1.5.5}$$

,

The term minimised in (1.5.5) is very similar to the right-hand side of (1.5.3); the only difference between the two equations is $\ell\left(y, \theta^*_{s_t}\right)$ in (1.5.3). If we inspect Step 3.2 and use the identity in (1.5.5), we can see that Algorithm 2 and Algorithm 3 produce the same outputs. The difference between the two algorithms are their computational times: the computational time of Algorithm 2 is $\mathcal{O}(t)$ while for Algorithm 3, $\mathcal{O}(1)$.

Algorithm 3 Recursive inference in [5]

Input: Estimated model parameters Θ^*, past output $\tilde{y}_{t-1}$ and past arrival cost function $\mathcal{A}_{t-1}$.

Step 1: Update arrival cost function $\mathcal{A}_t : \mathcal{K} \to \mathbb{R}$ for each $s_t \in \mathcal{K}$:

$$\mathcal{A}_t(s_t) \leftarrow \mathcal{L}^{\text{mode}}(s_t) + \min_{s_{t-1}} \{\ell\left(\tilde{y}_{t-1}, \theta_{s_{t-1}}\right) + \mathcal{A}_{t-1}(s_{t-1}) + \mathcal{L}^{\text{trans}}(s_t, s_{t-1})\}. \tag{3.1}$$

Step 2: Estimate (y_t, s_t):

$$(\hat{y}_t, \hat{s}_t) \leftarrow \arg\min_{y \in \mathcal{Y}_t, s} \{\ell(y, \theta_s) + \mathcal{A}_t(s)\}. \tag{3.2}$$

Output: Estimated output y_t and mode s_t, and updated arrival cost function $\mathcal{A}_t$.

In case we have access to the output y_t and are only interested in estimating the mode s_t given $y_1, y_2, \ldots, y_t$, Step 3.2 in Algorithm 3 can be replaced with

$$\hat{s}_t = \arg\min_{s} \{\ell(y_t, \theta_s) + \mathcal{A}_t(s)\}. \tag{1.5.6}$$

Chapter 2

Jump Models for Regime Modelling and Feature Selection

The chapter outlines and examines the constituent models of the Jump Model framework. A new set of models called *Regularised Jump Models* are then introduced and compared to the existing models. Before presenting the constituent models, an introduction to the unsupervised learning technique of *clustering* is given to show its similarities with the estimation of state sequences under the Jump Model framework.

2.1 Introduction to Clustering

Clustering is the task of partitioning a dataset such that data points belonging to the same group or *cluster* are more similar (normally quantified by some pre-specified metric) to each other that to those in other groups. This task is very similar to the estimation of state sequences; one difference is that the temporal nature of the data is accounted for in state sequence estimation. In normal clustering applications, the data is assumed to be independent and identically distributed (IID).

Clustering comprises multiple algorithms that differ in terms of what defines a cluster and how they are found. In this work, we limit the scope of the introduction to *centroid models*, models that represent each cluster by the mean vector of the data points conditional on those points belonging to the same cluster. The mean vector of a cluster is sometimes known as the cluster centroid.

One of the most ubiquitous centroid models is the K-means clustering algorithm introduced in [19]. In the algorithm, data points are divided into K clusters in which each point belongs to the cluster with the closest centroid in terms of Euclidean distance. The algorithm achieves this through the minimisation of the total squared Euclidean distance between the data points and their corresponding cluster centroids, known shorthand as the within-cluster sum of squares (WCSS).

Before mathematically defining the WCSS, some clustering notation is introduced. Denote $\boldsymbol{Y} := (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$ a sequence of T observations where each $\boldsymbol{y}_t \in \mathbb{R}^p$ for $t = 1, 2, \ldots, T$. p represents the number of features and for a high-dimensional dataset $\boldsymbol{Y}$, we have that $p >> 1$. We assume that the columns of Y are standardised i.e. each feature has mean zero and variance one.

Assuming the number of clusters to be K, let $C = \{C_1, C_2, \ldots, C_K\}$ be a collection

of K disjoint sets of cluster indices satisfying $\cup_{k=1}^{K} C_k = \{1, 2, \ldots, T\}$. The corresponding matrix of cluster means is given by $\underset{\sim}{\boldsymbol{\mu}} := (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K)' \in \mathbb{R}^{K \times p}$. The within-cluster sum of squares (WCSS) is defined as

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2, \tag{2.1.1}$$

where $\|\cdot\|$ is the $\mathcal{L}_2$ norm. The K-means clustering algorithm minimises the WCSS with respect to the clusters $C_1, C_2, \ldots, C_K$ and their corresponding means $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$.

The difference between the total sum of squares (TSS), defined as the total squared Euclidean distance between the data points and the unconditional mean vector $\bar{\boldsymbol{\mu}}$ (equal to the zero vector under the assumption of standardised data), and the WCSS is known as the between-cluster sum of squares (BCSS):

$$\underbrace{\sum_{t=1}^{T} \|\boldsymbol{y}_t - \bar{\boldsymbol{\mu}}\|^2}_{\text{TSS}} = \underbrace{\sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2}_{\text{WCSS}} + \underbrace{\sum_{k=1}^{K} |C_k| \|\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}}\|^2}_{\text{BCSS}}, \tag{2.1.2}$$

where $|C_k|$ is the number of data points in the k^{th} cluster. The BCSS measures the dispersion of the clusters: a high BCSS indicates that the clusters are spread out, while a small value indicates that the clusters are close together. The TSS is a fixed quantity and so from the decomposition given in (2.1.2), the minimisation of the WCSS (the objective of K-means clustering) is equivalent to the maximisation of the BCSS.

Using an initial cluster assignment, the K-means clustering algorithm solves the minimisation of (2.1.1) by alternating between minimising (2.1.1) with respect to the cluster centres $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ while holding the clusters $C_1, C_2, \ldots, C_K$ fixed, and minimising (2.1.1) with respect to the clusters $C_1, C_2, \ldots, C_K$ while holding the cluster centres fixed. These two steps are repeated until the clusters remain unchanged after one iteration.

This is very similar to the coordinate descent algorithm introduced in 1 for Jump Models. The K-means clustering algorithm is given in Algorithm 4.

2.2 Standard Jump Model

Before introducing the Standard Jump Model, some notation is given in addition to that in Section 2.1. The state or mode sequence associated with the sequence of observations $\boldsymbol{Y}$ is represented by $(s_1, s_2, \ldots, s_T)$ where each $s_t \in \{1, 2, \ldots, K\}$ for $t = 1, 2, \ldots, T$.

K is the number of states and is assumed to be known. The K model parameters are given by $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ where each $\boldsymbol{\mu}_k \in \mathbb{R}^p$ for $k = 1, 2, \ldots, K$. The model parameters are the conditional means of the features assigned to each of the K states, hence the notational choice $\boldsymbol{\mu}_k, k = 1, 2, \ldots, K$.

Definition 2.2.1 (Standard Jump Model)**.** The *Standard Jump Model* with K states is defined by the minimisation of the objective function

$$\sum_{t=1}^{T-1} \left[\|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\|^2 + \lambda \mathbf{1}_{\{s_t \neq s_{t+1}\}} \right] + \|\boldsymbol{y}_T - \boldsymbol{\mu}_{s_T}\|^2 \tag{2.2.1}$$

Algorithm 4 K-means clustering algorithm in [19]

Input: Training dataset $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, assumed number of clusters K and initial clusters $C^{(0)} := \left(C_1^{(0)}, C_2^{(0)} \ldots, C_K^{(0)}\right)$

Step 1: Iterate for $j = 1, 2, \ldots,$

$$\underline{\boldsymbol{\mu}}^{(j)} \leftarrow \arg\min_{\underline{\boldsymbol{\mu}}} \sum_{k=1}^{K} \sum_{t \in C_k^{(j-1)}} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2, \tag{4.1}$$

$$C^{(j)} \leftarrow \arg\min_{C} \sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k^{(j)}\|^2, \tag{4.2}$$

until $C^{(j)} = C^{(j-1)}$.

Output: Estimated cluster centres $\underline{\boldsymbol{\mu}}^* := \left(\boldsymbol{\mu}_1^{(j)}, \boldsymbol{\mu}_2^{(j)}, \ldots, \boldsymbol{\mu}_K^{(j)}\right)'$ and clusters $\{C_1^{(j)}, C_2^{(j)}, \ldots, C_K^{(j)}\}$.

over the model parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ and state sequence $(s_1, s_2, \ldots, s_T)$. The term $\|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\|^2$ represents the squared $\mathcal{L}_2$-distance between the vectors $\boldsymbol{y}_t$ and $\boldsymbol{\mu}_{s_t}$ and $\lambda \geq 0$ is a hyperparameter.

The Standard Jump Model is a specific case of the general framework outlined in Chapter 1. Comparing the equations in (1.4.1), (1.4.2) and (2.2.1), the regularisation function term $r(\theta_k)$ is set to zero for all $k \in \{1, 2, \ldots, K\}$ and $\mathcal{L}(S) = \lambda \sum_{t=1}^{T-1} \mathbf{1}_{\{s_t \neq s_{t+1}\}}$ is the number of "jumps", or state transitions in the sequence S, multiplied by λ.

The objective function in (1.4.1) can be interpreted as a tradeoff between model fitting and prior assumptions about the tendency of the sequence S to "jump", or change states. This tendency is controlled by the hyperparameter λ. When $\lambda = 0$, the model reduces to splitting the dataset in at most K states and fitting one model per state, thereby generalising the K-means algorithm ([20]).

Furthermore, the objective function in (2.2.1) reduces to that of K-means clustering when $\lambda = 0$. This can be seen by rewriting the first and third summands in 2.2.1 as

$$\sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\|^2 = \sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2 \tag{2.2.2}$$

which is equal to the right-hand side of (2.1.1). Finally, for $\lambda \to \infty$, the Standard Jump Model results in a single-state model since the cost of changing states becomes prohibitive.

2.3 Sparse Jump Model

The Sparse Jump Model is an extension of the Standard Jump Model in its incorporation of feature selection. Let $\boldsymbol{w} := (w_1, w_2, \ldots, w_p)' \in \mathbb{R}^p$ denote a vector of feature weights that are assumed to be the same across all states.

Feature selection is incorporated in the Sparse Jump Model by employing the optimisation programme in [29]. The programme is given by

$$\begin{aligned} \max \quad & \boldsymbol{w}^{'}\left(\sum_{k=1}^{K}|C_k|\left(\boldsymbol{\mu}_k-\bar{\boldsymbol{\mu}}\right)^2\right), \\ \text{such that} \quad & \|\boldsymbol{w}\|^2 \leq 1, \quad \|\boldsymbol{w}\|_1 \leq \kappa, \\ & w_p \geq 0 \ \forall p, \end{aligned} \tag{2.3.1}$$

with respect to the parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$, state sequence $(s_1, s_2, \ldots, s_T)$ and feature weights $w_1, w_2, \ldots, w_p$. The term $|C_k|\left(\boldsymbol{\mu}_k-\bar{\boldsymbol{\mu}}\right)^2$ in (2.3.1) is a vector of size p whose entries are the contributions of each feature to the BCSS in the k^{th} cluster.

If we consider the clusters $C_1, C_2, \ldots, C_K$ fixed, then the feature weights will be assigned to features based on their individual BCSS contributions: features with larger BCSS contributions will be given larger weights which, in turn, optimises the overall spread between the clusters.

$\kappa \in [1, \sqrt{p}]$ in (2.3.1) is a hyperparameter that controls the degree of sparsity in the feature weights. The squared $\mathcal{L}_2$ penalty in (2.3.1) serves an important role since without it, at most one element of $\boldsymbol{w}$ would be non-zero in general when features are correlated (see [31] for more details). If $w_1 = w_2 = \cdots = w_p$, then 2.3.1 reduces to the maximisation of the BCSS (minimisation of the WCSS) which is the objective of the K-means clustering algorithm.

Combining 2.3.1 with a jump penalty, we give the below definition for the Sparse Jump Model:

Definition 2.3.1 (Sparse Jump Model)**.** The *Sparse Jump Model* with K states is defined by the optimisation programme

$$\begin{aligned} \max \quad & \boldsymbol{w}^{'}\left(\sum_{k=1}^{K}|C_k|\left(\boldsymbol{\mu}_k-\bar{\boldsymbol{\mu}}\right)^2\right) - \lambda\sum_{t=1}^{T}\mathbf{1}_{\{s_t \neq s_{t+1}\}} \\ \text{such that} \quad & \|\boldsymbol{w}\|^2 \leq 1, \quad \|\boldsymbol{w}\|_1 \leq \kappa, \\ & w_p \geq 0 \ \forall p, \end{aligned} \tag{2.3.2}$$

,

with respect to the parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$, state sequence $(s_1, s_2, \ldots, s_T)$ and feature weights $w_1, w_2, \ldots, w_p$. $\kappa \in [1, \sqrt{p}]$ in (2.3.1) is a hyperparameter that controls the degree of sparsity in the feature weights.

If $w_1 = w_2 = \ldots, w_p$, then Definition 2.3.1 reduces to the Standard Jump Model in Definition 2.2.1.

2.4 Continuous Jump Model

In the Standard and Sparse Jump Models, the hidden state variable is discrete-valued since $s_t \in \{1, 2, \ldots, K\}$. [4] extends the Standard Jump Model by generalising the discrete state variable to be a probability vector over all regimes; these authors call this extension the *Continuous Jump Model*.

In mathematical terms, they extend the state space from the discrete set $\{1, 2, \ldots, K\}$ to the probability simplex $\Delta_K := \left\{\boldsymbol{s} = (s_1, s_2, \ldots, s_K)^{'} \in \mathbb{R}^K : \sum_{k=1}^{K} s_k = 1, s_k \geq 0\right\}$.

Each state vector s_t is then a probability vector over all the states and the hidden vector sequence is now represented by $\boldsymbol{S} := [\boldsymbol{s}_1, \boldsymbol{s}_2, \ldots, \boldsymbol{s}_T]' \in \mathbb{R}^{T \times K}$, which should satisfy $\boldsymbol{S} \geq 0$ and $\boldsymbol{S}\mathbf{1}_K = \mathbf{1}_T$, where $\mathbf{1}_n$ is a vector comprising n ones.

Definition 2.4.1 (Continuous Jump Model)**.** The *Continuous Jump Model* with K states is defined by the minimisation problem

$$\arg\min_{\underline{\boldsymbol{\mu}}, \boldsymbol{S}} \sum_{t=1}^{T} L\left(\boldsymbol{y}_t, \underline{\boldsymbol{\mu}}, \boldsymbol{s}_t\right) + \frac{\lambda}{4} \sum_{t=0}^{T-1} \|\boldsymbol{s}_t - \boldsymbol{s}_{t+1}\|_1^2, \tag{2.4.1}$$

where $\underline{\boldsymbol{\mu}}$ are the model parameters, S is the hidden state vector sequence, and the loss function L is defined as a weighted average of the Euclidean distance between y_t and each cluster centre. The loss function is written as

$$L\left(\boldsymbol{y}_t, \underline{\boldsymbol{\mu}}, \boldsymbol{s}_t\right) = \sum_{k=1}^{K} s_{t,k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2,$$

where $s_{t,k}$ is the k^{th} element of the vector $\boldsymbol{s}_t$.

[3] proposed a version of the Continuous Jump Model that incorporates feature selection, analogous to the relation between the Standard and Sparse Jump Models introduced in Sections 2.2 and 2.3 respectively. Since this work is dedicated primarily to Jump Models that assume the hidden state variable to be discrete-valued, we leave readers to consult [3] and [4] for details on the theory and calibration of the Continuous Jump Model.

2.5 Regularised Jump Models

In this section, a new approach to feature selection in the Jump Model framework is introduced. The approach is an adaptation of the Regularised K-means algorithm proposed in [25] to the Jump Model framework. Therefore, we call the models constituting this approach *Regularised Jump Models*.

2.5.1 Regularised K-Means

Definition 2.5.1. The *Regularised K-means* algorithm is defined by the minimisation of

$$\sum_{k=1}^{K} \left\{ \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2 \right\} + \gamma \mathcal{P}\left(\underline{\boldsymbol{\mu}}\right), \tag{2.5.1}$$

with respect to the clusters $C_1, C_2, \ldots, C_K$ and matrix of cluster centres $\underline{\boldsymbol{\mu}} \in \mathbb{R}^{K \times p}$. $\gamma \geq 0$ is a tuning parameter that controls the amount of regularisation applied to the cluster centres. $\mathcal{P} : \mathbb{R}^{K \times p} \to \mathbb{R}$ is a penalty function that depends on $\underline{\boldsymbol{\mu}}$. The first term is the standard K-means clustering objective function defined in (2.1.1).

From [25], below are several penalty function options which are named after their counterparts from regularised regression:

$$\mathcal{L}_0: \quad \mathcal{P}_0\left(\underline{\boldsymbol{\mu}}\right) = \sum_{j=1}^{p} \mathbf{1}_{\{\|\boldsymbol{\mu}_{\cdot,j}\|>0\}} \tag{2.5.2a}$$

$$\text{Lasso}: \quad \mathcal{P}_1\left(\underline{\boldsymbol{\mu}}\right) = \sum_{j=1}^{p} \|\boldsymbol{\mu}_{\cdot,j}\|_1 \tag{2.5.2b}$$

$$\text{Ridge}: \quad \mathcal{P}_2\left(\underline{\boldsymbol{\mu}}\right) = \sum_{j=1}^{p} \|\boldsymbol{\mu}_{\cdot,j}\|^2 \tag{2.5.2c}$$

$$\text{Group-Lasso}: \quad \mathcal{P}_3\left(\underline{\boldsymbol{\mu}}\right) = \sum_{j=1}^{p} \|\boldsymbol{\mu}_{\cdot,j}\| \tag{2.5.2d}$$

where $\boldsymbol{\mu}_{\cdot,j}$ is the j^{th} column of $\underline{\boldsymbol{\mu}}$. The penalty on $\underline{\boldsymbol{\mu}}$ balances the size of the cluster centres and their contribution to the objective function in (2.5.1).

The intuition for penalising the size of the cluster centres lies in the fact that when a variable does not contribute to the partitioning of the data, its estimated cluster centres will be close to the overall mean of the data (which is equal to the zero vector in the case of standardised data).

A proposition in [25] is given here to establish this fact. Let $(\Sigma, \mathcal{F}, \mathbb{Q})$ be a probability space where $\mathbb{Q}$ is the cumulative distribution function of a random vector $Y \in \mathbb{R}^m$. In this setting, $m < p$ and each cluster corresponds to a region in $\mathbb{R}^m$; let these regions be denoted $R_1, R_2, \ldots, R_K$. Consider the optimal value of the standard K-means clustering objective function:

$$V = \int \min_{k\in\{1,2,\ldots,K\}} \|\boldsymbol{y} - \boldsymbol{\mu}_k\|^2 \mathbb{Q}\left(\boldsymbol{dy}\right).$$

Now suppose a variable Z is added to obtain a $(m+1)$-dimensional random vector $Y^* := (Y, Z)$ with corresponding measure $\mathbb{Q}^*$. We have the following proposition:

Proposition 2.5.2. *Let V and V^* be the optimal values of the K-means objective function on $\mathbb{Q}$ and $\mathbb{Q}^*$ respectively. Assume that the added variable Z is uninformative and independent from the original vector Y. More specifically, assume that:*

1. *$\mathbb{R}_k^* := \mathbb{R}_k \times \mathbb{R}$ for all $k = 1, 2, \ldots, K$. This effectively assumes that the value of the uninformative variable Z doesn't affect the cluster assignment.*
2. *$\mathbb{Q}\left(\boldsymbol{dy}|z\right) = \mathbb{Q}\left(\boldsymbol{dy}\right)$ (Y and Z are independent).*

Then the following holds:

R.1 $V^* = 1 + V$

R.2 The optimal centre of the added variable Z is 0 for every cluster $k = 1, 2, \ldots, K$.

Proof. Proof can be found in Section A.1 of the Appendix in [25]. □

Remark 2.5.3. Result R.1 shows that the addition of an uninformative variable increases the value of the objective function, thereby worsening the model fit. This, in turn, offers a basis for feature selection. Result R.2 justifies the penalisation of the size of the cluster

centres as a feature selection approach.

[25] notes that the above argument in Proposition 2.5.2 is simplified as it assumes that the cluster assignments do not change when adding the uninformative variable Z. The asymptotic assignments might change if the added variable dominates the clustering structure but this is rather unlikely under the assumption that at least a few informative variables are already present and that the variables are standardised.

2.5.2 Regularised Jump Model Equation

Combining (2.5.1) with a jump penalty, we propose the following definition for the Regularised Jump Model.

Definition 2.5.4 (Regularised Jump Model)**.** The *Regularised Jump Model* with K states is defined by the minimisation of the objective function

$$\sum_{t=1}^{T-1} \left\{ \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}\|^2 + \lambda \mathbf{1}_{\{s_t \neq s_{t+1}\}} \right\} + \|\boldsymbol{y}_T - \boldsymbol{\mu}_{s_T}\|^2 + \gamma \mathcal{P}\left(\underline{\boldsymbol{\mu}}\right), \tag{2.5.3}$$

with respect to $\underline{\boldsymbol{\mu}}$ and the state sequence $(s_1, s_2, \ldots, s_T)$. $\lambda, \gamma \geq 0$ are hyperparameters and the penalty function $\mathcal{P} : \mathbb{R}^{K \times p} \to \mathbb{R}$ can be chosen from those listed in (2.5.2).

$\gamma = 0$ reduces Definition 2.5.4 to the Standard Jump Model in Definition 2.2.1. $\lambda = 0$ reduces Definition 2.5.4 to the Regularised K-Means model in Definition 2.5.1.

2.5.3 Comparison with Sparse Jump Models

Both the Regularised and Sparse Jump Models employ feature selection. However, their feature selection approaches are different. In the calibration of the Sparse Jump Model (shown in Algorithm 6), a feature weight vector $\boldsymbol{w} \in \mathbb{R}^p$ is introduced and the observation sequence $Y = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)' \in \mathbb{R}^{T \times p}$ is multiplied row-wise by $\boldsymbol{w}$.

The row-wise multiplication transforms the data to a new coordinate system, a transformation similar to that in Principal Component Analysis (PCA). In the PCA method, data is transformed such that the greatest variance by some scalar projection of the data comes to lie on the first coordinate, the second greatest variance on the second coordinate, and so forth. In both PCA and the Sparse Jump Model, the features lose their original interpretation.

In the calibration of the Sparse Jump Model, feature weights are assigned based on their contribution to the BCSS. For an appropriate choice of the tuning parameter κ in Definition 2.3.1, features with lower BCSS contributions are given zero weight, thus *indirectly* inducing sparsity in the features.

In the Regularised Jump Model, feature selection is performed by direct regularisation on the cluster centres. The calibration algorithm for the Regularised Jump Models (Algorithm 8) alternates between learning the most informative set of features in a subspace of $\mathbb{R}^p$ based on their corresponding cluster centres' reduction of the objective function in (2.5.3), and then projecting the cluster centres onto this subspace at each iteration.

The Regularised Jump Model selects informative variables in the original coordinate system, thus allowing them to retain their original interpretations. This is the main

appeal of the Regularised Jump Model over the Sparse Jump Model: the Regularised Jump Model performs *direct* and *interpretable* feature selection, in contrast to the Sparse Jump Model which promotes feature selection by proxy and may not always yield sparse solutions.

Additionally, whilst not explored in this work, the asymptotic properties proved in [25] for the Regularised K-means model could be leveraged to prove similar ones for the Regularised Jump Model. One of these properties is consistency and strong consistency in terms of the Hausdorff distance (see [25] Theorem 6 and the proof in Appendix A. A.3). No similar properties have been proved for the Sparse K-Means model introduced in [29], the clustering analogue to the Sparse Jump Model.

Chapter 3

Calibration of Jump Models

The chapter details the algorithms that perform calibration of the Jump Models outlined in the previous chapter. The models are then tested in a simulation experiment.

3.1 Standard Jump Model Calibration

Denote $\underline{\boldsymbol{\mu}} := (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K)^{'} \in \mathbb{R}^{K \times p}$ the matrix of model parameters and the state sequence $S := (s_1, s_2, \ldots, s_T)$.

Using Algorithm 1 as a template, the calibration algorithm for the Standard Jump Model was proposed in [22] and is shown in Algorithm 5. The model is calibrated using a coordinate descent algorithm that alternates between finding the model parameters $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ that minimise the objective function (2.2.1) with a fixed state sequence (Step 1.1) and finding the state sequence $(s_1, s_2, \ldots, s_T)$ that minimise the objective function (2.2.1) with fixed $\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K$ (Step 1.2).

Following the algorithm proposed in [22], the process is repeated ten times at the most or until the state sequence does not change after one iteration. However, there is no guarantee that the solution reached is the global solution since the solution depends on the initial state sequence.

Therefore, we adopt the initialisation method in [22]: the coordinate descent algorithm is run from ten different state sequences in parallel and the model that achieves the lowest objective function value is chosen. These initial state sequences are generated by the K-means++ seeding technique introduced in [2] which has been shown to improve both the speed and accuracy of standard K-means clustering.

We note that the objective function in Step 5.1 of Algorithm 5 is convex in $\underline{\boldsymbol{\mu}}$ and can be solved in closed-form. If state k appears in $S^{(j-1)}$ ($(j-1)^{\text{th}}$ iteration of the state sequence) at least once, $\boldsymbol{\mu}_k \in \mathbb{R}^p$ has the below optimal solution at the j^{th} iteration:

$$\boldsymbol{\mu}_k^{(j)} = \frac{\sum_{t=1}^{T} \boldsymbol{y}_t \mathbf{1}_{\{s_t^{(j-1)}=k\}}}{\sum_{t=1}^{T} \mathbf{1}_{\{s_t^{(j-1)}=k\}}}, \; k = 1, 2, \ldots, K.$$

$S^{(j)}$ in Step 5.2 is obtained using the dynamic programming template in (1.5.1). Define

$$V(T,s) = \|\boldsymbol{y}_T - \boldsymbol{\mu}_s\|^2, \tag{3.1.1a}$$

$$V(t,s) = \|\boldsymbol{y}_t - \boldsymbol{\mu}_s\|^2 + \min_j \left\{ V(t+1,j) \ + \lambda \mathbf{1}_{\{s \neq j\}} \right\}, \tag{3.1.1b}$$

for $t = T-1, \ldots, 2, 1$. The most likely state sequence is then given by

$$s_1 = \arg\min_j V(1,j), \tag{3.1.2a}$$

$$s_t = \arg\min_j \left\{ V(t,j) + \lambda \mathbf{1}_{\{s_{t-1} \neq j\}} \right\}, \ t = 2, \ldots, T. \tag{3.1.2b}$$

Algorithm 5 Calibration of Standard Jump Model in [22]

Input: Standardised training dataset $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, assumed number of states K, and jump penalty λ.

Step 1: Initialise state sequence $S^{(0)} := \left(s_1^{(0)}, s_2^{(0)}, \ldots, s_T^{(0)}\right)$.

Step 2: Iterate for $j = 1, 2, \ldots, 10$:

Fit model parameters $\boldsymbol{\mu}^{(j)}$:

$$\underline{\boldsymbol{\mu}}^{(j)} \leftarrow \arg\min_{\underline{\boldsymbol{\mu}}} \sum_{t=1}^{T} \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t^{(j-1)}}\|^2. \tag{5.1}$$

Fit state sequence $S^{(j)}$:

$$S^{(j)} \leftarrow \arg\min_S \left\{ \sum_{t=1}^{T-1} \left\{ \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}^{(j)}\|^2 + \lambda \mathbf{1}_{\{s_t \neq s_{t+1}\}} \right\} + \|\boldsymbol{y}_T - \boldsymbol{\mu}_{s_T}^{(j)}\|^2 \right\}. \tag{5.2}$$

Break if $S^{(j-1)} = S^{(j)}$.

Output: Estimated model parameters $\underline{\boldsymbol{\mu}}^*$ and state sequence S^*.

3.2 Sparse Jump Model Calibration

The calibration of the Sparse Jump Model can be performed using an extension of the coordinate descent algorithm in Algorithm 5. Holding the feature weight vector $\boldsymbol{w}$ fixed, (2.3.2) is optimised in terms of $\underline{\boldsymbol{\mu}}$ and S. Secondly, holding $\underline{\boldsymbol{\mu}}$ and S fixed, (2.3.2) is optimised in terms of $\boldsymbol{w}$.

[20] comments that this iterative approach is not guaranteed to generate a global optimum because the problem is non-convex. Additionally, the first optimisation involves applying the fitting algorithm of the Standard Jump Model to a weighted version of the data, which by itself is not guaranteed to find a global optimum.

In Step 2(d), solving (2.3.2) with respect to $\boldsymbol{w}$, while keeping $\underline{\boldsymbol{\mu}}$ and S fixed, can be done using *soft-thresholding*, a technique used in signal processing to compress signals.

A soft-thresholding operator takes as inputs an array of values and a threshold value. The operator applies shrinkage whereby values below the threshold are set to zero, and values above the threshold are reduced by the threshold value.

The operation effectively denoises the input array by shrinking or eliminating small components while preserving larger components. The soft-threholding operator S is given by $S(\boldsymbol{x}, c) = \text{sgn}(\boldsymbol{x}) \odot (|\boldsymbol{x}| - c)_+$, where $\boldsymbol{x}_+$ denotes the positive part of the elements in $\boldsymbol{x}$ and $\odot$ denotes element-wise multiplication.

The solution to the convex problem (2.3.2), which follows from the Karush-Kuhn-Tucker conditions (more details can be found in [7]), is $\boldsymbol{w} = \frac{S(\boldsymbol{x},\Delta)}{\|S(\boldsymbol{x},\Delta)\|}$ where $\boldsymbol{x} = \sum_{k=1}^{K} |C_k| (\boldsymbol{\mu}_k - \bar{\boldsymbol{\mu}})^2$ is a vector comprising the between-cluster sum of squares (BCSS) contributions of each feature.

Here, $\Delta = 0$ if that results in $\|\boldsymbol{w}\|_1 \leq \kappa$; otherwise, we choose $\Delta > 0$ to yield $\|\boldsymbol{w}\|_1 = \kappa$. This assumes that there is a unique maximal element of $\boldsymbol{x}$ and that $1 \leq \kappa \leq \sqrt{p}$ [29].

3.3 Regularised Jump Model Calibration

We propose calibrating the Regularised Jump Models using a coordinate descent algorithm similar to Algorithm 5 for Standard Jump Models. The coordinate descent algorithm alternates between minimising (2.5.3) with respect to the state sequence $\underline{\boldsymbol{\mu}}$ while keeping S fixed, and minimising (2.5.3) with respect to S while keeping $\underline{\boldsymbol{\mu}}$ fixed. These two steps are repeated for ten iterations at the most or until S does not change after two consecutive iterations.

Firstly, seven initial state sequences are generated using the strategy proposed in [25] which the authors show to be the most optimal for the Regularised K-Means model after benchmarking it against popular initialisation strategies. The initialisation strategy incorporates potential sparsity in the initial cluster centres in light of the feature selection aspect of the Regularised Jump Model. The initialisation strategy is given in Algorithm 7.

Each initial state sequence outputted from Algorithm 7 is used as an input in Algorithm 8. The algorithm is run in parallel using these seven initial cluster assignments and the run which ultimately yield the lowest objective function value, is chosen as the final result.

Once a initial state sequence has been generated, the model parameters $\underline{\boldsymbol{\mu}}$ are updated (Step 8.1). In particular, assuming a fixed state sequence $S = (s_1, s_2, \ldots, s_T)$, the following problem is solved:

$$\arg\min_{\underline{\boldsymbol{\mu}}} \sum_{t=1}^{T} ||\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}||^2 + \gamma \mathcal{P}(\underline{\boldsymbol{\mu}}) . \tag{3.3.1}$$

Algorithm 6 Calibration of Sparse Jump Model in [20]

Input: Standardised training dataset $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, assumed number of states K, and hyperparameters λ, κ.

Step 1: Initialise feature weights $\boldsymbol{w}$ as $\boldsymbol{w}^{(0)} = \left(\frac{1}{\sqrt{p}}, \frac{1}{\sqrt{p}}, \ldots, \frac{1}{\sqrt{p}}\right)$.

Step 2: Iterate for $i = 1, 2, \ldots,$ until $\|\boldsymbol{w}^{(i)} - \boldsymbol{w}^{(i-1)}\|_1 / \|\boldsymbol{w}^{(i-1)}\|_1 < 10^{-4}$:

(a) Compute sequence of weighted features

$$\boldsymbol{z}_t = \boldsymbol{y}_t \odot \sqrt{\boldsymbol{w}^{(i-1)}},\ t = 1, 2, \ldots, T,$$

where $\sqrt{\boldsymbol{w}^{(i)}}$ is the element-wise square root of $\boldsymbol{w}^{(i)}$.

(b) Initialise state sequence $S^{(0)} := \left(s_1^{(0)}, s_2^{(0)}, \ldots, s_T^{(0)}\right)$.

(c) Iterate for $j = 1, 2, \ldots, 10$:

i. Fit model parameters:

$$\underline{\boldsymbol{\mu}}^{(j)} \leftarrow \arg\min_{\underline{\boldsymbol{\mu}}} \sum_{t=1}^{T} \|\boldsymbol{z}_t - \boldsymbol{\mu}_{s_t^{(j-1)}}\|^2.$$

ii. Fit state sequence:

$$S^{(j)} \leftarrow \arg\min_{S} \left\{ \sum_{t=1}^{T} \left\{ \|\boldsymbol{z}_t - \boldsymbol{\mu}_{s_t}^{(j)}\|^2 + \lambda \mathbf{1}_{\{s_t \neq s_{t+1}\}} \right\} + \|\boldsymbol{z}_T - \boldsymbol{\mu}_{s_T}^{(j)}\|^2 \right\}.$$

Break if $S^{(j-1)} = S^{(j)}$.

(d) Update weights $\boldsymbol{w}^{(i)}$, while holding the model parameters $\underline{\boldsymbol{\mu}}^{(j)}$ and $S^{(j)}$ fixed, by solving (2.3.2) using soft thresholding.

Output: Estimated model parameters $\underline{\boldsymbol{\mu}}^*$, state sequence S^* and feature weights $\boldsymbol{w}^*$.

Algorithm 7 Initialisation of state sequences for calibration of Regularised Jump Model in [25]

Step 1 Cluster $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$ using standard K-means (shown in Algorithm 4), obtaining a matrix of initial cluster centres $\underline{\boldsymbol{\mu}} = (\boldsymbol{\mu}_1, \boldsymbol{\mu}_2, \ldots, \boldsymbol{\mu}_K)'$.

Step 2 Compute the Euclidean norm for each cluster centre $d_j = \|\boldsymbol{\mu}_{\cdot,j}\|$ for $j = 1, 2, \ldots, p$ and order them in descending order.

Step 3 Execute K-means on the subset of variables corresponding to the 1, 2, 5, 10, 25, 50 and 100% largest d_j.

Output: Seven initial state sequences $\left(s_1^1, s_2^1, \ldots, s_T^1\right), \left(s_1^2, s_2^2, \ldots, s_T^2\right), \ldots, \left(s_1^7, s_2^7, \ldots, s_T^7\right)$.

The solutions to (3.3.1) for each penalty function in (2.5.2), is given in A.1.3. Once $\underline{\boldsymbol{\mu}}$ is updated, the state sequence S is updated.

Since Steps 8.2 and 5.2 are identical, the state sequence is updated using the same dynamic programming equations in (3.1.1) and (3.1.2).

Algorithm 8 Coordinate descent algorithm for calibration of Regularised Jump Model

Input: Standardised training dataset $\boldsymbol{Y} = (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$, assumed number of states K, penalty function $\mathcal{P}$, hyperparameters λ, γ, and initial state sequence $S^{(0)} := \left(s_1^{(0)}, s_2^{(0)}, \ldots, s_T^{(0)}\right)$ generated using Algorithm 7.

Step 1: Iterate for $k = 1, 2, \ldots, 10$:

$$\underline{\boldsymbol{\mu}}^{(k)} \leftarrow \arg\min_{\underline{\boldsymbol{\mu}}} \left\{ \sum_{t=1}^{T} ||\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t^{(k-1)}}||^2 + \gamma \mathcal{P}\left(\underline{\boldsymbol{\mu}}\right) \right\}, \tag{8.1}$$

$$S^{(k)} \leftarrow \arg\min_{S} \left\{ \sum_{t=1}^{T-1} \left\{ \|\boldsymbol{y}_t - \boldsymbol{\mu}_{s_t}^{(k)}\|^2 + \lambda \mathbf{1}_{\{s_t \neq s_{t+1}\}} \right\} + \|\boldsymbol{y}_T - \boldsymbol{\mu}_{s_T}^{(k)}\|^2 \right\}. \tag{8.2}$$

Break if $S^{(k)} = S^{(k-1)}$.

Output: Estimated matrix of cluster centres (or model parameters) $\underline{\boldsymbol{\mu}}^*$ and state sequence S^*.

3.4 Hyperparameter Tuning

Table 3.1 shows the hyperparameters of each Jump Model.

Model	λ	κ	γ
Standard Jump	✓		
Sparse Jump	✓	✓	
Regularised Jump	✓		✓

Table 3.1: Jump Model hyperparameters.

The literature on the hyperparameter tuning of Jump Models is limited. [11] suggests tuning based on a grid search of potential hyperparmeter values and picking which combination produces the lowest Fang-Tang Information Criterion (FTIC) value, a criterion introduced in [12].

In [11], the FTIC equation is an approximation and applies only to the Standard and Sparse Jump Models. Approximating information criteria such as the FTIC would require knowledge of the likelihood function of the various Jump Models which, for the Regularised Jump Models, is unavailable.

Instead, we opt for a non-parametric criterion that is applicable to all Jump Models. We propose hyperparameter tuning based on a criterion pertaining to *clustering stability*. The key idea of clustering stability is that if we repeatedly draw samples from the

same distribution as that of the original dataset, and apply the Jump Model calibration algorithms, a good algorithm should produce state sequences that are similar from one sample to another.

Denote $Z = \{X_1, X_2, \ldots, X_T\}$ a random sample of size T from some unknown distribution function $F : \mathbb{R}^p \to \mathbb{R}$. Let $\boldsymbol{\theta}$ denotes the vector of active hyperparameters for a given Jump Model which can be referenced in Table 3.1. A clustering assignment ψ is defined as a mapping $\psi : \mathbb{R}^p \to \{1, 2, \ldots, K\}$ and a given Jump Model generates T clustering assignments $\Psi(., \boldsymbol{\theta})$ when applied to a sample Z.

Definition 3.4.1 (Clustering Distance)**.** The *clustering distance* between any two clustering assignments ψ_1 and ψ_2 is defined as

$$d(\psi_1, \psi_2) = \mathbb{E}_{X,Y \sim F}\{\mathbf{1}_{\{\psi_1(X)=\psi_1(Y)\}} - \mathbf{1}_{\{\psi_2(X)=\psi_2(Y)\}}\},$$

where $X, Y \in \mathbb{R}^p$ are realisations sampled from F.

The distance between ψ_1 and ψ_2 measures the probability of their disagreement. The clustering instability of a given Jump Model is given in the following definition:

Definition 3.4.2 (Clustering Instability)**.** The *clustering instability* of a clustering algorithm Ψ is given by

$$S(\Psi; \boldsymbol{\theta}, T) = \mathbb{E}[d(\Psi(Z_1; \boldsymbol{\theta}), \Psi(Z_2; \boldsymbol{\theta}))], \tag{3.4.1}$$

where $\Psi(Z_1; \boldsymbol{\theta})$ and $\Psi(Z_2; \boldsymbol{\theta})$ are two clustering assignments obtained by applying $\Psi(.; \boldsymbol{\theta})$ to Z_1 and Z_2 which are two independent samples from F of size T.

Hyperparameter tuning is thus performed by finding a set of hyperparameter values that minimise (3.4.1). Various methods of estimating (3.4.1) for hyperparameter tuning in clustering problems have been proposed ([26], [16] and [6]).

We opt for a simple method of estimating (3.4.1) that is similar to the one proposed in [16]; ten bootstrapped samples of the dataset Y are generated, each of which are size T. For a fixed combination of hyperparameter values and a given Jump Model, ten clustering assignments (or state sequences) are estimated once the model is fitted onto each bootstrapped sample. The number of times any two state sequences are not equal, are then counted and averaged across all $\binom{10}{2} = 45$ comparisons. This estimate can be expressed mathematically as

$$\hat{S}(\Psi; \boldsymbol{\theta}, T) = \binom{10}{2}^{-1} \sum_{i=1}^{10} \sum_{j=i+1}^{10} \Big(\sum_{t=1}^{T} \mathbf{1}_{\{\hat{s}_t^i \neq \hat{s}_t^j\}} \Big), \tag{3.4.2}$$

where $\hat{s}_t^i$ is the estimated state at time t for the i^{th} boostrapped sample.

3.5 Simulation Study

We conduct a simulation study to compare the accuracy of the Standard, Sparse and Regularised Jump Models, with respect to both state estimation and feature selection. In the simulation study, the true state sequence and array of relevant features are both known, which makes it possible to evaluate the ability of each model to correctly identify the state sequence and array of relevant features.

We adopt the same data generating process as in [20]. In their simulation study, the authors test the Standard and Sparse Jump Models against several popular regime-switching models such as the Hidden Markov Model (HMM) and its various extensions, and determine that these two Jump Models deliver superior performance.

Instead of duplicating the simulation study by testing the same regime-switching models, we test the proposed Regularised Jump Models and compare their performances with the Standard and Sparse Jump Models.

3.5.1 Setup

Data is simulated from a three-state multivariate Gaussian Hidden Markov Model (HMM) (an introduction to the HMM can be found in [24]).

$$\boldsymbol{y}_t | s_t \sim \mathcal{N}\left(\boldsymbol{\mu}_{s_t}, \boldsymbol{I}_P\right),$$

where $\boldsymbol{I}_P$ is an identity matrix of order P and s_t is a first-order Markov chain, with parameters

$$(\boldsymbol{\mu}_1)_p = \mu \mathbf{1}_{\{p \leq 15\}}, \quad \boldsymbol{\mu}_2 = \mathbf{0}_P, \quad (\boldsymbol{\mu}_3)_p = -\mu \mathbf{1}_{\{p \leq 15\}},$$

$$\Pi = \begin{pmatrix} 0.9903 & 0.0047 & 0.0050 \\ 0.0157 & 0.9666 & 0.0177 \\ 0.0284 & 0.0300 & 0.9416 \end{pmatrix},$$

where $(\boldsymbol{\mu})_p$ is the p^{th} element of the vector $\boldsymbol{\mu}$ and $\mathbf{0}_P$ is a vector containing P zeros. Π is the transition probability matrix of the state sequence and the values of Π come from an empirical application of the HMM onto weekly stock returns in [22]. Since the covariance matrix in all states is the identity matrix, all information separating the states is contained in the conditional mean vector $\boldsymbol{\mu}_{s_t}$.

The data is simulated by first generating a sequence of states according to Π, starting from its stationary distribution (we set $\mathbb{P}(s_0 = k) = \frac{1}{3}$ for $k = 1, 2, 3$). Secondly, at time t, observations are sampled from multivariate Gaussian distributions with mean vector $\boldsymbol{\mu}_{s_t}$ and covariance matrix $\boldsymbol{I}_P$.

In state one, the first fifteen features have mean value μ and in state three, they have mean value $-\mu$. All other features have mean zero in all states. Hence, features beyond the first fifteen are white noise and are irrelevant. Under the second state, the mean value of the relevant and irrelevant features are equal, thereby raising the importance of feature selection in improving the state estimation accuracy of the models.

3.5.2 Model Tuning and Fitting

Once the data is simulated, the hyperparameters for each Jump Model are tuned based on the minimisation of the clustering instability estimate in (3.4.2). The time-series bootstrapping is performed using the Stationary Bootstrap method introduced in [23]. The hyperparameter tuning is performed using the Optuna library in Python (see [1] for more details). In our implementation of the Standard and Sparse Jump Models, we used and adapted the supplementary Python code provided in [20].

The jump and regularisation penalties, λ and γ respectively, were chosen over seven logarithmically spaced values between 10^{-1} and 10^2. κ for the Sparse Jump Model was chosen over seven equally spaced values between 1 and $\sqrt{P}$. Once the hyperparameters are tuned, the Jump Models were fit using the algorithms outlined in the chapter.

3.5.3 Evaluation of Model Accuracy

To evaluate model accuracy of each Jump Model with respect to state estimation, we use the Balanced Accuracy (BAC) metric introduced in [8]. BAC can be written as

$$\text{BAC} = \frac{1}{K}\sum_{k=1}^{K}\frac{tp_k}{tp_k + fn_k}, \tag{3.5.1}$$

which is the average of accuracy per observed state to avoid inflated performance estimates on imbalanced datasets ([20]). tp_k is the number of true positives and fn_k is the number of false negatives in state k.

BAC is also used to measure model accuracy of the models with respect to feature selection. Firstly, because the Standard and Regularised $\mathcal{P}_2$ Jump Models do not induce sparsity in the features (see Proposition A.1.3 for more details on the Regularised $\mathcal{P}_2$ Jump Model), they are excluded. The estimated array of relevant features $(\omega_j) \in \mathbb{R}^P$ has entries

$$\omega_j = \mathbf{1}_{\{\|\hat{\boldsymbol{\mu}}_{\cdot,j}\|>0\}},\ j = 1, 2, \ldots, P, \tag{3.5.2}$$

where $\hat{\boldsymbol{\mu}}_{\cdot,j}$ is the j^{th} column of the estimated matrix of cluster centres $\hat{\underline{\boldsymbol{\mu}}}$. The true array of relevant features is defined as

$$\Big(\underbrace{1, 1, \ldots, 1}_{\text{Relevant features}} \ , \ \underbrace{0, 0, \ldots, 0}_{\text{Irrelevant features}} \Big). \tag{3.5.3}$$

In (3.5.3), the first fifteen entries are ones, indicating that the first fifteen features are relevant. The remaining $P - 15$ entries are zeros, indicating that the last $P - 15$ features are irrelevant. Using the estimated and true sequences defined above, the BAC is then calculated to evaluate the accuracy of the model in terms of feature selection.

3.5.4 Results

Figure 3.1 shows the BAC of the Regularised Jump Models as a function of the regularisation parameter γ, for different numbers of observations of T and different jump penalties λ, using the same simulation setup laid out in Subsection 3.5.1 (setting $\mu = 0.5$ and $P = 60$). The BAC was averaged over 100 simulations.

Table 3.2 compare the state estimation performances of the Standard Jump Model (Standard), Sparse Jump Model (Sparse) and Regularised Jump Model with $\mathcal{P}_k$ Penalty Function (Regularised $\mathcal{P}_k$) for $k \in \{0, 1, 2, 3\}$, for different values of μ and for different numbers of features P. Table 3.3 compare the feature selection performances of the Sparse, Regularised $\mathcal{P}_0, \mathcal{P}_1$ and $\mathcal{P}_3$ Jump Models for different values of μ and for different numbers of features P.

The reported values of Tables 3.2 and 3.3 are the mean (and standard deviation) of the BAC of the estimated state sequence and of (3.5.2) respectively; the means and standard deviations are calculated over 100 simulations of $T = 500$ observations for each

combination of μ and P.

Similar to [20], we use the Wilcoxon signed-rank test to determine whether the differences between the BACs in the Regularised Jump Models and those in the Sparse Jump Model are statistically significant. Bold entries in Tables 3.2 and 3.3 denote BACs of a Regularised Jump Model that are higher than that of the Sparse Jump Model with statistical significance $\alpha = 0.05$.

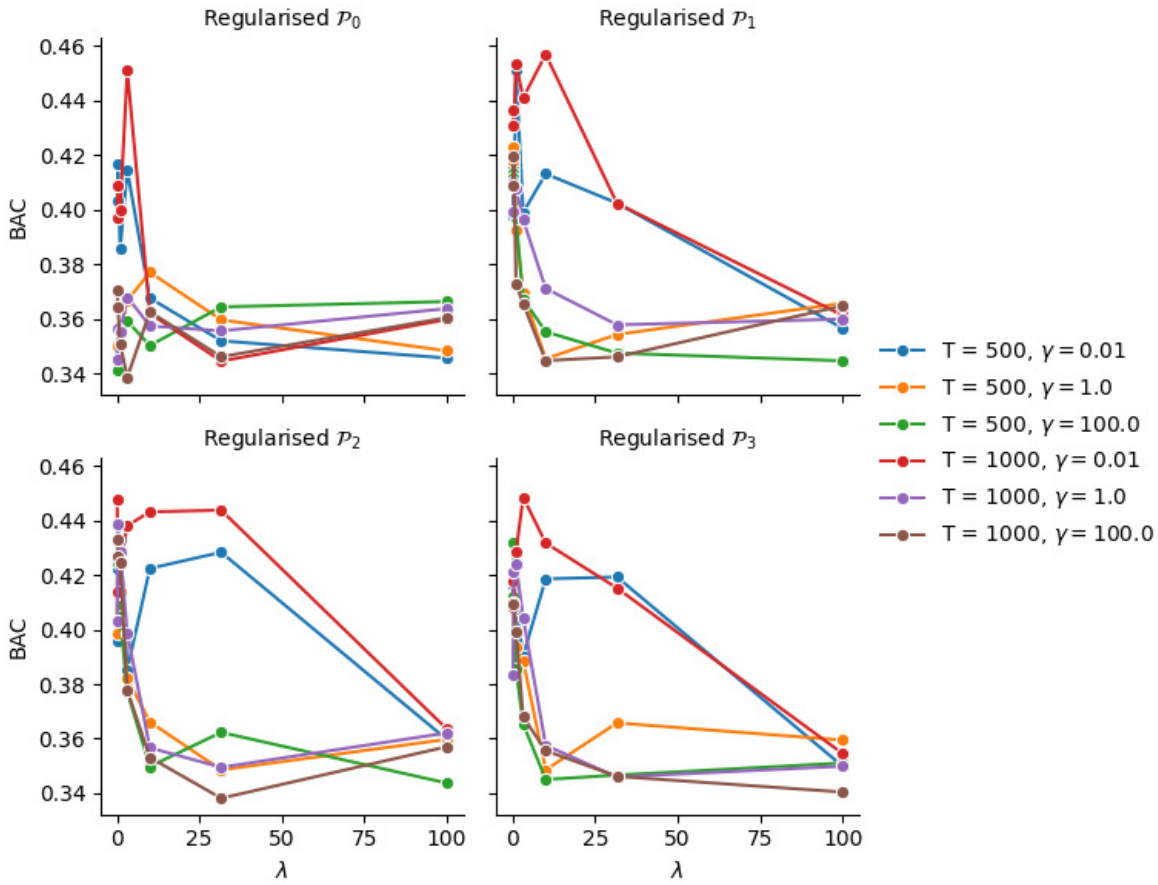

Figure 3.1: Average BAC of Regularised Jump Models as a function of the jump penalty λ, for various combinations of time lengths T and regularisation parameter values γ.

3.5.5 Observations of Results

Differences in BAC based on hyperparameter and time length: Figure 3.1 shows marginal differences in BACs based on hyperparameter values and time length, for the Regularised Jump models. For the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ models, the optimal values for γ and λ are 0.1 and ≈ 10 respectively. The BAC decreases past $\lambda = 25$. In most cases, a higher time length leads to marginally higher BACs.

BAC Dependence on μ, T and P: In addition to the feature means μ and the number of total features P, the BACs in Tables 3.2 and 3.3 are also a function of the number of relevant features and the time length of the series T.

Increasing either of the latter would generally lead to higher BACs. This can be intuited from Table 3.2 in which the BACs generally decrease as the number of total features P increases, which is equivalent to the ratio of irrelevant to relevant features increasing.

Importance of feature selection: Comparing Standard and the remaining models in Table 3.2, it is evident that feature selection improves accuracy of state estimation when the number of features P is increased. In Table 3.2, the BAC of the Standard Jump Model, a model that does not perform feature selection, generally decreases when P is increased. On the other hand, the BAC of the Sparse and Regularised Jump Models stay approximately level compared to the case of there being no irrelevant features (when $P = 15$).

The feature selection is more efficient for higher values of μ since for higher values of μ, the relevance of the first fifteen features becomes more apparent and hence easier for the model to identify. This logic is reinforced by the results in Table 3.3; the feature selection performance of the models improves for higher values of μ.

Correspondence between state estimation and feature selection: When comparing the relative performances of the Sparse, Regularised $\mathcal{P}_0, \mathcal{P}_1$ and $\mathcal{P}_3$ Jump Models in Tables 3.2 and 3.3, we can see a correspondence between classification accuracy in terms of state estimation, and classification accuracy in terms of feature selection: models that are more accurate in terms of feature selection, are also more accurate in terms of state estimation. This further highlights the importance of feature selection in accurately estimating the true state sequence.

If we unbundle the BACs in Tables 3.2 and 3.3 in terms of μ, model and number of features P, and calculate the Spearman rank correlation coefficient between these two unbundled series of BAC values, we compute an estimate of 91% with a p-value of $\mathcal{O}\left(10^{-24}\right)$. This leads us to reject the null hypothesis of there being no monotonic relationship between state estimation accuracy and feature selection accuracy.

Evidence of Regularised Jump Model Outperformance: Tables 3.2 and 3.3 demonstrate outperformance of the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ Models compared to both the Standard and Sparse Jump Models. The Regularised $\mathcal{P}_0$ model performs roughly in line with, or slight worse than, the Sparse Jump Model in terms of state estimation and feature selection.

In Table 3.2, the outperformance of the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ is most evident for the cases $\mu \geq 0.5$ and $P < 300$. For $\mu = 0.25$ and $P = 300$, all models produce roughly the same BACs that range between 0.334 and 0.344.

Similar results can be observed in Table 3.3. All models perform at approximately the same level for $\mu = 0.25$ and for $\mu \geq 0.5$, the Regularised $\mathcal{P}_1$ and $\mathcal{P}_3$ outperform both the Sparse and Regularised $\mathcal{P}_0$ models. The outperformance of the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ over the Sparse Jump Model is statistically significant for some cases, as shown by the bold entries in Tables 3.2 and 3.3.

	P	15	30	60	150	300
μ	**Model**					
0.25	Standard	0.351 (0.06)	0.338 (0.05)	0.332 (0.05)	0.343 (0.04)	0.334 (0.03)
	Sparse	0.349 (0.06)	0.343 (0.05)	0.343 (0.05)	0.337 (0.04)	0.336 (0.03)
	Regularised $\mathcal{P}_0$	0.354 (0.06)	0.345 (0.05)	0.336 (0.05)	0.341 (0.05)	0.334 (0.05)
	Regularised $\mathcal{P}_1$	0.359 (0.08)	0.345 (0.06)	0.341 (0.06)	0.335 (0.04)	0.344 (0.04)
	Regularised $\mathcal{P}_2$	0.343 (0.07)	0.342 (0.06)	0.340 (0.05)	0.335 (0.05)	0.344 (0.04)
	Regularised $\mathcal{P}_3$	0.348 (0.07)	0.343 (0.06)	0.339 (0.05)	0.342 (0.05)	0.334 (0.04)
0.50	Standard	0.362 (0.10)	0.368 (0.08)	0.341 (0.07)	0.341 (0.05)	0.336 (0.04)
	Sparse	0.376 (0.08)	0.356 (0.08)	0.345 (0.06)	0.345 (0.05)	0.344 (0.04)
	Regularised $\mathcal{P}_0$	0.358 (0.09)	0.374 (0.09)	0.365 (0.09)	0.349 (0.07)	0.339 (0.06)
	Regularised $\mathcal{P}_1$	0.362 (0.11)	0.393 (0.13)	0.387 (0.12)	0.350 (0.09)	0.349 (0.05)
	Regularised $\mathcal{P}_2$	0.380 (0.14)	0.382 (0.14)	0.389 (0.13)	0.344 (0.07)	0.360 (0.07)
	Regularised $\mathcal{P}_3$	0.383 (0.14)	0.382 (0.14)	0.374 (0.13)	0.358 (0.08)	0.354 (0.06)
0.75	Standard	0.385 (0.15)	0.401 (0.13)	0.366 (0.10)	0.353 (0.06)	0.355 (0.06)
	Sparse	0.380 (0.12)	0.363 (0.11)	0.364 (0.09)	0.347 (0.07)	0.335 (0.06)
	Regularised $\mathcal{P}_0$	0.418 (0.14)	0.386 (0.14)	0.389 (0.14)	0.387 (0.13)	0.408 (0.14)
	Regularised $\mathcal{P}_1$	0.418 (0.18)	0.406 (0.18)	0.432 (0.18)	0.394 (0.17)	0.458 (0.18)
	Regularised $\mathcal{P}_2$	0.400 (0.18)	0.420 (0.16)	0.447 (0.18)	0.399 (0.18)	0.447 (0.18)
	Regularised $\mathcal{P}_3$	0.407 (0.18)	0.435 (0.19)	0.391 (0.17)	0.403 (0.15)	0.468 (0.17)
1.00	Standard	0.448 (0.20)	0.423 (0.15)	0.391 (0.13)	0.357 (0.09)	0.347 (0.08)
	Sparse	0.408 (0.15)	0.393 (0.14)	0.363 (0.12)	0.374 (0.11)	0.347 (0.08)
	Regularised $\mathcal{P}_0$	0.423 (0.19)	0.409 (0.18)	0.464 (0.19)	0.411 (0.16)	0.494 (0.20)
	Regularised $\mathcal{P}_1$	0.483 (0.22)	0.446 (0.24)	0.524 (0.25)	0.499 (0.24)	**0.564** (0.22)
	Regularised $\mathcal{P}_2$	0.495 (0.23)	0.465 (0.24)	0.523 (0.23)	0.462 (0.23)	0.498 (0.23)
	Regularised $\mathcal{P}_3$	0.444 (0.22)	0.447 (0.24)	0.507 (0.24)	0.492 (0.22)	**0.597** (0.23)

Table 3.2: Average BAC of state sequence (3 d.p.) for each Jump Model, across different values of μ and number of features P. Values in brackets are standard deviations (2 d.p.).

	P	30	60	150	300
μ	**Model**				
0.25	Sparse	0.499 (0.08)	0.502 (0.05)	0.502 (0.03)	0.506 (0.03)
	Regularised $\mathcal{P}_0$	0.507 (0.04)	0.498 (0.02)	0.502 (0.01)	0.501 (0.02)
	Regularised $\mathcal{P}_1$	0.511 (0.05)	0.505 (0.03)	0.496 (0.02)	0.504 (0.03)
	Regularised $\mathcal{P}_3$	0.503 (0.05)	0.506 (0.03)	0.499 (0.03)	0.498 (0.04)
0.5	Sparse	0.509 (0.07)	0.507 (0.05)	0.509 (0.04)	0.505 (0.04)
	Regularised $\mathcal{P}_0$	0.530 (0.05)	0.529 (0.04)	0.513 (0.03)	0.511 (0.03)
	Regularised $\mathcal{P}_1$	0.560 (0.10)	0.573 (0.10)	0.535 (0.08)	0.509 (0.04)
	Regularised $\mathcal{P}_3$	0.582 (0.10)	0.588 (0.11)	0.528 (0.06)	0.513 (0.05)
0.75	Sparse	0.538 (0.07)	0.539 (0.08)	0.531 (0.07)	0.522 (0.06)
	Regularised $\mathcal{P}_0$	0.540 (0.07)	0.538 (0.05)	0.544 (0.07)	0.568 (0.05)
	Regularised $\mathcal{P}_1$	0.594 (0.13)	0.608 (0.14)	0.621 (0.15)	**0.650** (0.13)
	Regularised $\mathcal{P}_3$	0.621 (0.15)	0.603 (0.13)	0.631 (0.16)	**0.665** (0.14)
1.00	Sparse	0.574 (0.11)	0.602 (0.12)	0.587 (0.12)	0.563 (0.10)
	Regularised $\mathcal{P}_0$	0.557 (0.10)	0.549 (0.09)	0.548 (0.08)	0.624 (0.10)
	Regularised $\mathcal{P}_1$	0.622 (0.16)	0.660 (0.18)	**0.695** (0.18)	**0.743** (0.17)
	Regularised $\mathcal{P}_3$	0.603 (0.14)	0.629 (0.15)	0.637 (0.16)	**0.782** (0.17)

Table 3.3: Average BAC of relevant features (3 d.p.) for each Jump Model, across different values of μ and number of features P. Values in brackets are standard deviations (2 d.p.).

Chapter 4

Empirical Study

The chapter presents the empirical calibration of Jump Models onto a dataset comprising market and macroeconomic variables. To the best of our knowledge, this is the first empirical Jump Model calibration whose features include macroeconomic variables. In [20] and [11] for instance, Jump Models are fitted onto features pertaining to the returns of equities and cryptocurrencies respectively. In our study, we calibrate Jump Models onto a dataset containing a broad and diverse collection of features which are believed to influence market regime dynamics.

In the calibration, we assume $K = 2$ number of states or regimes (the terms "regime" and "state" are used interchangeably henceforth). We label the first regime a normal/bull market which is marked by characteristics such as stable economic conditions, normal to strong levels of business profitability and positive investor sentiment. The second regime is labelled a bear market and is marked by characteristics opposite to that of a bull market.

Once the Jump Models were calibrated, the model outputs were used to backtest strategies that trade the Euro iTraxx Main five-year contract, a Index CDS contract. A short introduction to Credit Default Swaps is given in A.2.

4.1 Data Pre-Processing

The twenty-two data series listed in Table A.3 were used in the empirical calibration of the Jump Models. The series cover a broad range of factors relevant to financial markets; these factors have been labelled Commodites, Credit Valuation and Momentum, Currency, Macro Indicators, Risk-Free Rates and Sentiment.

Before calibrating the Jump Models, each series was differenced to make it more stationary. The differencing type was based on the data type of the series: for series with a data type of Price, a percentage difference was applied. For the remaining data series, a simple difference was applied. Each series was differenced using one-week, four-week and twelve-week lags. Therefore, there were $p = 22 \times 3 = 66$ features used in the empirical calibration. We call the resulting dataset with 66 columns the *Features Dataset*.

The data series shown in Table A.3 differ in term of their date of inception. As a result, a start date of 31st December 2002 was chosen to strike a balance between maximising the length and coverage of the Features Dataset. Missing entries were imputed using the Multivariate Imputation by Chained Equations method ([27]).

4.2 Backtest Methodology

4.2.1 CDS Data and Return Estimation

We used Euro iTraxx Main 5Y spreads from the 12th of October 2011 to the 5th of August 2024. The spreads of this Index CDS contract are given in Figure 4.1. We refer to the specifc CDS contract as "the CDS contract" henceforth.

We convert the spreads shown in Figure 4.1 into a daily return approximation to be used for the backtest. Let s_t denote the CDS spread at time t. Assuming 252 business days in a year, the approximated daily return at time t is calculated as

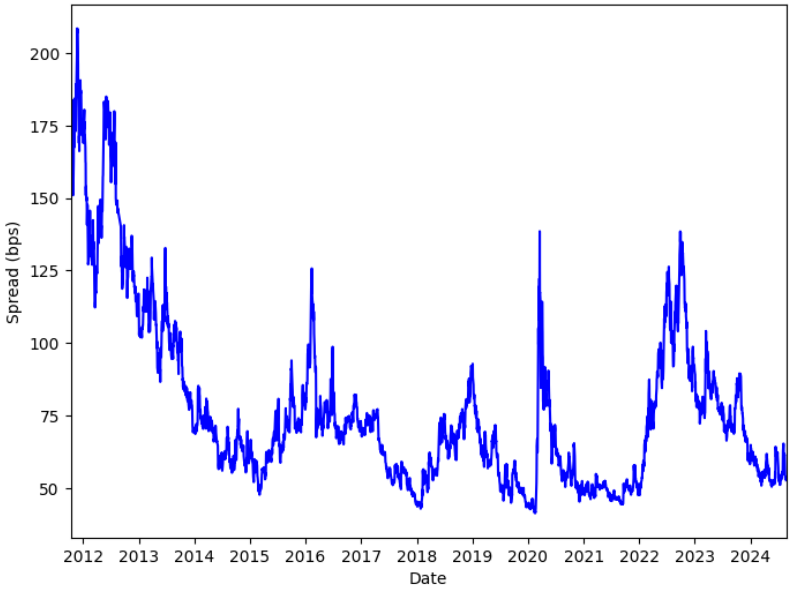

Figure 4.1: EUR iTraxx Main 5Y Spreads between October 2011 and August 2024.

$$\hat{r}_t = \underbrace{-\frac{1}{252} s_{t-1}}_{\text{Carry}} + \underbrace{SD\,(s_t - s_{t-1})}_{\text{Spread Change}}, \tag{4.2.1}$$

where SD denotes the *spread duration* of the CDS contract. Spread duration is the price sensitivity of a credit instrument to changes in credit spreads. Since the term of the studied contract is five years, we assume $SD = 4.5$.

Assuming the normal case of credit spreads being strictly positive, the Carry term in (4.2.1) shows that a CDS protection buyer pays a rate proportional to s_{t-1}. The negative carry incurred is offset when the Spread Change is positive i.e when credit spreads widen. The converse holds for a CDS protection seller: the seller earns a rate proportional to s_{t-1} but suffers losses when credit spreads widen.

4.2.2 Model Training and Online Learning of Market Regimes

The Jump Models tested were tuned and fitted onto the Features Dataset on the last day of every month covered by the dataset, using a rolling ten-year window. Each rolling-window sub-sample of the Features Dataset was standardised using the sub-sample's mean and standard deviation.

Once the Jump Models were tuned (using the methodology laid out in Section 3.4) and fitted, states for the following month were estimated using the online learning framework introduced in Algorithm 3.

The online learning algorithm is given in [21]. Let $\underset{\sim}{\boldsymbol{\mu}} \in \mathbb{R}^{2\times 66}$ be the estimated model parameter values on day t (assuming 30 days in a month, $t = 1, 2, \ldots, 29$). Let $\hat{s_0}$ be the estimated state on the last day of the previous month. Finally, let λ be the tuned hyperparameter value. By time t, we assume knowledge of the standardised data point $\boldsymbol{z}_t \in \mathbb{R}^{66}$ and the prior arrival cost function $\mathcal{A}_{t-1}$.

As hinted in [21], the model parameters $\underset{\sim}{\boldsymbol{\mu}}$ can be updated once the arrival cost function is updated and the state is estimated at time t; this parameter update can affect state estimates in later days of the month. This can be a useful feature since, under our backtest methodology, the Jump Models are fitted on the last day of the month and so, without updating, the model parameters do not change intra-month.

Parameter updating of a given Jump Model would help the model adapt better to incoming daily data and navigate sharp movements in financial markets such as an equities sell-off and/or a volatility spike. This work proposes an averaging method to update the model parameters $\underset{\sim}{\boldsymbol{\mu}}$. Under our approach, the $\hat{s}_t^{\text{th}}$ row of $\underset{\sim}{\boldsymbol{\mu}}$ is updated, where $\hat{s}_t$ is the estimated state at time t.

In mathematical terms, we firstly add a time superscript to $\underset{\sim}{\boldsymbol{\mu}}$ and denote it $\underset{\sim}{\boldsymbol{\mu}}^t \in \mathbb{R}^{2\times 66}$. Furthermore, let $\boldsymbol{\mu}_{\hat{s}_t}^t \in \mathbb{R}^{66}$ be the $\hat{s}_t^{\text{th}}$ row of $\underset{\sim}{\boldsymbol{\mu}}^t$. $\boldsymbol{\mu}_{\hat{s}_t}^t$ is updated by the averaging equation

$$\boldsymbol{\mu}_{\hat{s}_t}^t = \boldsymbol{\mu}_{\hat{s}_t}^{t-1} + \alpha_{t-1}\left(\boldsymbol{z}_t - \boldsymbol{\mu}_{\hat{s}_t}^{t-1}\right), \tag{4.2.2}$$

where $\alpha_t \in [0, 1]$ is the step size at time t. If $\alpha_t = 1/t$, then (4.2.2) is a simple moving-average (SMA) of the model parameters. If $\alpha_t \equiv \alpha$, then (4.2.2) is an exponential moving-average (EMA) with smoothing factor α.

The SMA assigns equal weight to all past values of $\boldsymbol{\mu}_{\hat{s}_t}^t$ while the EMA assigns weights that decrease exponentially over time. For α close to one, the EMA assigns more weight to recent observations which makes the EMA more quickly respond to changes in the incoming data. For α close to zero, the converse holds: there is less weight given to recent observations which makes the EMA less responsive to changes in the incoming data.

The model training and state estimation process is visualised in Figure 4.2. The proposed online learning algorithm is given in Algorithm 9.

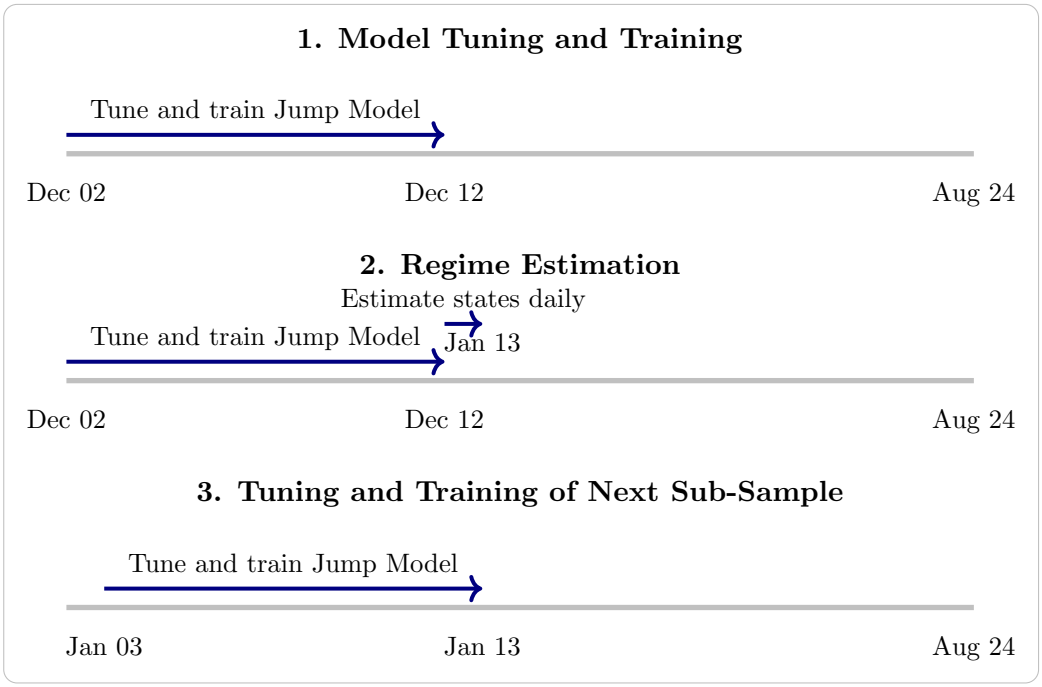

Figure 4.2: Process for training of Features Dataset.

Algorithm 9 Online learning of state sequence

Input: Initial estimates of model parameters and state ($\hat{\underline{\boldsymbol{\mu}}}^0$ and $\hat{s}_0$ respectively), tuned jump penalty λ, standardised observation $\boldsymbol{z}_t$, prior arrival cost function $\mathcal{A}_{t-1}$, and step size parameter α_t:

Iterate for $t = 1, 2, \ldots, 29$,

$$\mathcal{A}_t(s_t) \leftarrow \min_{s_{t-1}} \{\|\boldsymbol{z}_t - \hat{\boldsymbol{\mu}}_{s_{t-1}}^{t-1}\|^2 + \mathcal{A}_{t-1}(s_{t-1}) + \lambda \mathbf{1}_{\{s_{t-1} \neq s_t\}}\}, \tag{9.1}$$

$$\hat{s}_t \leftarrow \arg\min_{s} \{\|\boldsymbol{z}_t - \hat{\boldsymbol{\mu}}_{s}^{t-1}\|^2 + \mathcal{A}_t(s)\}, \tag{9.2}$$

$$\hat{\boldsymbol{\mu}}_{\hat{s}_t}^{t} \leftarrow \hat{\boldsymbol{\mu}}_{\hat{s}_t}^{t-1} + \alpha_{t-1}\left(\boldsymbol{z}_t - \hat{\boldsymbol{\mu}}_{\hat{s}_t}^{t-1}\right). \tag{9.3}$$

Output: Sequence of arrival functions $(\mathcal{A}_1, \mathcal{A}_2, \ldots, \mathcal{A}_{29})$, estimated states $(\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_{29})$ and parameters $\left(\hat{\underline{\boldsymbol{\mu}}}^1, \hat{\underline{\boldsymbol{\mu}}}^2, \ldots, \hat{\underline{\boldsymbol{\mu}}}^{29}\right)$.

4.2.3 Trading Strategies

The states estimated from Algorithm 9 are then used to trade the CDS contract. The trading strategies considered are based on the stylised fact of financial markets that bear markets are generally characterised by a sharp rise in corporate credit spreads, reflecting heightened credit risk in corporate borrowers. This can be visualised by Figure 4.3, in which we have the same spreads shown in Figure 4.1 shaded with the regime estimates of the Regularised $\mathcal{P}_3$ Jump Model.

We can see from Figure 4.3 that time periods shaded with red coincide most of

the time with a sharp rise in the spread of the CDS contract. Therefore, buying CDS protection during these time periods would generate positive profits. Conversely, during time periods shaded green, credit spreads are generally at lower levels and are relatively stable. In those periods, it is more likely that selling CDS protection would generate positive profits.

Therefore, we've devised two simple trading strategies based on the above trading logic. Let $\hat{s}_t \in \{1, 2\}$ be the estimated state at time t, where $s_t = 1$ denotes a normal/bull market and $s_t = 2$ denotes a bear market. Additionally, let w_t be the weight on the CDS contract. The two trading strategies are expressed mathematically as

Strategy 1 (Buying 100% Protection): $w_t = \hat{s}_t - 1$,

Strategy 2 (Buying or Selling 100% Protection): $w_t = 2\hat{s}_t - 3$,

where $w_t \in \{0, 1\}$ in Strategy 1 and $w_t \in \{-1, 1\}$ in Strategy 2. Assuming zero trading costs, the daily return of the strategy is calculated as $w_{t-1}\hat{r}_t$, where $\hat{r}_t$ is the CDS return approximation in (4.2.1).

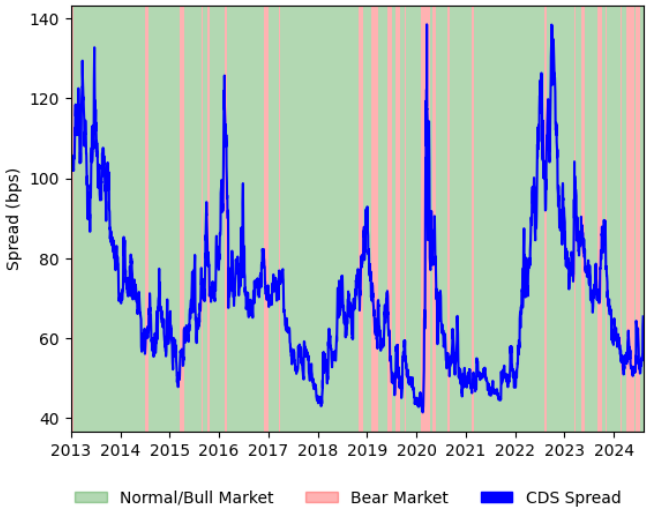

Figure 4.3: EUR iTraxx Main 5Y Spreads shaded with state estimates from the Regularised $\mathcal{P}_3$ Jump Model. The states were estimated using the methodology visualised in Figure 4.2.

4.3 Backtest Results

The six Jump Models were backtested using the methodology outlined in Section 4.2. Jump Models that employ parameter updating (Step 9.3) were also tested. More specifically, we tested all six Jump Models with a SMA parameter update, and EMA parameter updates with smoothing factors $\alpha = 0.25, 0.5, 0.75$.

We found that the state estimates of the Jump Models with these parameter update configurations were identical to those without them. Therefore, the strategy performances of Jump Models with parameter updates were excluded.

4.3.1 Strategy Performance and Benchmarking

Figures 4.4 and 4.5 depict the total returns and drawdowns of Strategies 1 and 2 respectively, for each Jump Model tested. Tables 4.4 and 4.5 provide some performance metrics of Strategy 1 and 2 respectively; an overview of the calculations behind each performance metric is given in A.4.

Furthermore, Figures 4.6 and 4.7 show the same total returns of Strategies 1 and 2 respectively as that in Figures 4.4 and 4.5, compared to the total returns of a benchmark. The benchmark return was calculated as the return of the CDS contract multiplied by the average weight of the strategy being benchmarked. The average weights for both strategies and all Jump Models are shown in Table 4.3.

Denoting the average weight of the strategy as $\bar{w} := \frac{1}{T}\sum_{t=1}^{T} w_t$, the daily return of the benchmark at time t was calculated as $\bar{w}\hat{r}_t$, where $\hat{r}_t$ was calculated using (4.2.1). Figures 4.6 and 4.7 therefore compare dynamic CDS strategies, whose weights change according to the state estimates of a given Jump Model, with static CDS strategies such that their average exposures are equivalent.

Referring to the state estimates from each Jump Model in Figure A.2, the state estimates from the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ Jump Models can be described as a normal/bull market interspersed with short-lived bear market periods. The state estimates from the Standard, Sparse and Regularised $\mathcal{P}_0$ Jump Models, on the other hand, switch more often between normal/bull and bear markets.

These descriptions can be quantified by the empirical transition probability matrices in Figure A.3 where the probabilities of staying in a normal/bull market estimated by the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ Jump Models proved to be higher than those estimated by the other models.

Figures 4.4 and 4.5 show that the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ Jump Models outperform the other three Jump Models in both Strategies 1 and 2. One period of outperformance from the three models was in March 2020: the strategies corresponding to the Regularised $\mathcal{P}_1, \mathcal{P}_2$ and $\mathcal{P}_3$ Jump Models timed the purchase of CDS protection better than the other models when corporate credit spreads rose sharply in response to the events surrounding the COVID-19 pandemic.

In Strategy 1, all Jump Models generate a negative average return (shown by the AAGR and CAGR in Table 4.1). This is characteristic of strategies that buy CDS protection. Buying CDS protection is an example of a positively skewed strategy that incurs regular, small losses, represented by the CDS premiums paid to the protection seller, but has the potential of generating substantial profits during rare events, as empirically shown by the market events of March 2020.

Furthermore, Figure 4.6 shows that all strategies underperform their benchmarks except the Regularised $\mathcal{P}_3$ Jump Model. This provides evidence that dynamically buying CDS protection based on the Regularised $\mathcal{P}_3$ Jump Model can outperform a static strategy that buys a fixed amount of CDS protection, and thereby reduce the negative carry associated with buying CDS protection.

In Strategy 2, Table 4.2 shows that the Regularised Jump Models generate a positive average return, as opposed to the Standard and Sparse Jump Models. Referring to

Table 4.3, the negative weights for the Strategy 2 row indicate that all strategies are benchmarked against a strategy that sells CDS protection.

The relative performance of the Regularised $\mathcal{P}_3$ Jump Model in Figure 4.7 provides evidence that a dynamic strategy that buys and sells CDS protection based on the model's state estimates, outperforms a strategy that sells a fixed amount of CDS protection.

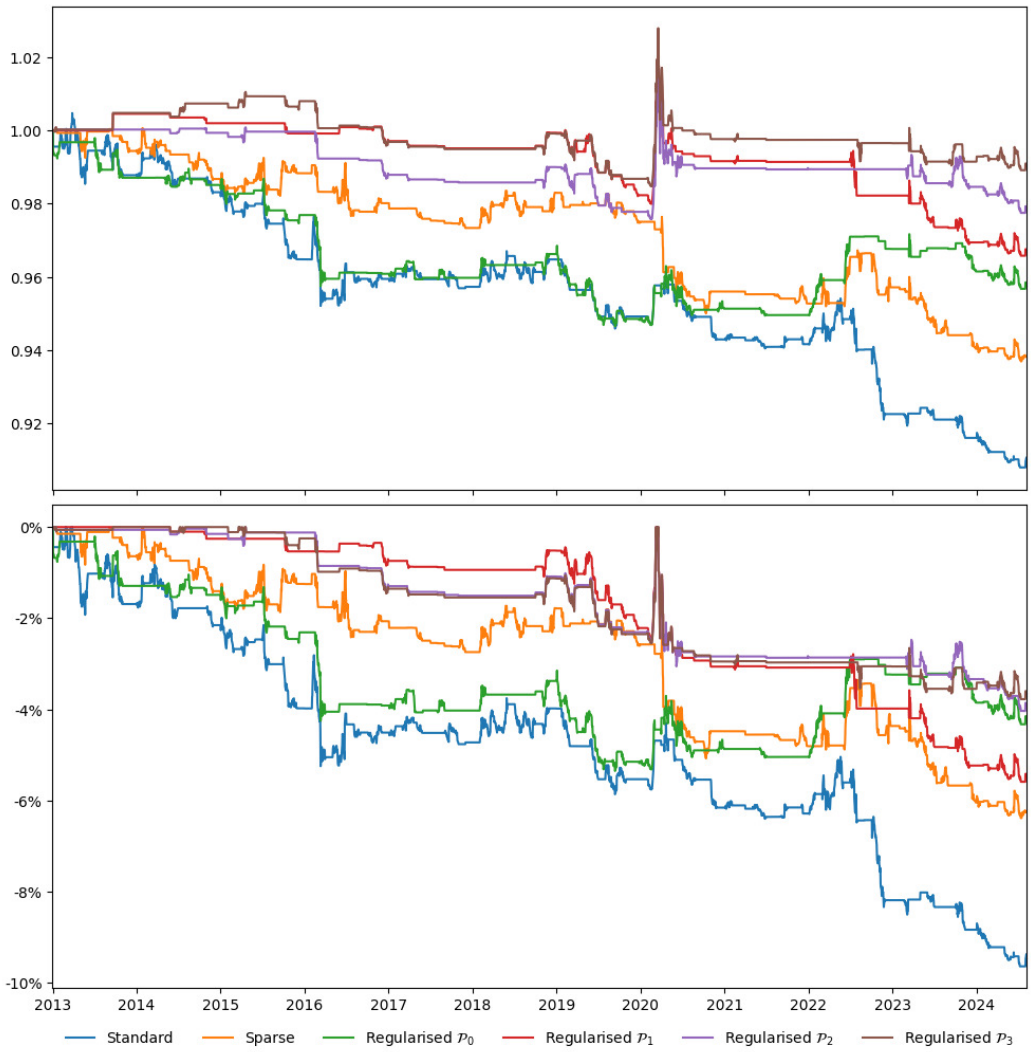

Figure 4.4: Total returns and drawdowns of Strategy 1.

	Standard	Sparse	Regularised $\mathcal{P}_0$	Regularised $\mathcal{P}_1$	Regularised $\mathcal{P}_2$	Regularised $\mathcal{P}_3$
AAGR (%)	-0.79	-0.53	-0.35	-0.27	-0.17	-0.07
CAGR (%)	-0.80	-0.50	-0.36	-0.28	-0.18	-0.08
Volatility (%)	1.12	1.02	0.86	0.97	0.94	0.95
Maximum Drawdown (%)	9.63	6.38	5.35	5.59	4.03	3.78
Sharpe Ratio	-0.69	-0.52	-0.41	-0.28	-0.18	-0.07
Calmar Ratio	-0.08	-0.09	-0.07	-0.05	-0.04	-0.02

Table 4.1: Strategy 1 Performance Metrics (2.d.p.).

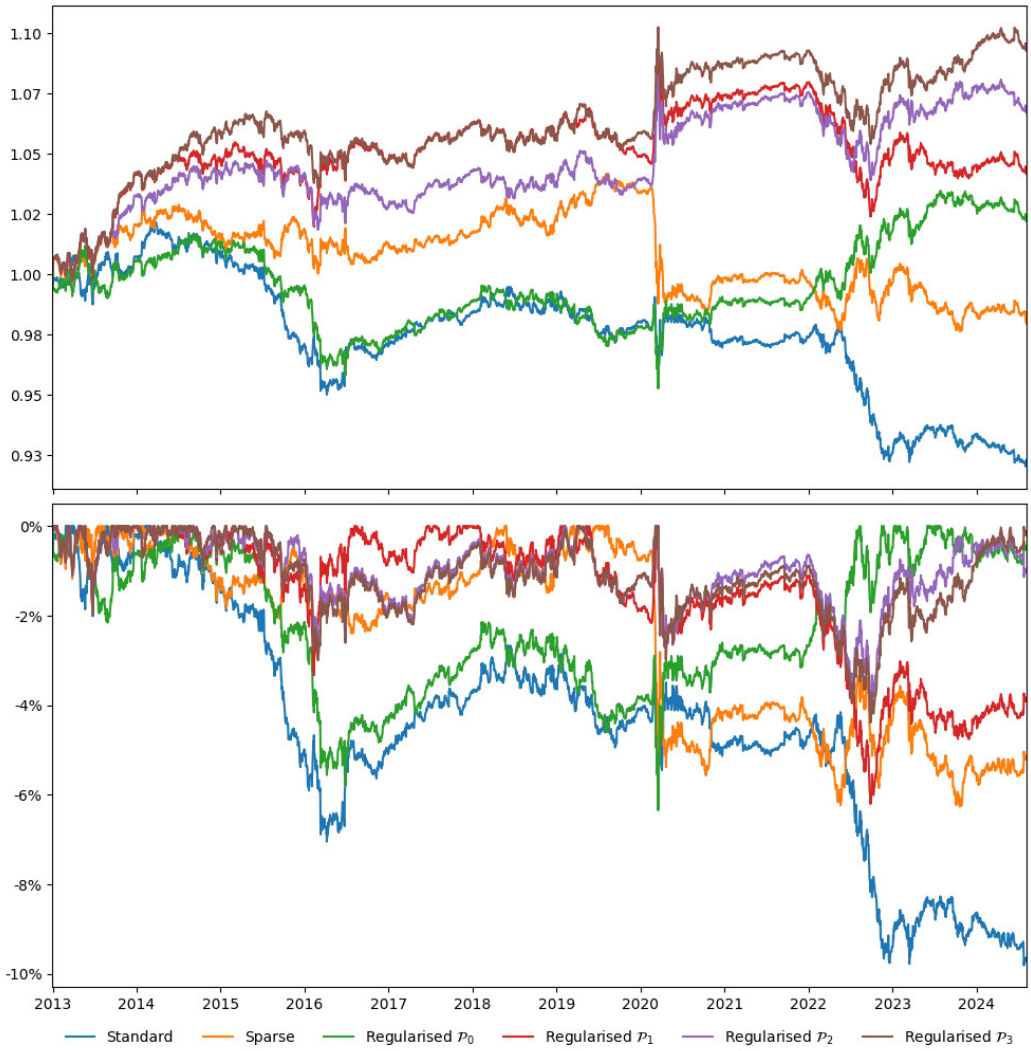

Figure 4.5: Total returns and drawdowns of Strategy 2.

	Standard	Sparse	Regularised $\mathcal{P}_0$	Regularised $\mathcal{P}_1$	Regularised $\mathcal{P}_2$	Regularised $\mathcal{P}_3$
AAGR (%)	-0.65	-0.16	0.20	0.35	0.55	0.75
CAGR (%)	-0.69	-0.18	0.19	0.35	0.56	0.77
Volatility (%)	1.73	1.73	1.73	1.73	1.73	1.73
Maximum Drawdown (%)	9.81	6.27	6.35	6.21	4.01	4.39
Sharpe Ratio	-0.38	-0.09	0.11	0.20	0.32	0.43
Calmar Ratio	-0.07	-0.03	0.03	0.06	0.14	0.17

Table 4.2: Strategy 2 Performance Metrics (2.d.p.).

	Standard	Sparse	Regularised $\mathcal{P}_0$	Regularised $\mathcal{P}_1$	Regularised $\mathcal{P}_2$	Regularised $\mathcal{P}_3$
Strategy 1 (%)	47.2	36.4	28.9	16.4	15.6	15.5
Strategy 2 (%)	-5.7	-27.2	-42.2	-67.3	-68.8	-69.1

Table 4.3: Average weights (1.d.p.) for both strategies and all Jump Models.

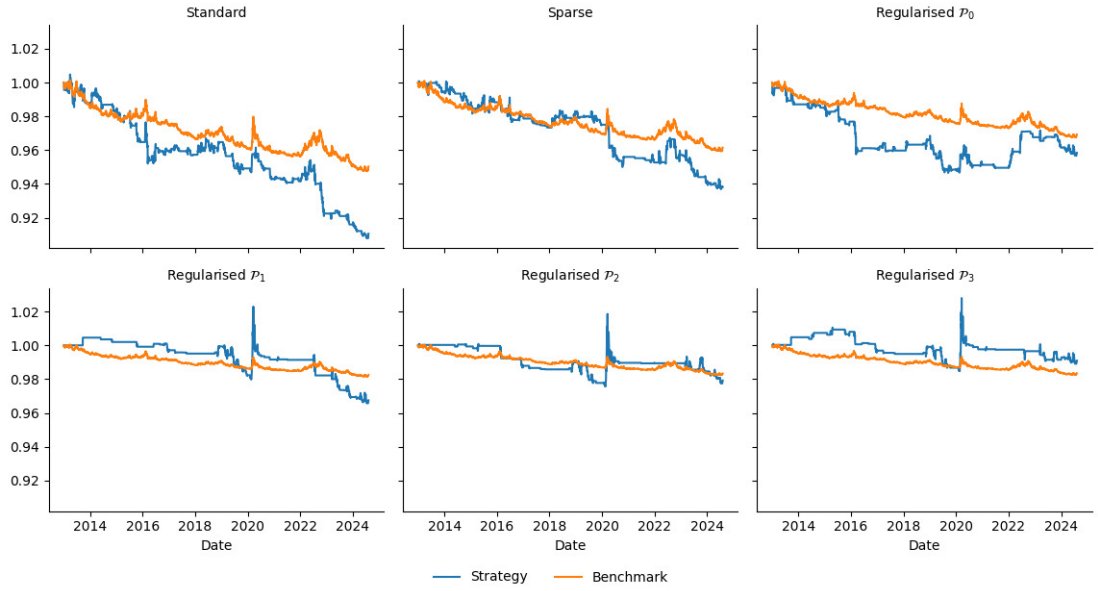

Figure 4.6: Total returns and drawdowns of Strategy 1 and its benchmark.

Figure 4.7: Total returns and drawdowns of Strategy 2 and its benchmark.

4.3.2 Feature Selection

The various Jump Models were calibrated onto the Features Dataset on the last day of every month covered by the dataset. Therefore, in the Sparse Jump Model, a monthly series of estimated feature weights $\boldsymbol{w} \in \mathbb{R}^{66}$ is generated. Similarly, the monthly calibration of the Regularised Jump Models produces a series of estimates of $\underline{\boldsymbol{\mu}} \in \mathbb{R}^{2\times 66}$.

The time-series interpretation of $p = 66$ feature parameters was a difficult task and so we simplified the task by summing the feature parameters according to the six categories that the features were assigned to (see Table A.3 for reference).

The feature weights estimated by the Sparse Jump Model were summed according to the categories to which the features were assigned, and then normalised such that the sum of the weights equalled 100%.The blue lines in Figure 4.8 show the feature category weights estimated by the Sparse Jump Model.

Similarly, in the monthly calibrations of the Regularised $\mathcal{P}_0, \mathcal{P}_1$ and $\mathcal{P}_3$ Jump Models, we used (3.5.2) by assigning a value of one to each feature whose corresponding column had a strictly positive Euclidean distance, and a value of zero otherwise.

The array of active features (ω_j) was divided by the array's sum and then summed according to the feature's category. The blue lines in Figures 4.9, 4.10 and 4.11 show the feature category weights of the Regularised $\mathcal{P}_0, \mathcal{P}_1$ and $\mathcal{P}_3$ Jump Models respectively. Due to the oscillations in the feature category weights, we added a orange line which depicts the rolling two-year average of the feature category weights.

Comparing the four figures, the feature category weights from the Sparse Jump Model are more dispersed than those from the Regularised Jump Models. The feature category weights from the Regularised $\mathcal{P}_0$ Jump Model are the least dispersed over time.

This is because the parameter update equation for the Regularised $\mathcal{P}_0$ adopts hard-thresholding. The cluster centres of features whose contribution to the model fit falls below a certain threshold, are set to zero while those that exceed the threshold remain unchanged (see Proposition A.1.3 for more details). The feature category weights from the Regularised $\mathcal{P}_1$ and $\mathcal{P}_3$ Jump Models are similar to each other and are more dispersed than those from the Regularised $\mathcal{P}_0$ Jump Model.

A common characteristic of the four figures is the high weight given to features in the Credit Valuation and Momentum category. This characteristic suggests that metrics pertaining to US and European corporate bonds are strongly associated with market regime dynamics. Furthermore, the high relative weight given to the category provides evidence that the information contained in the underlying metrics is shared with metrics from the other five categories.

Many empirical studies have attested to the high informational content of metrics pertaining to corporate bonds. For instance, [14] construct a credit spread index based on an extensive micro-level dataset of secondary market prices of corporate bonds and determine that the index is a robust predictor of economic activity across a variety of economic indicators.

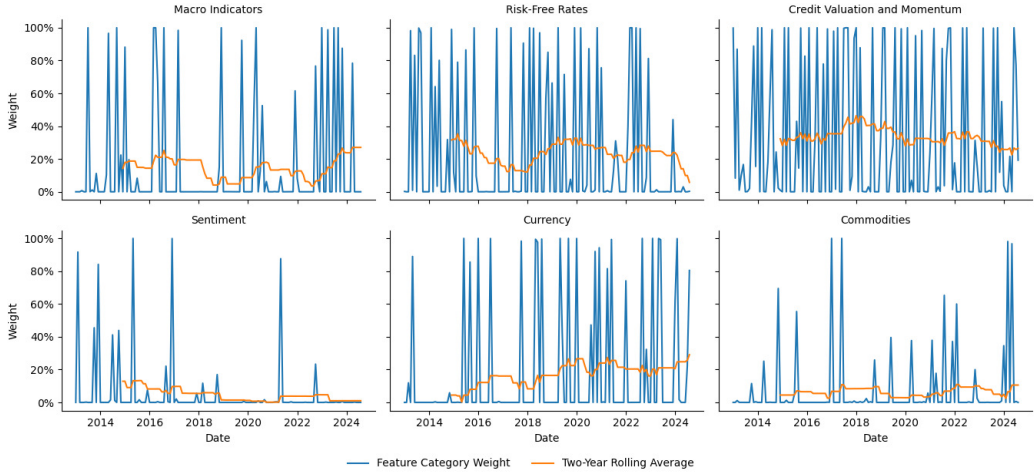

Figure 4.8: Feature category weights from the Sparse Jump Model.

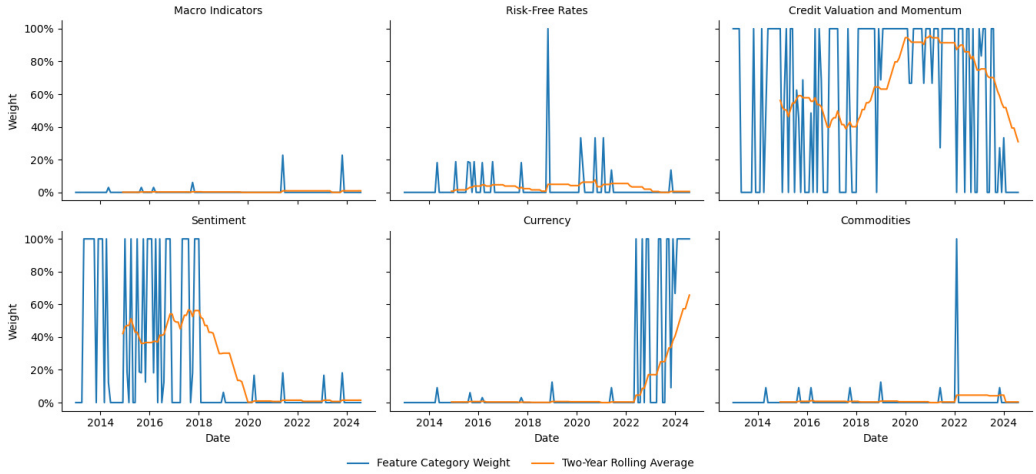

Figure 4.9: Feature category weights from the Regularised $\mathcal{P}_0$ Jump Model.

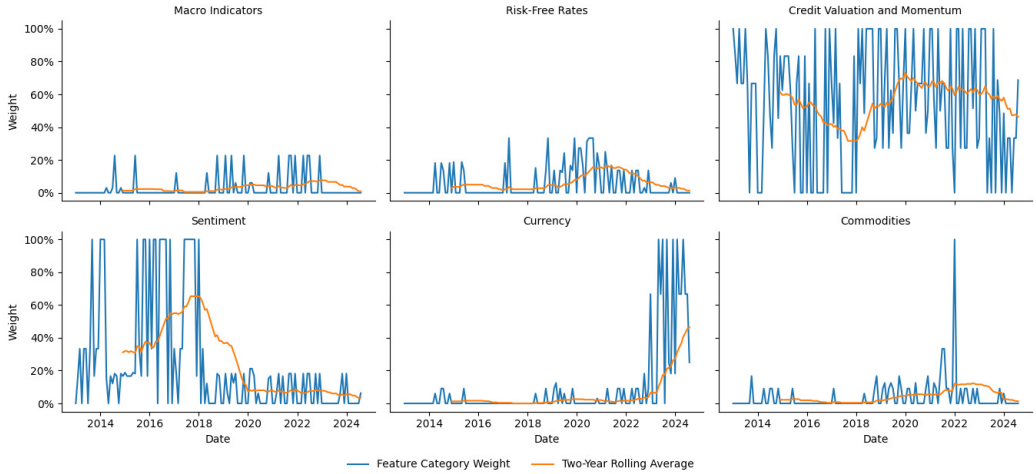

Figure 4.10: Feature category weights from the Regularised $\mathcal{P}_1$ Jump Model.

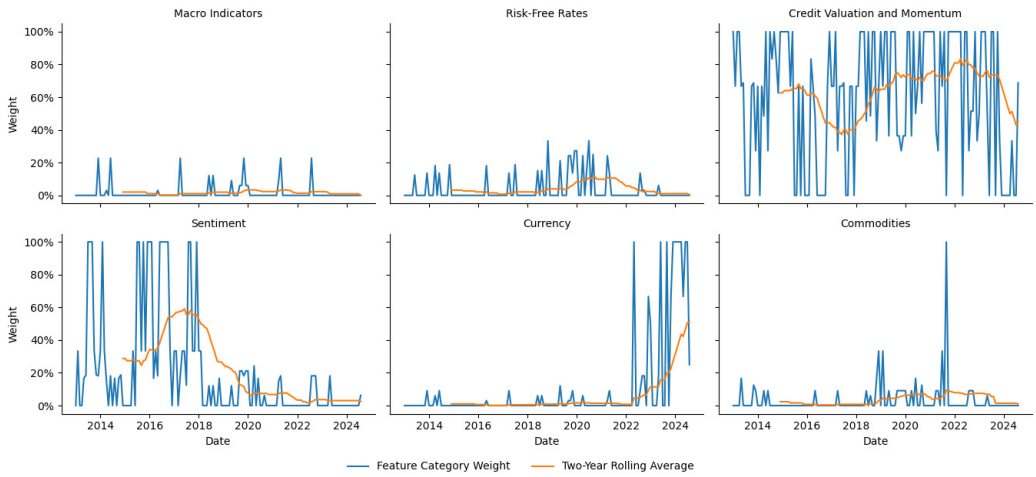

Figure 4.11: Feature category weights from the Regularised $\mathcal{P}_3$ Jump Model.

Conclusion

In this work, we've made the following original contributions.

Proposal of a new class of Jump Models: In the new class of models called *Regularised Jump Models*, the regularised K-means model proposed in [25] has been adapted to the Jump Model Framework. These models perform state and parameter estimation, and feature selection jointly. The feature selection process performed by the Regularised Jump Models is more direct and interpretable than that from the Sparse Jump Model, which performs feature selection by proxy.

These models were tested in a simulation experiment and demonstrate evidence of outperformance over existing Jump Models. In a backtest of CDS trading strategies, we found that strategies based on the Regularised Jump Models produced risk-adjusted returns superior to those of the Standard and Sparse Jump Models.

One avenue of future research is adapting the asymptotic properties of the Regularised K-Means model proven in [25] to the Regularised Jump Models. Some of these properties include consistency and strong consistency in terms of the Hausdorff distance (see [25] Theorem 6 and the proof in Appendix A. A.3).

Hyperparameter tuning: We've proposed a new hyperparameter tuning method for Jump Models, based on the idea of clustering stability. There is scope for future research on developing and testing alternative hyperparameter tuning methods for Jump Models.

Extension of online learning algorithm for Jump Models: We've extended the online learning algorithm in [20] by adding a step that recursively updates the parameters of a given Jump Model. We suggest testing this extension in a simulation experiment similar to the one conducted in [20].

Empirical calibration onto dataset containing macroeconomic factors: To the best of our knowledge, our work is first in calibrating Jump Models onto a dataset containing both market and macroeconomic features.

The empirical calibration has overall determined that features related to corporate bonds have the highest influence in transitions between market regimes. We suggest further work on the calibration of Jump Models onto macroeconomic and market features to empirically assess this finding.

Appendix A

Technical Proofs and Supplementary Material

A.1 Parameter Update Equations for Regularised Jump Model Calibration

Before giving the proposition on the parameter update equations for the Regularised Jump Models, we prove the following useful lemma:

Lemma A.1.1. *Let* $\boldsymbol{Y} := (\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T)$ *be a* $T \times p$ *data matrix and* $\boldsymbol{M}$ *be a* $T \times K$ *cluster assignment matrix with elements* $m_{t,k} = \mathbf{1}_{\{\boldsymbol{y}_t \in C_k\}}$. (2.1.1) *can be rewritten as*

$$\sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2 = \sum_{j=1}^{p} ||\boldsymbol{Y}_{.,j} - \boldsymbol{M}\boldsymbol{\mu}_{.,j}||,$$

where $\boldsymbol{Y}_{.,j}$ *and* $\boldsymbol{\mu}_{.,j}$ *are the* j^{th} *columns of* $\boldsymbol{Y}$ *and* $\underline{\boldsymbol{\mu}}$ *respectively.*

Proof. A.1.1 follows from the following equality:

$$\begin{aligned}\sum_{k=1}^{K} \sum_{t \in C_k} \|\boldsymbol{y}_t - \boldsymbol{\mu}_k\|^2 &= \sum_{k=1}^{K} \sum_{t \in C_k} \sum_{j=1}^{p} (y_{t,j} - \mu_{k,j})^2 \\ &= \sum_{j=1}^{p} \left(\boldsymbol{Y}_{.,j} - \boldsymbol{M}\boldsymbol{\mu}_{.,j}\right)^T \left(\boldsymbol{Y}_{.,j} - \boldsymbol{M}\boldsymbol{\mu}_{.,j}\right)\end{aligned}$$

□

Remark A.1.2. The result in A.1.1 shows that the WCSS can be decomposed additively across the features in $\boldsymbol{Y}$. The contribution to the WCSS from the j^{th} feature is $\|\boldsymbol{Y}_{.,j} - \boldsymbol{M}\boldsymbol{\mu}_{.,j}\|$.

Proposition A.1.3. *Suppose that we have an assignment of the elements of* Y *into* K *clusters* $C_1, C_2, \ldots, C_K$. *Let* $|C_k|$ *denote the number of elements in cluster* k. *Furthermore, let* $\boldsymbol{M}$ *be a* $T \times K$ *cluster assignment matrix with elements* $m_{t,k} = \mathbf{1}_{\{\boldsymbol{y}_t \in C_k\}}$.

Let $\underline{\boldsymbol{\mu}}^*$ *be the corresponding* $K \times p$ *matrix of cluster centres and* $\underline{\mu}^*_{k,j}$ *be the* (k, j) *element of* $\underline{\boldsymbol{\mu}}^*$ *(*k^{th} *cluster centre along the* j^{th} *dimension). Keeping the cluster assignment matrix* $\boldsymbol{M}$ *fixed, solving* (3.3.1) *gives the following expressions for each penalty function* $\mathcal{P}\left(\underline{\boldsymbol{\mu}}\right)$*:*

$$\mathcal{P}_0\left(\underline{\boldsymbol{\mu}}\right):\ \underline{\mu}_{k,j} = \begin{cases} \underline{\mu}^*_{k,j} & \text{if } \|\boldsymbol{y}_{.,j}\|^2 > \|\boldsymbol{y}_{.,j} - \boldsymbol{M}\underline{\boldsymbol{\mu}}^*_{.,j}\| + T\gamma \\ 0 & \text{otherwise} \end{cases} \tag{A.1.1a}$$

$$\mathcal{P}_1\left(\underline{\boldsymbol{\mu}}\right): \underline{\mu}_{k,j} = \max\left(0, 1 - \frac{T\gamma}{2|C_k||\underline{\mu}^*_{k,j}|}\right)\underline{\mu}^*_{k,j} \tag{A.1.1b}$$

$$\mathcal{P}_2\left(\underline{\boldsymbol{\mu}}\right): \underline{\mu}_{k,j} = \frac{1}{1 + \frac{T\gamma}{|C_k|}}\underline{\mu}^*_{k,j} \tag{A.1.1c}$$

$$\mathcal{P}_3\left(\underline{\boldsymbol{\mu}}\right): \underline{\mu}_{k,j} = \frac{1}{1 + \frac{T\gamma}{2|C_k|\|\underline{\boldsymbol{\mu}}_{.,j}\|}}\underline{\mu}^*_{k,j} \quad \text{if } \underline{\boldsymbol{\mu}}_{.,j} \neq \mathbf{0}_K, \tag{A.1.1d}$$

where $\boldsymbol{y}_{.,j}$ is the j^{th} column of Y and $\mathbf{0}_K$ is a zero vector of size K.

Proof. Proof can be found in A.2. of the Appendix in [25]. □

Remark A.1.4. The update equations in (A.1.1) lend insight into the effects of each penalty function. Firstly, to understand the update in (A.1.1a), we use the result in A.1.1 and the associated Remark A.1.2. The term $\|\boldsymbol{y}_{.,j} - \boldsymbol{M}\underline{\boldsymbol{\mu}}^*_{.,j}\|$ is the WCSS contribution fron the j^{th} feature.

$\mathcal{P}_0$ leads to hard-thresholding, the process of setting to zero the elements of an input vector whose absolute values are lower than an input threshold value; elements whose absolute values exceed the threhold are left unchanged. The threshold value in (A.1.1a) is the sum of $T\gamma$ and the WCSS contribution from each feature. The interpretation of (A.1.1a) is that variables are included in the clustering if it sufficiently contributes to a decrease of the WCSS.

$\mathcal{P}_1$ leads to soft-thresholding, whereby some elements are set to zero and the rest are shrunk towards zero. This resembles the solution of $\mathcal{L}_1$-regularised (or Lasso) regression with orthonormal covariates. $\mathcal{P}_2$ is a ridge-type penalty that scales down each element of the array uniformly. $\mathcal{P}_2$ is the only penalty function that does not directly induce sparsity.

$\mathcal{P}_3$ does not have an explicit updating equation since the right-hand side of (A.1.1d) includes the $\mathcal{L}_2$ norm of $\|\boldsymbol{\mu}_{.,j}\|$. The solution is thus found through an iterative algorithm.

A.2 Introduction to Credit Default Swaps

A Credit Default Swap (CDS) is a swap agreement between two parties in which one party (Protection Seller) agrees to compensate another party (Protection Buyer) in the event of a debt default during the term of the agreement. In other words, the Protection Seller agrees to insure the Protection Buyer against a reference asset (Reference Entity) defaulting. In exchange, the Protection Buyer makes a series of regular payments.

The price of a CDS contract is typically quoted in terms of the CDS "break-even" spread or simply CDS spread; this metric reflects the annualised payment that the Protection Buyer would make per unit of CDS protection. For instance, if the CDS spread of the Reference Entity is 50 basis points (0.05%), then an investor buying a CDS contract insuring 10 million Sterling would make annual payments worth 50,000 Sterling. Payments

are usually on a quarterly basis and continue until either the CDS contract expires or the Reference Entity defaults.

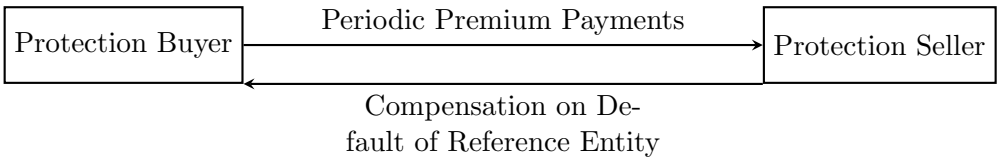

Figure A.1: Payoff structure of a standard CDS contract.

A.2.1 Index CDS

The example depicted in A.1 is called a *single-name CDS* because the borrower is a single entity. A second type of CDS, an *index CDS*, involves a combination of borrowers. This type of CDS allows participants to take positions on the credit risk of a combination of borrowers, in much the same way that investors can trade exchange-traded funds (ETFs) that are combinations of equities of various companies.

IHS Markit is the major provider of CDS indexes. The Markit indexes are classified by region as well as by credit quality. North American indexes are identified by the symbol CDX, and European, Asian, and Australian indexes are identified as iTraxx. Within each geographic category are investment-grade and high-yield indexes. The former are identified as CDX IG and iTraxx Main, each comprising 125 entities. The latter are identified as CDX HY, consisting of 100 entities, and iTraxx Crossover, consisting of up to 50 high-yield entities.

A.3 Data Series for Empirical Calibration

Series Name	**Series Category**	**Series Data Type**
Oil	Commodities	Price
Gold	Commodities	Price
US IG Credit Total Return	Credit Valuation/Momentum	Price
EUR IG Credit Total Return	Credit Valuation/Momentum	Price
US IG Credit OAS	Credit Valuation/Momentum	Percentage
EUR IG Credit OAS	Credit Valuation/Momentum	Percentage
US HY Credit OAS	Credit Valuation/Momentum	Percentage
EUR HY Credit OAS	Credit Valuation/Momentum	Percentage
US Dollar Index	Currency	Index
Trade-Weighted US Dollar Index	Currency	Index
Citi US Policy Uncertainty	Macro Indicators	Index
Citi US Economic Surprises	Macro Indicators	Index
Citi EUR Economic Surprises	Macro Indicators	Index
Citi Emerging Markets Economic Surprises	Macro Indicators	Index
Citi Global Economic Surprises	Macro Indicators	Index
US 10Y Government Bond Yield	Risk-Free Rate	Percentage
US 2Y Government Bond Yield	Risk-Free Rate	Percentage
US Government Bond Curve (10 Yield minus 2 Yield)	Risk-Free Rate	Percentage
VIX Index	Sentiment	Index
MOVE Index	Sentiment	Index
SKEW Index	Sentiment	Index
CBOE US Equity Put/Call Ratio	Sentiment	Index

A.4 Calculation of Performance Metrics

Let $(r_1, r_2, \ldots, r_T)$ be a sequence of daily returns of size T. Assume there are 252 business days and that the risk-free rate r_f is equal to zero.

AAGR: The Average Annual Growth Rate (AAGR) is defined as

$$\text{AAGR} = \frac{252}{T} \sum_{t=1}^{T} r_t.$$

CAGR: The Compound Annual Growth Rate (CAGR) is defined as

$$\text{CAGR} = \Big(\prod_{t=1}^{T} (1 + r_t) \Big)^{\frac{252}{T}}.$$

Volatility: The annualised volatility is

$$\text{Volatility} = \sqrt{\frac{252}{T-1} \sum_{t=1}^{T} (r_t - \bar{r_t})^2}.$$

Maximum Drawdown: The total return at time t, denoted P_t, can be written as

$$P_t = \prod_{i=1}^{t} (1 + r_i)\,. \tag{A.4.1}$$

We also define the High Watermark (HWM) at time t as $\text{HWM}_t := \max_{1 \leq t} P_t$. The Maximum Drawdown can then be written as

$$\text{Maximum Drawdown} = \frac{P_t - \text{HWM}_t}{\text{HWM}_t}.$$

Sharpe Ratio : The Sharpe Ratio is defined as

$$\text{Sharpe Ratio} = \frac{\text{AAGR}}{\text{Volatility}}.$$

Calmar Ratio : The Calmar Ratio is defined as

$$\text{Calmar Ratio} = \frac{\text{CAGR}}{\text{Maximum Drawdown}}.$$

A.5 Jump Model State Estimates

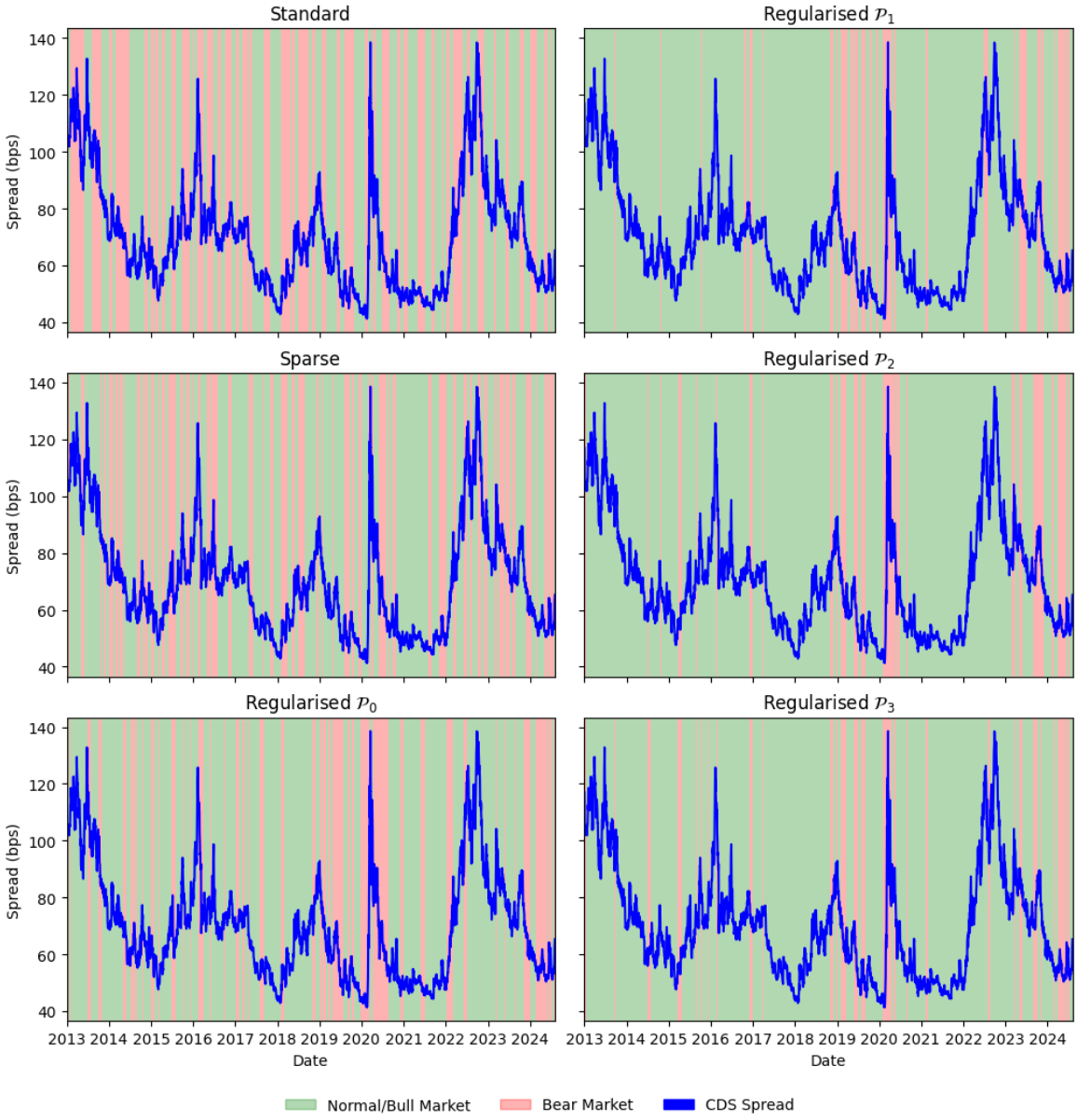

Figure A.2: EUR iTraxx Main 5Y Spreads shaded with empirical state estimates from each Jump Model.

A.6 Empirical Transition Probability Matrices

Let $(\hat{s}_1, \hat{s}_2, \ldots, \hat{s}_T)$ denote the sequence of state estimates, where $\hat{s}_t \in \{1, 2\}$, $t = , 1, 2, \ldots, T$. The estimated transition probability matrix $\hat{\Pi} := (\hat{\pi}_{i,j}) \in \mathbb{R}^{2\times 2}$ has entries

$$\hat{\pi}_{i,j} = \frac{\sum_{t=1}^{T-1} \mathbf{1}_{\{s_{t+1}=j | s_t=i\}}}{\sum_{t=1}^{T} \mathbf{1}_{\{s_t=i\}}}.$$

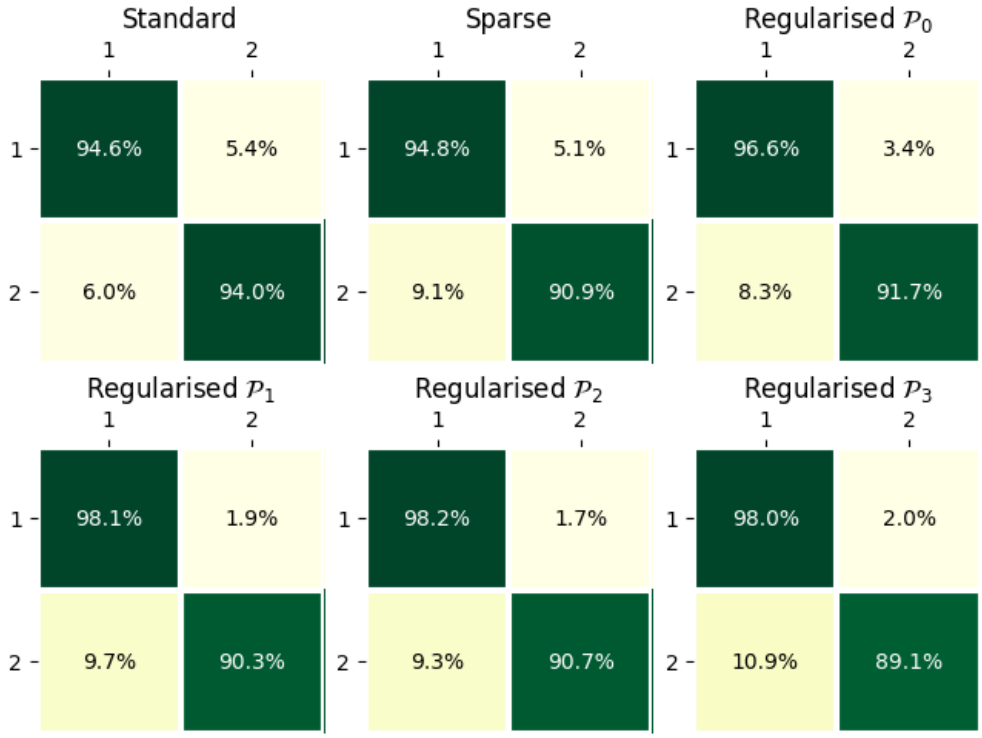

Figure A.3: Empirical transition probability matrices of states estimated from the Jump Models (matrix entries in 1.d.p.).

Bibliography

[1] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, *Optuna: A next-generation hyperparameter optimization framework*, in The 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.

[2] D. Arthur and S. Vassilvitskii, *k-means++: the advantages of careful seeding*, in Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '07, USA, 2007, Society for Industrial and Applied Mathematics, p. 1027–1035.

[3] A. O. Aydinhan, *Essays on Advanced Methods in Portfolio Optimization*, PhD thesis, Princeton University, 2023.

[4] A. O. Aydinhan, P. N. Kolm, J. M. Mulvey, and Y. Shu, *Identifying patterns in financial markets: extending the statistical jump model for regime identification*, Annals of Operations Research, (2024).

[5] A. Bemporad, V. Breschi, D. Piga, and S. P. Boyd, *Fitting jump models*, Automatica, 96 (2018), pp. 11–21.

[6] A. Ben-Hur, A. Elisseeff, and I. Guyon, *A stability based method for discovering structure in clustered data*, Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing, 2002 (2002), pp. 6–17.

[7] S. Boyd and L. Vandenberghe, *Convex Optimization*, Cambridge University Press, 2004.

[8] K. H. Brodersen, C. S. Ong, K. E. Stephan, and J. M. Buhmann, *The balanced accuracy and its posterior distribution*, in 2010 20th International Conference on Pattern Recognition, 2010, pp. 3121–3124.

[9] J. Bulla, *Hidden markov models with t components. increased persistence and other aspects*, Quantitative Finance, 11 (2011), pp. 459–475.

[10] A. B. Chan and N. Vasconcelos, *Modeling, clustering, and segmenting video with mixtures of dynamic textures*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 30 (2008), pp. 909–926.

[11] F. Cortese, P. Kolm, and E. Lindstrom, *Generalized information criteria for high-dimensional sparse statistical jump models*, SSRN Electronic Journal, (2024).

[12] Y. Fan and C. Y. Tang, *Tuning parameter selection in high dimensional penalized likelihood*, Journal of the Royal Statistical Society. Series B (Statistical Methodology), 75 (2013), pp. 531–552.

[13] Z. Ghahramani and M. Jordan, *Factorial hidden markov models*, in Advances in Neural Information Processing Systems, D. Touretzky, M. Mozer, and M. Hasselmo, eds., vol. 8, MIT Press, 1995.

[14] S. Gilchrist and E. Zakrajšek, *Credit spreads and business cycle fluctuations*, Working Paper 17021, National Bureau of Economic Research, May 2011.

[15] J. D. Hamilton, *A new approach to the economic analysis of nonstationary time series and the business cycle*, Econometrica, 57 (1989), pp. 357–384.

[16] J. Haslbeck and D. Wulff, *Estimating the number of clusters via a corrected clustering instability*, Computational Statistics, 35 (2020).

[17] B. Horvath and Z. Issa, *Non-parametric online market regime detection and regime clustering for multidimensional and path-dependent data structures*, SSRN Electronic Journal, (2023).

[18] A. Jacquier, P. Bilokon, and C. McIndoe, *Market regime classification with signatures*, SSRN Electronic Journal, (2021).

[19] S. P. Lloyd, *Least squares quantization in pcm*, IEEE Trans. Inf. Theory, 28 (1982), pp. 129–136.

[20] P. Nystrup, P. Kolm, and E. Lindström, *Feature selection in jump models*, Expert Systems with Applications, 184 (2021), p. 115558.

[21] P. Nystrup, P. N. Kolm, and E. Lindström, *Greedy online classification of persistent market states using realized intraday volatility features*, The Journal of Financial Data Science, 2 (2020), pp. 25–39.

[22] P. Nystrup, E. Lindström, and H. Madsen, *Learning hidden markov models with persistent states by penalizing jumps*, Expert Systems with Applications, 150 (2020), p. 113307.

[23] D. N. Politis and J. P. Romano, *The stationary bootstrap*, Journal of the American Statistical Association, 89 (1994), pp. 1303–1313.

[24] L. Rabiner, *A tutorial on hidden markov models and selected applications in speech recognition*, Proceedings of the IEEE, 77 (1989), pp. 257–286.

[25] J. Raymaekers and R. H. Zamar, *Regularized k-means through hard-thresholding*, J. Mach. Learn. Res., 23 (2022).

[26] W. Sun, J. Wang, and Y. Fang, *Regularized k-means clustering of high-dimensional data and its asymptotic consistency*, Electronic Journal of Statistics, 6 (2012), pp. 148 – 167.

[27] S. van Buuren and K. Groothuis-Oudshoorn, *mice: Multivariate imputation by chained equations in r*, Journal of Statistical Software, 45 (2011), p. 1–67.

[28] F. Wilcoxon, *Individual comparisons by ranking methods*, Biometrics Bulletin, 1 (1945), pp. 80–83.

[29] D. M. Witten and R. Tibshirani, *A framework for feature selection in clustering*, Journal of the American Statistical Association, 105 (2010), pp. 713–726. PMID: 20811510.

[30] S.-Z. Yu, *Hidden semi-markov models*, Artificial Intelligence, 174 (2010), pp. 215–243. Special Review Issue.

[31] H. Zou and T. Hastie, *Regularization and Variable Selection Via the Elastic Net*, Journal of the Royal Statistical Society Series B: Statistical Methodology, 67 (2005), pp. 301–320.