Analysis of Sibling Time Series Data: Alignment and Difference Detection

by

Jennifer Listgarten

A thesis submitted in conformity with the requirements
for the degree of Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto

Abstract

Analysis of Sibling Time Series Data: Alignment and Difference Detection

Jennifer Listgarten
Doctor of Philosophy
Graduate Department of Computer Science
University of Toronto
2007

Many practical problems over a wide range of domains require synthesizing information from time series data. Two distinct, yet related, problems in time series data are those of alignment and difference detection. These tasks may be coupled together so that a solution to one is difficult without a solution to the other.

We introduce a unified, probabilistic approach to the problems of alignment and of alignment with difference detection. This approach takes the form of a class of models called *Continuous Profile Models* for simultaneously analyzing sets of sibling time series – those which contain shared sub-structure, but which may also differ. In this type of generative model, each time series belonging to one class is generated as a noisy transformation of a single *latent trace* in the model. A latent trace can be viewed as an underlying, noiseless representation of the set of observable time series belonging to one class, and is learned from the data. If multiple classes of data exist, then one latent trace per class is learned, and these are aligned to each other during inference. The latent traces lie at the core of this class of models, and provide the basis for alignment and difference detection.

Our approach to alignment has several benefits over traditional approaches. It provides a principled method for finding parameters in the model, such as the reference template and error/distance function, rather than specifying these in an *a priori* and/or *ad hoc* manner. It simultaneously aligns all data in one go, rather than aligning them in a greedy, incremental fashion. It corrects scaling of signal intensity while performing alignment. Additionally, the probabilistic framework allows us the option of using fully Bayesian inference, if desired, so

that we may gauge uncertainty in our model parameters, integrate out model parameters, and avoid cross-validation, which can be problematic with limited data. Lastly, the CPM is the first model, to our knowledge, to tackle the simultaneous problems of alignment and difference detection.

We focus on Liquid-Chromatography-Mass Spectrometry proteomics data, for examination and demonstration of our methods, although our methods are not confined to this domain.

Acknowledgements

By far the most important acknowledgment and thanks goes to my two supervisors, Sam Roweis and Radford Neal. Each one is unsurpassed as a supervisor, except by their combination. Additionally, the machine learning group in Toronto – faculty, postdocs and students alike provided a wonderful environment in every respect. In particular I would like to thank Roland Memisevic, Rama Natarajan, David Ross, Ted Meeds, Nati Srebro, Ben Marlin and (honorary ML member due to his use of our lounge) Horst Samulowitz. Also, I would like to thank Anna Bretscher, Josh Buresh-Oppenheim and Diego Macrini for their long and lasting friendship throughout and past our studies together in Toronto, Kathryn Graham for her biological knowledge and Alberta beer, and Jack Newton for his endless supply of technical knowledge, gourmet cooking and parties. I thank my parents for their constant support. And JR for all.

I would also like to thank my other committee members, Brendan Frey and Andrew Emili, and my external examiner, Kevin Murphy for their valuable feedback, and also Ben Marlin for his slice sampling function, Tom Minka for his Matlab Lightspeed and Fastfit toolboxes, Carl Rasmussen for his Matlab minimize routine, and Giorgio Tomasi for his Matlab implementations of DTW and COW.

Part of my funding came from the Natural Sciences and Engineering Research Council of Canada and the Ontario Graduate Scholarship program.

Contents

1 Continuous Profile Models **1**

1.1 The Alignment Problem 1

1.2 Amplitude Normalization 2

1.3 Alignment of Liquid-Chromatography Experiments 3

1.4 The Difference Detection Problem 4

1.5 Contributions of this Thesis 5

1.6 Organization of this Document 6

1.7 Notations and Conventions 7

2 Related Work on Analysis of Sets of Time Series Data **8**

2.1 Dynamic Time Warping 8

2.1.1 Regularization of DTW 9

2.2 Other Approaches to Alignment 12

2.2.1 Vector Time Series Alignment 13

2.3 Multi-Class Alignment and Difference Detection 14

2.4 Supervised/Semi-Supervised Alignment 15

3 LC-MS Proteomics – Introduction and Review **17**

3.1 From DNA to RNA to Proteins 17

3.2 Mass Spectrometry for Proteomics 18

3.2.1 Mass Spectrometers 19

3.2.2 Tandem Mass Spectrometers (MS/MS) 20

3.2.3 Liquid Chromatography – Mass Spectrometry (LC-MS) 21

3.2.4 Paradigms for Information Extraction from LC-MS 22

3.3 Analysis of LC-MS Data 23

3.3.1 Quantization of *m/z* values 24

3.3.2 Alignment in time (and *m/z*) . . . 25
3.3.3 Evaluation of Processing . . . 28
3.3.4 Biomarker Discovery and Classification . . . 29

4 An EM-Based Continuous Profile Model 33
4.1 Continuous Profile Models . . . 33
4.1.1 Etymology of the Name 'CPM' . . . 35
4.2 Single-Class EM-CPM . . . 35
4.3 Extension to Multiple Classes . . . 40
4.4 Extension to Vector Time Series . . . 41
4.5 Training the CPM with EM . . . 43
4.5.1 E-Step . . . 44
4.5.2 M-Step . . . 45
4.5.3 Initialization and Setting of Hyper-parameters . . . 47
4.6 Obtaining an Alignment after Training . . . 48
4.6.1 Unrolling an Alignment . . . 48
4.6.2 Alignment Scores . . . 48
4.7 Scaling Spline Alternative . . . 50
4.8 Modeling Choices in the Design of the EM-CPM . . . 51
4.8.1 Relation to Input-Output HMMs . . . 54
4.9 Single-Class Experiments . . . 54
4.9.1 Demonstration of the Model in Action . . . 55
4.9.2 Convergence . . . 55
4.9.3 Stability with Respect to Initialization . . . 56
4.9.4 To Learn or Not To Learn the State Transition Probabilities . . . 57
4.9.5 Smoothing Parameter . . . 57
4.9.6 Scale States Verus Scaling Spline . . . 58
4.9.7 Examining the Posterior . . . 74
4.9.8 Vector Time Series Alignment . . . 74
4.9.9 Comparison to Dynamic Time Warping-Like Algorithms . . . 75
4.9.10 Demonstration on Speech Data . . . 86
4.10 Multi-Class Experiments . . . 86
4.11 Discussion . . . 89

5 A Hierarchical Bayesian Continuous Profile Model 92
5.1 Motivation . . . 92
5.2 Hierarchical Structure: Parent Traces to Child Traces . . . 93
5.3 General Set-Up and Notation Reminder . . . 94
5.4 Formal Specification of the HB-CPM . . . 95
5.5 Training the HB-CPM by MCMC . . . 100
5.5.1 Obtaining Alignments and Normalization After Training . . . 116
5.6 More General Hierarchical Bayesian CPMs . . . 117
5.7 Experiments . . . 119
5.7.1 Initialization . . . 119
5.7.2 NASA data . . . 119
5.7.3 LC-UV data . . . 120
5.8 Discussion . . . 122

6 Difference Detection in LC-MS Data for Protein Biomarker Discovery 132
6.1 Introduction . . . 132
6.2 Related Work . . . 133
6.3 The Data . . . 134
6.4 Approach to Detection . . . 135
6.4.1 The Data Matrix . . . 135
6.4.2 Smoothing of Residual 'Mis-Alignment' . . . 136
6.4.3 A Spatial Test Statistic for Difference Detection . . . 138
6.4.4 Comparison to Ground Truth . . . 139
6.4.5 Precision-Recall Curves . . . 141
6.5 Experiments and Results . . . 142
6.5.1 Effect of Number of *m/z* Bins . . . 143
6.5.2 Stability of DTW Versus EM-CPM . . . 144
6.5.3 Effect of Using Different Number of Replicates . . . 146
6.5.4 Obtaining Alignments from the HB-CPM . . . 146
6.5.5 Assessing The Difficulty of the Difference Detection Problem . . . 149
6.5.6 Predictive Modeling . . . 150
6.6 Compute Times . . . 151
6.7 Discussion and Conclusion . . . 151

7 Summary and Future Directions **157**
7.0.1 Limitations . . . 159
7.1 Future Directions . . . 160

Appendices **163**

A Probability Distributions Notation **163**

B Hidden Markov Models **164**
B.0.1 Formal Introduction to HMMs and Relevant Algorithms . . . 165
B.0.2 Forward-Backward algorithm . . . 167
B.0.3 Using the posterior of the hidden state sequence . . . 168

C The Expectation-Maximization Algorithm **172**

D M-Step Derivations for the EM-CPM **176**
D.1 Latent Trace M-Step . . . 177
D.2 HMM Emission Variance M-Step . . . 178
D.3 Time State Transition Parameter M-Step . . . 178
D.4 Scale State Transition Parameter M-Step . . . 179
D.5 Global Scaling Constant M-Step . . . 180
D.6 Scaling Spline M-Step . . . 180

E The Bayesian Paradigm and Sampling Methods **182**
E.1 The Bayesian Modeling Paradigm . . . 182
E.2 Estimation of the Posterior by Sampling . . . 185

F Identities Used for HB-CPM Inference **189**
F.1 Sherman-Morrison-Woodbury matrix inversion formula . . . 189
F.2 Gaussian mixing property . . . 189
F.3 Breaking apart a compound gaussian . . . 190
F.4 Multivariate gaussian conditionals . . . 191
F.5 Matrix version of completing the square . . . 192
F.6 Unrolling unnormalized gaussian clique potentials . . . 192

Bibliography **194**

Chapter 1

Continuous Profile Models

1.1 The Alignment Problem

Many practical problems in many domains require synthesizing information from data sampled over time or space. An example of such *time series* are speech waveforms, in which the digital representation of a vocal utterance is represented by real-valued numbers at a sequence of discrete time points. Time series data are often noisy, and in particular, the timing of salient events can be extremely variable. This variability arises because timing during collection of the data cannot be accurately controlled, or, if it can be controlled, the measured time does not correspond to the timing of the underlying processes we wish to model and understand. For example, speech recognition needs to account for the fact that different people speak at different rates. The underlying processes producing speech are the utterances of particular syllables (or phonemes), which can span more or less time, depending on the particular speaker. It is useful to find some canonical/reference time frame to which all the time series can be mapped, so as to make them directly comparable to one another. We refer to the problem of making two or more time series comparable to one another by changing their relative timing as the *alignment* problem.

It is valuable to note that the alignment problem is ill-posed. That is, there does not exist a single, best solution which could be agreed upon. If one has speech data from several speakers, say with each speaking at his/her own uniform rate, then what would be the best reference time frame to which to map all of these speech time series? One could just as well use the slowest person's time frame, or the fastest person's, or any of the ones in between. Or one could use a time frame that was never observed. One could even devise a pathological time frame in which the original data was mapped non-monotonically (*i.e.,* the relative timing of events in a given

time series would change so that the future with respect to one event becomes the past in the pathological time frame).

Are any of these reference frames better than the others? One could argue quite convincingly that the pathological time frame is not desirable because we would like a reference time frame which maps events monotonically. But beyond this criterion, can we reasonably argue for one time frame over another? One could appeal to the Occam's razor principle which states that simplicity is best – all else being equal. In the alignment setting, this principle could be translated into the desideratum that a reference time frame should be one in which the least 'complex' mappings from observed time series to the reference time is achieved, provided the quality of the alignment is not sacrificed (the best quality alignment would be one in which the underlying processes are in perfect correspondence with each other). Indeed, adherence to Occam's principle would rule out our pathological reference frame since swapping past and future is surely a complex mapping by any definition. This idea of keeping it simple enters into traditional alignment algorithms, under the guise of 'regularization', in which model complexity is held at bay.

The alignment problem is a pervasive one, spanning not only speech and music processing, but also, for example, equipment and industrial plant diagnosis/monitoring, and analysis of biological time series from microarray and liquid/gas chromatography-based laboratory (LC/GC) data (such as mass spectrometry (LC-MS) and ultraviolet diode arrays). A main contribution of this thesis is to provide a robust model and algorithm for alignment of time series data.

1.2 Amplitude Normalization

In addition to the need for correction of time, one may also need to correct for systematic differences in the amplitude of the signal at each time point. This is the problem of *normalization*. For example, in a laboratory experiment, it may be the case that an instrument is slightly miscalibrated one day, and that all measurements are thus systematically larger than on a previous day. Or it may be the case that some speaker's volume trails off at the end of a sentence, while others' remains constant, or takes on some other pattern. These are not variations in the timing of events, but in the amplitude of events at each time point. Normalization might be viewed as an easier, or less critical problem than alignment, although the two can be intertwined – that is, one may only be able to do 'optimal' alignment after normalization, and one may only be able to do 'optimal' normalization after alignment. Indeed, there is a degeneracy arising from performing both of these corrections, since one could imagine putting an observation in one

time series into corespondance with that in another either by moving it in time, or by changing its scale.

1.3 Alignment of Liquid-Chromatography Experiments

As mentioned, the problem of alignment arises in many biological or chemical experimental settings, such as experiments which measure the expression of genes at various points in the cell cycle (using microarrays). In this thesis, liquid-chromatography based experiments are the primary application. In liquid chromatography, some solution of interest (*e.g.*, serum) is passed through a chromatography column which separates parts of the solution on the basis of some chemical property (for example, hydrophobicity). At discrete time intervals, solution is collected as it exits the column, and then is analyzed by an instrument (*e.g.*, a mass spectrometer). Analysis by the instrument at each discrete time point can provide a feature vector (*e.g.*, mass/charge ratios in mass spectrometry), or scalar values (*e.g.*, UV absorbance at a single wavelength).

If a single specimen of solution is split into two parts, and each of these are then run through the same liquid chromatography column, they will not travel through the column in identical ways. Their paths are affected by chemical and physical properties of the column which may not remain identical from run to run or even within a run, as well as by ambient factors such as temperature and pressure in the laboratory. Thus data collected from LC experiments can be extremely variable in time.

Liquid-chromatography (LC) techniques are currently being developed and refined with the aim of providing a robust platform with which to detect differences in biological organisms – be they plants, animals or humans. Detected differences can reveal new fundamental biological insights, or can be applied in more clinical settings. LC-mass spectrometry technology has recently undergone explosive growth in tackling the problem of biomarker discovery – for example, detecting biological markers that can predict treatment outcome or severity of disease, thereby providing the potential for improved health care and better understanding of the mechanisms of drug and disease. In botany, LC-UV data is used to help understand the uptake and metabolism of compounds in plants by looking for differences across experimental conditions.

Many of the models and algorithms introduced in this thesis are examined in the context of LC data, although our models and algorithms are very general, and could of course be applied to any number of domains.

1.4 The Difference Detection Problem

In many time series settings, one is interested in aligning data for the purpose of comparison. Comparison could take the form of a simple yes/no response. For example, *does a particular speech utterance match a stored record* – for the purpose of, say, biometric identification. Alternatively, comparison could take on a more detailed form – *what are all of the differences between LC-MS profiles from aggressive prostate cancer serum samples and those arising from a more benign form of the disease.* The first type of comparison, *classification*, is an easier task, since it requires only that we be able to model sufficient information so as to reliably distinguish between two categories. The second type of comparison, *difference detection*, is more challenging since it requires finding all patterns which systematically differ between categories, even if these are rare, noisy or of little value for classification (*e.g.*, are redundant given other information).

Note that there are different ways in which one could define the problem of difference detection. For example, one could define it as looking for all single features which appear in Category A, but not in Category B. Alternatively, one might define it as looking for all feature combinations (*i.e.*, including sets of features) which systematically differ between categories. In other words, there may be individual features which do not differ between categories, but perhaps two such features in combination with one another do provide a systematically different pattern. Technically, we are distinguishing between first order difference detection (using single features), and higher order difference detection (in which *sets* of features are sought).[1] In this thesis, we consider only the first order difference detetecion problem, which is by far the easier problem, since looking for all combinations of features is intractable.

Difference detection manifests itself in many of the same domains as alignment (*e.g.*, biometric speech analysis, industrial plant diagnosis/monitoring, LC-based biomarker discover). When the difference detection problem arises in static data (*i.e.*, not time series data), one can appeal to traditional statistical or machine learning methods to model any systematic differences. However, when one wants to perform difference detection on time series data requiring alignment, the problem can become much more difficult. Difficulty arises if the systematic differences which exist between categories make it difficult to align the data, producing a chicken/egg scenario – to find differences between categories, one may need to first align the data, but to align the data, one may first need to know where the differences are so that they do not disturb the alignment.

[1] Analogously, classification is routinely performed using first order features, or higher order features, or both.

Previous approaches to this problem ignore this chicken-egg scenario, choosing to start with the alignment problem (while ignoring differences between time series), followed by difference detection. That is, people typically apply a *single class* alignment algorithm to the data, assuming that differences need not be accounted for, and then perform difference detection following alignment. A main contribution of this thesis is to present a unified solution to these two, simultaneous problems.

We use the term *sibling time series* to refer to sets of time series which share common substructure, but may also differ from one another. Thus difference detection (and alignment) operate on sibling time series data.

1.5 Contributions of this Thesis

In this thesis, we develop a novel, unified, probabilistic framework for alignment, normalization, and difference detection in sibling time series. In particular, we present a new class of probabilistic, generative models, *Continuous Profile Models* (CPMs), in analogy to Profile HMMs for discrete sequences, which simultaneously *align* and *normalize* sibling time series data. If more than one class of data is present, one can additionally, simultaneously, perform *difference detection*.

The core foundation of Continuous Profile Models is the concept of a *latent trace*, which is a latent variable in these models.[2] A latent trace can be viewed as a canonical, underlying representation of a set of time series data from one class, and is inferred by our algorithms during training/inference. By inferring the latent trace, one can obtain alignments and normalize the data. In the multi-class models, there is one latent trace per class, which provides the foundation for multi-class alignment and difference detection. In this scenario, not only are the latent traces from each class learned, but they are also simultaneously aligned to each other while accounting for differences between them. These class-specific latent traces are then in correspondence with one another and can then be used to reveal differences between classes.

Continuous Profile Models use the formalisms and machinery of probabilistic, generative modeling. This allows for a principled and unified approach to both alignment alone and alignment together with detection. By principled, we mean that we avoid requirements of specifying a reference time frame and a distance/error function for measuring how similar a set of

[2]Latent variables are not directly observed, but explain commonalities among things that are observed. Also, technically, in the first model we introduce, the latent trace might better be considered a parameter, since we are learning only a point-estimate of it.

observed time series are to one another in an *ad hoc* manner. Instead, we use parameters/latent variables which represent these quantities, and learn them from the data in a statistically sound way, thereby providing an algorithm tailored to the data at hand. CPMs are applicable to the alignment problem alone, and also to the problem of simultaneous alignment and difference detection. We emphasize both of these problems equally in this thesis.

Within the CPM framework,

1. We introduce a MAP (maximum *a posteriori*) -based Continuous Profile Model (EM-CPM) for alignment of (single-class) sibling time series data, and investigate properties of this model and algorithm through experimentation on LC-MS data. Training in this model is done using the EM algorithm.

2. We introduce a fully Bayesian, Hierarchical Continuous Profile Model (HB-CPM) which uses Markov Chain Monte Carlo (MCMC) for inference. This model is particularly well suited to performing simultaneous alignment and detection under a single, unified modeling paradigm. We investigate properties of this model and algorithm on some NASA solenoid valve data (providing a nice illustrative example) and on LC-UV data from a botany laboratory.

3. We apply our models to an LC-MS proteomic spike-in experiment in which known proteins are added to a base of serum, providing a realistic, yet verifiable set-up to assess different algorithms. We demonstrate the advantages of using CPMs over the more traditional Dynamic Time Warping class of algorithms for alignment. In this setting, we are also able to demonstrate that the problem of difference detection is indeed a more challenging one than classification.

1.6 Organization of this Document

The remainder of this thesis is structured as follows:

Chapter 2: We discuss previous work pertinent to the problem of analyzing sets of related time series data.

Chapter 3: We provide an introduction and brief review of the field of proteomics as it pertains to our work in Chapter 6. Readers not interested in this field can skip this chapter without much loss.

Chapter 4: We introduce the class of models called *Continuous Profile Models* (CPM) – a class of probabilistic generative models for alignment and normalization of sets of related time series data. We then present one specific instance of CPMs– the EM-based CPM (EM-CPM), as well as a method to train the EM-CPM. We explore use of the EM-CPM, mainly in the context of a liquid-chromatography mass spectrometry (LC-MS) data set, but also with a speech data set. Lastly, we briefly explore a multi-class version of the EM-CPM as applied to a NASA solenoid valve data set.

Chapter 5: We introduce a fully Bayesian instance of the class of CPM models, the *Hierarchical Bayesian Continuous Profile Model* (HB-CPM), as well as an associated (MCMC) algorithm for inference. We then explore use of this model on two data sets – a three-class liquid-chromatography-ultraviolet-diode array data from a study of the plant *Arabidopsis thaliana* and a two-class solenoid valve current data set.

Chapter 6: We present a simple technique for discovering differences between two classes of samples, which is used after data alignment. We apply this technique to LC-MS serum proteomic data without use of tandem mass spectrometry, gels, or labeling and test our technique on a controlled and realistic (spike-in, serum biomarker discovery) experiment which is therefore verifiable. This set-up allows us to assess different approaches to the alignment problem, by comparing precision-recall curves built from knowledge of the spike-in ground truth. We are thus able to contrast the performance of Dynamic Time Warping, the EM-CPM and the HB-CPM, demonstrating some advantages of CPMs.

Chapter 7: We wrap-up with a discussion of what we have accomplished and of future directions.

1.7 Notations and Conventions

Forms of the standard distributions not appearing in the text can be found in the first Appendix. Most background material is also found in various Appendixes rather than in the main manuscript so that readers already familiar with these topics can easily access the novel contributions of this thesis. Additionally, some technical details and derivations required for our models are relegated to various Appendixes in order to make for a clearer read.

Chapter 2

Related Work on Analysis of Sets of Time Series Data

Synopsis: In this chapter we discuss previous work on analyzing sets of related time series data. The main discussion focuses on alignment of times series, with a brief discussion of a few related topics at the end. Discussion of these topics in the context of Liquid Chromatography-Mass Spectrometry proteomics can be found in Chapter 3.

2.1 Dynamic Time Warping

We'll start our discussion with Dynamic Time Warping (DTW) [70], the classical algorithm for alignment and the foundation for many other alignment algorithms. In Chapter 4.2 we compare and contrast the underlying concepts of our Continuous Profile Models to those of DTW-like algorithms, while in Chapter 6, we compare these algorithms' performance on a specific task (difference detection in LC-MS data).

Dynamic Time Warping is a dynamic programming (DP)[1] based alignment algorithm which originated in the speech recognition community as a robust distance measure between two time series. DTW works on pairs of time series, aligning the time frame of one time series to another. It can also work by aligning each time series to some template which is chosen *a priori* – possibly one of the observed time series in a set. The reference template and also the distance function, which provides a value for how close two measurements are, must be specified ahead

[1]Dynamic programming refers to a particular algorithmic paradigm in which the naive computations to solve a problem are so numerous as to be intractable, but in which one can find structure in the computations so as to be able to re-use portions of them to make the overall computations much more efficient.

of time, usually by trial and error or previous experience on similar data sets.

The 'vanilla' version of DTW (that is, the simplest possible version) can be easily understood: Suppose we have two scalar time series, $x_1, \ldots, x_N$, and $y_1, \ldots, y_M$, with lengths respectively N and M (one of these time series might be some template specified ahead of time, or they might both be observed time series). The output of DTW tells us, for every position, in each sequence, what position in the other sequence it should be matched to in the aligned reference frame. That is, DTW, after completion, will specify a vector of pairs, $(q_1, p_1), \ldots, (q_T, p_T)$, of length T, whose i^{th} entry, (q_i, p_i), specifies that in the aligned reference frame, x_{q_i} and y_{p_i} occur at the same 'corrected' time step. Depending on the warping path chosen by the algorithm, T can be bigger or smaller, and different extensions to the vanilla algorithm can alter the preferences for the length of this path (by altering the regularization – explained in the next section). Before the outputs, $(q_1, p_1), \ldots, (q_T, p_T)$, can be computed, one must first do a dynamic programming pass over the data, which incrementally and recursively builds up the 'cost' of taking every possible warping path, $(q_1, p_1), \ldots, (q_T, p_T)$. After this DP cost matrix is computed, one can then do a 'backward' pass to find the optimal path, $(q'_1, p'_1), \ldots, (q'_T, p'_T)$, which is the final answer. A schematic view of the DTW search space is shown in Figure 4.17, based on a similar figure in [37]. Details of the algorithm are best understood by looking at the pseudo-code, shown in Algorithm 1 (adapted from Dan Ellis' on-line Matlab code[2]). The time and space complexity for alignment of two time series is proportional to NM, the product of the lengths of each time series being aligned.

A point that is easy to miss, but becomes obvious when one tries to use DTW to align sets of time series data, is that in aligning an entire *set of time series*, one must align each time series in turn. Thus, each application of DTW, in general, provides a warping path of a different length from the others. That is, the reference time frame uncovered by each application of DTW *to the same reference template* is different, and one must devise ways to make them comparable (such as picking one recovered DTW warping path as a reference time frame, and interpolating all others in time to this one). Thus there is a subtle but critical difference between a reference time frame, and a reference template when using DTW.

2.1.1 Regularization of DTW

The term *regularization* in data modeling refers to the concept of restricting the complexity of the model. In most cases, regularization will cause a poorer 'match' of the observed data to the model, and so may seem, at first glance, counter-intuitive. However, the underlying principle at work is that overly-complex models can overfit the data, and thus not model the signal of

[2] http://labrosa.ee.columbia.edu/matlab/dtw/

Algorithm 1 'Vanilla' Dynamic Time Warping – Input: x,y, distance function, Output: q,p

$N \leftarrow$ length(x), $M \leftarrow$ length(y)
$C \leftarrow$ matrix[1..N+1][1..M+1] %%*(empty DP cost matrix)*
$P \leftarrow$ matrix[1..N][1..M] %%*(empty matrix of pointers to do trace-back)*
%% *Initialize cost matrix*
for $i = 1$ to N **do**
  for $j = 1$ to M **do**
    $C[i][j] =$ Infinity
  end for
end for
$C[1][1] = 0$
%% *Now fill it in using DP, keeping track of which choices we make for later traceback.*
for $i = 1$ to N **do**
  for $j = 1$ to M **do**
    cost = distance($x[i], y[j]$)
    $C[i+1][j+1] \leftarrow$ minimum($C[i][j]$ + cost, $C[i][j+1]$ + cost, $C[i+1][j]$ + cost)
    if $C[i+1][j+1] == C[i][j]$+ cost **then**
      $P[i][j] \leftarrow 1$
    else if $C[i+1][j+1] == C[i][j+1]$+ cost **then**
      $P[i][j] \leftarrow 2$
    else
      $P[i][j] \leftarrow 3$
    end if
  end for
end for
%% *Backwards traceback to optain warping path stored in vectors p,q*
$i \leftarrow N$, $j \leftarrow M$
$q[1] \leftarrow i$, $p[1] \leftarrow j$
index $\leftarrow 2$
while $i > 1$ and $j > 1$ **do**
  backPointer $\leftarrow P[i][j]$
  if backPointer $== 1$ **then**
    $i \leftarrow i - 1$, $j \leftarrow j - 1$
  else if backPointer $== 2$ **then**
    $i \leftarrow i - 1$
  else
    $j \leftarrow j - 1$
  end if
  q[index] $\leftarrow i$, p[index] $\leftarrow j$
  index $\leftarrow$ index+1;
end while

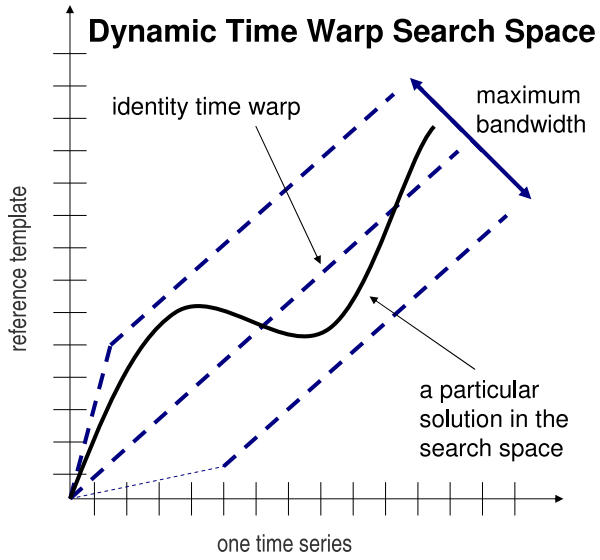


Figure 2.1: A schematic view of the Dynamic Time Warping search space.

interest as best as possible. By restricting the model complexity, one can try to make the model ignore the noise, and model only the true signal of interest. A variety of regularization techniques exist, such as early stopping during training, shrinkage, parameter tying, and what some consider to be the most principled approach, that of full Bayesian modeling, in which model complexity is restricted indirectly by allowing as complex a model as desired, but by requiring one to integrate over all possible parameter settings in the model (*i.e.*, using the posterior of these parameter settings) – thereby blurring out very unlikely ones if not warranted by the data. More details on this full Bayesian approach are provided in Appendix E.

DTW can be regularized in a variety of ways [81], which are best understood by visualizing the DTW cost matrix, and the possible paths through it. Suppose we have two time series, each of the same length. A 'neutral' warp would be a straight line with slope 1 through the cost matrix. If the first time series were systematically 'faster', then the optimal warping path slope would be greater than 1, and if it were systematically slower, then the optimal warping path slope would be less than 1. If the relative 'speeds' of each time series varied over time, then locally, the slope would change as appropriate. Three common ways to regularize DTW are to 1) specify a minimum/maximum slope in the warping path, 2) specify a maximum bandwidth for the warping path (*i.e.*, a maximum amount that the path can depart from the neutral path), and 3) encourage warping paths to be of shorter rather than of longer lengths. Note that in the

naive implementation shown in Table 1, the cost is proportional to the length of the warping path, T, and hence the third type of regularization is inherent in vanilla DTW, although some people alter the algorithm to remove this bias.

Nielsen *et al.* [61] developed an algorithm based closely on DTW, which they call Correlated Optimized Warping (COW). One can view COW as a regularized version of DTW. In DTW, every observed time point from a given time series is represented in the DP cost matrix, and each point can be moved independently from the others (except when regularization prevents this). In COW, one instead chooses a set of core time points (nodes) the size of which is smaller than the total number of time points; one is only allowed to warp these core nodes, and must use linear interpolation to warp the portion of the time series lying between them.

Prince and Marcotte recently developed a DTW-based alignment algorithm, 'Ordered Bijective Interpolated Warping' (OBI-Warp) [67] whose main contribution is in how they deal with the output of DTW across an entire set of time series. As mentioned earlier, the output of DTW from a set of time series data does not provide a single alignment across all samples. Prince and Marcotte develop a set of rules that dictate which points from the warping path should be included in a subsequent interpolation scheme which provides a smooth warping function.

DTW and variations of DTW are widely used across a broad range of practical domains. In particular, they play a large role in chromatography-based laboratory experiments which rely almost solely upon this class of algorithms for alignment.

2.2 Other Approaches to Alignment

In the statistics community, Ramsay *et al.* introduce a curve clustering model, in which a time warping function, $h(t)$, for each time series is learned by way of learning its relative curvature, $w = \frac{D^2 h}{Dh}$ [33]. The relative curvature is parameterized with order one B-spline coefficients, which are learned by optimizing a penalized, squared error criterion, which measures the distance between an aligned time series and a learned template. The penalty is on w^2 which assigns low penalty to flat warping functions (*i.e.* linear), and high penalty to wiggly warping functions (*i.e.*, highly non-linear warps). This approach comes from viewing the alignment problems as a second-order linear stochastic differential equation. The 'regression template' is learned by way of a Procrustes fitting criterion which is conceptually similar in nature to Viterbi learning [65] – an approximation to the Expectation-Maximization learning algorithm (Appendix C provides an introduction to the EM algorithm) in which the posterior over the hidden states is collapsed to the MAP state sequence. Although this iterative template fitting might do well in practice, it has no guarantee of convergence.

Gaffney and Smyth introduced a probabilistic model which jointly clusters and aligns curves [28]. The alignment part of the model can be thought of as a B-spline regression model in which the dependent variable, $\vec{y}_i$, is the amplitude of the measured signal, and the input, $\vec{x}_i$, is the vector of time points, (i indexes over the input vectors being aligned). The alignment comes into the regression by way of four parameters, a_i, b_i, two controlling the scaling and offset in time, and another two, c_i, d_i, controlling the scaling and offset in amplitude space: $\vec{y}_i = c_i[a_i\vec{x}_i - v_i]\vec{\beta} + d_i + \epsilon_i$, where $\vec{\beta}$ are the regression coefficients and ϵ_i are zero-mean Gaussian noise with covariance σ_k^2, and $[a_i\vec{x}_i - v_i]$ represents the regression matrix (spline basis matrix) evaluated at at the transformed time. Thus their method takes into account changes in the scale of the signal while aligning. However, it only allows for a linear time warp with an offset. EM (Expectation-Maximization) is used to maximize the log probability of the observed data; the posterior over the hidden variables is approximated by way of i) variational Gaussian approximation, and ii) Monte Carlo integration (each on different parts of the factorized posterior). Ziv Bar-Joseph *et al* use a somewhat similar approach to cluster and align microarray time series data in [9].

A brief description is given of an alignment algorithm in [74] which uses a 'guide tree' to progressively warp together experiments. The guide tree can be built for example by UPGMA (bottom-up, average-link, hierarchical clustering), using the distances between samples. The authors do not describe in detail how samples are warped together, and indicate that the use of the guide tree removes the need for a template to which samples should be warped. Presumably the algorithm starts by aligning the two 'closest' samples, which then together can be thought of as a single pseudo-sample, to which the next closest samples is aligned, *etc*. While this may reduce the need for a template, it is not clear that starting with the two closest time series, as measured in the original space, is a good approach. One could instead use an iterative approach in which one chooses to measure distance in the original space, performs the merging with the guide tree to obtain a set of aligned data, and then measures the distance in this aligned space, and then uses this refined distance to re-merge with a new guide tree. However, such an iterative approach would have no guarantee of convergence (or guarantee of maximizing any particular objective function).

2.2.1 Vector Time Series Alignment

The algorithms we have described so far are applicable only to scalar time series data, or, if applied to vector time series, can only be done so in a restricted way – time warping across dimensions must be identical. Radulovic *et al.* [68], in the context of LC-MS experiments which are inherently vector time series data, develop an alignment algorithm which does not

have this restriction. If one thinks of a single, vector time series data as a matrix, with time along one axis, and dimensionality along the other, than Radulovic *et al.*'s algorithm works by splitting this matrix up into equal sized blocks (where one block spans a set of continugous time points, and a set of contiguous dimensions (*m/z* values), and then warping each of these blocks, separately, linearly, in time, but in a manner so that the boundaries of each block are constrained to meet. An 'overlap' objective function is used which measures the point-wise overlap of the binarized data matrix from two experiments. Presumably one could easily extend this portion of the algorithm to non-binary data. Search through the space of transformations is performed by Accelerated Random Search, a Monte Carlo method for maximization. After the optimal set of linear transformations is found, a final corrective 'wobble' is applied (the wobble is applied to each 'peak' – a set of 'on' binarized features which form a maximally connected graph, where graph edges exist for features which are next to each other in the data matrix). While linear transformations on a coarse grid of blocks may seem restrictive, this may be compensated for by the *post hoc* 'wobbles'. However, it is critical to note that this refinement step is only possible on binarized data, a rather large restriction.

2.3 Multi-Class Alignment and Difference Detection

Although alignment has been extensively studied, to our knowledge, no existing model attempts to perform alignment of time series belonging to different categories/classes in a principled way (*i.e.*, incorporating *a priori* knowledge that classes differ into the alignment algorithm).

We define *single class alignment* algorithms as those which assume that all sequences come from the same class and attempt to align them all together, explaining any differences as random noise. In many real-world situations, however, we have time series from multiple classes (categories) and our prior belief is that there is both substantial shared structure between the class distributions and, simultaneously, systematic (although often rare) differences between them. While in some circumstances, when differences are small and infrequent, single class alignment can be applied to multi-class data, it is much more desirable to have a model which performs true *multi-class alignment* in a principled way, allowing for more refined and accurate modeling of the data.

A problem related to multi-class alignment is the time series difference detection problem – in which we seek to align and compare sets of related time series in order to characterize how they differ from one another. In order to find differences, we need to align, but in order to align, we may require knowledge of the differences; hence the tasks become coupled.

As mentioned earlier, Gaffney *et al* [28] clustered and aligned time series data from differ-

ent classes, jointly, and Ziv Bar-Joseph *et al* use a somewhat similar approach to cluster and align microarray time series data in [9]. However, these approaches do not attempt to put time series from different classes in to correspondence with one another – only those time series within a class are aligned to one another. Similarly, Ramsay *et al*'s algorithm [33] accounts for systematic changes in the range and domain of time series in a way that registers curves with the same fundamental shape. However, their method does not allow known fundamental differences between shapes to be taken into account. The anomaly detection (AD) literature deals with related, yet distinct problems. For example, Chan *et al* [17] build a model of one class of time series data (they use the same NASA valve data as we do in this thesis), and then match test data, possibly belonging to another class (*e.g.* 'abnormal' shuttle valve data) to this model to obtain an anomaly score. Emphasis in the AD community is on detecting abnormal events relative to a normal baseline, in an on-line manner, rather than comparing and contrasting two different classes in one go. The problem of 'elastic curve matching' is addressed in [42], where a target time series that best matches a query series is found, by mapping the best matching subsequence paradigm to the problem of a cheapest path in a DAG (directed acyclic graph).

2.4 Supervised/Semi-Supervised Alignment

All of the alignment methods we have discussed so far are unsupervised in the sense that no knowledge of the underlying processes of interest are used during alignment. However, some algorithms use such knowledge to guide their alignment, and we refer to such approaches as *supervised alignment*. One example is an algorithm by Fischer, Grossman *et al.* [27] in which side-information (in the form of peptide sequence information for LC-MS data) is used to guide alignment. The side information allows them to identify underlying processes which are common to all time series being observed (even if they are known not to all contain the same set of underlying processes). They then use ridge regression (*i.e.*, regularization by way of a squared error term on the regression weights) to estimate a time warping function that optimizes a robust error function. The regression input is the value of the signal at a particular time point, while the regression output is the amount of warping that should be applied to that point. Optimization is performed by an iteratively re-weighted least squares scheme. Regression is first performed using only the correspondences of underlying processes found from the side information, and then the alignment is refined using ideas motivated by the co-training algorithm which allows supervised methods to incorporate unlabeled data. This set-up provides a nice framework for semi-supervised alignment.

Although supervised methods have the potential to provide improved results over purely unsupervised methods, the requirement of (correctly) detecting some of the underlying pro-

cesses before doing alignment can be limiting in many domains. Our goal in this thesis is to analyze time series data in an unsupervised fashion,[3] without reliance on measurement of underlying processes.

[3]Later in this thesis, when we perform multi-class alignment, we do use the class labels, but this supervisory information is different in nature than that discussed here.

Chapter 3

LC-MS Proteomics – Introduction and Review

Synopsis: In this chapter we provide a brief introduction and selected review of the field of LC-MS proteomics so that readers unfamiliar with this topic may more fully understand our work in this area, as well as the context and motivation for our work. Part of this chapter contains sections of our paper [46], and we refer the reader there for broader discussion and more details which are beyond the scope of this thesis. Detailed discussion of time series alignment is found in the previous chapter, with the discussion here focused on issues specific to LC-MS data.

3.1 From DNA to RNA to Proteins

With the sequencing of the human genome largely complete and publicly available, emphasis in molecular biology is shifting away from DNA sequencing and related problems, to the understanding of biological events that are downstream effects of the DNA sequences.While DNA is often called the blue-print of life, there is, under most circumstances, no simple mapping between a person's DNA sequence and that person's health, intelligence, metabolism, *etc.* In between the DNA and these phenotypes lies a complicated biological machinery, in which proteins play a crucial role.

The central dogma in molecular biology stipulates that the flow of information is unidirectional, flowing from DNA to mRNA to protein. Specifically, DNA is transcribed into mRNA (by way of pre-messenger RNA), and mRNA is translated into protein, which in turn directly affects many events in the body, from very local events like cell death and signal transduction, to more global events such as neurological signaling and muscle contraction. In this cascade of processes, errors occur during both translation and transcription. Furthermore, the

existence of splice variants (isoforms), created when pre-messenger RNAs are cut and pasted together in different ways, means that several transcripts can be made from the same DNA sequence. Additionally, after mRNA has been translated into protein, the protein can undergo post-translational modifications which alter its structure or chemical properties, and hence its functionality. Lastly, the level of transcriptional and translational activity of any given gene and protein is affected by the presence or absence of other proteins; thus a complex network of dependencies and interactions exists [4]. As one goes further downstream, from DNA, to mRNA, to proteins, the complexity of the system increases greatly. While it is thought that there are around 20,000 genes coded for by human DNA, it is estimated that a large portion of these have splice variants (often tens to hundreds of variants each), and these in turn may each have several hundred possible post-translational modifications [41, 44]. Since proteins work together, the final complexity of a biological system is enormous, making the search for interesting and relevant patterns a challenging task.

Proteomics seeks to answer the following questions: 1) what are all of the possible proteins, and isoforms, ever expressed in a given organism, 2) which post-translational modifications occur in each of these proteins, 3) how do the answers to 1-2 differ in different parts of the organism and under different conditions, 4) how do the answers to 1-3 help us to better understand general biological processes and disease development, diagnosis and treatment? [39, 83, 69].

The research community is tackling these questions using a wide range of technologies, including protein tissue arrays, two-dimensional polyacrylamide gel electrophoresis (2D-gel), nuclear magnetic resonance, and various kinds of mass spectrometry. In the next section, we give a brief overview of Mass Spectrometry technology for proteomics.

3.2 Mass Spectrometry for Proteomics

Mass spectrometry technology is currently being developed to advance the field of proteomics [3, 39, 40, 83, 62, 22, 84, 63, 1]. A mass spectrometer takes a sample as input, for example, human blood serum, and produces a measure of the abundance of molecules that have particular *m/z* (mass/charge) ratios. In proteomics the molecules in question are peptides, or sometimes whole proteins. From the pattern of measured abundance values one can hope to infer which proteins were present in the sample and in what quantity. For protein mixtures that are very complex, such as blood serum, a separation step (*e.g.*, chromatography) is often used to physically separate parts of the sample before injection into the mass spectrometer. This separation means that a less complex mixture is fed into the mass spectrometer at any one time. The field of mass spectrometry is huge and highly technical, and we do not here attempt to provide even brief coverage of this area. We instead cursorily mention the main classes of technological

platforms available so that the reader is aware of them. For a more in-depth discussion of these and their relative merits in the context of proteomics, see [3, 2].

3.2.1 Mass Spectrometers

A mass spectrometer consists of 1) an ionization source which converts the sample molecules into gas phase ions, 2) a mass analyzer that separates ions on the basis of their *m/z* values, and is also able to measure the *m/z* values, 3) a detector that registers the relative abundance of ions at each *m/z* value. Different mass spectrometry platforms differ in the way each of these components is set up [3, 2].

For the ion source, two main technologies are commonly used: electrospray ionization (ESI) and matrix-assisted laser desorption/ionization (MALDI) [3]. Because ESI generates ions directly from solution, it is easily coupled to a liquid-chromatography or capillary electrophoresis system, which separates the protein mixture out over time. This makes it particularly well suited to examining complex protein mixtures. MALDI technology works best when restricted to simpler protein mixtures, for which it provides high ion yields of the intact analyte, and allows for the measurement of compounds with high accuracy and sensitivity [2].

For the mass analyzer, there are four basic types used in proteomic studies: i) ion trap, ii) time-of-flight (TOF), iii) quadrupole, and iv) Fourier transform cyclotron (FT-MS). The mass analyzer plays a critical role in mass spectrometry, and its quality is assessed by its sensitivity, mass resolution and accuracy, and dynamic range [3].

For the detector, three main technologies are used, i) electron multiplier and ii) Photomultiplier Conversion Dynode (Scintillation Counting), and iii) a Faraday cup. Each of these essentially amplifies the incident ion signal into an output current which can then be measured directly.

Additionally, a platform commonly used in proteomic studies is SELDI, (surface-enhanced laser desorbtion/ionization), a refinement of MALDI, in which samples are pre-treated with various proteomic chips (consisting of chromatography surfaces or biological surfaces), performing protein extractions based on a variety of interactions such as hydrophobicity and metal binding The proteins in the sample selectively bind to the surface while non-specifically bound proteins are washed away [22, 62]. An important difference between SELDI and liquid chromatography mass spectrometry (LC-MS) lies in the fact that with SELDI, much of the sample is discarded (any part of the sample which is not selectively bound to the matrix).

Two important types of systematic patterns relevant to proteomics occur in MS data – the first relates to the charge on the ions, and the second to isotopes. Most molecules can be ionized in a number of different ways, resulting in a net positive or negative charge on the ion, which

can vary. For a peptide with 9 amino acids, there may be one, two or three positive charges, and a peptide of length 30,000 may have more than twenty ions. Multiply-charged molecules are most common with ESI. When a molecule which is ionized in multiple ways goes through the mass spectrometer, its signal is spread out along the *m/z* axis in what is called a *charge state envelope*. If the sample being analyzed is simple enough, it is possible to use *a priori* knowledge to figure out which *m/z* peaks are related to each other (called deconvolution). This deconvolution relies on a second type of pattern in MS spectra, isotope shoulders. Isotope shoulders arise because because the carbon atoms contained in peptides can occur with either the standard six neutrons (^{12}C) or (around 1%) with seven neutrons (^{13}C). The presence of this heavier isotope can be seen as peaks of lower abundance shifted to the right of the mono-isotopic peak (all ^{12}C).[1] The carbon isotope dominate other isotopes because roughly fifty percent of the mass of a peptide is carbon [51]. From the spacing of *m/z* peaks in the isotope shoulder, it is possible to deduce the charge state of an ion. Knowing the charge state, it is possible to deconvolve spectra consisting of multiply charged ions [16].

Very high-resolution mass spectrometers can provide information about very subtle structure in the spectra; however, at present, such high cost instruments are accessible mainly to those in industry. The 'workhorse' instruments that are prevalent today in proteomics laboratories are mass spectrometers of much lower resolution, for which subtleties of the spectral patterns are not easily discernable. It is data from this latter type of instrument which we use in this thesis.

3.2.2 Tandem Mass Spectrometers (MS/MS)

One of the more powerful and commonly used methods of obtaining rich information from mass spectrometry technology is to use a Tandem-Mass Spectrometer (MS/MS or Tandem MS). With MS/MS, two mass spectrometers are used in conjunction with one another in order to elucidate the detailed make-up of some of the analyte ions. The two mass spectrometers are connected in series by a chamber called a collision cell. Suppose one is interested in looking in more detail at the make-up of ions with *m/z*=1000, then all of the analyte ions are passed through the first mass analyzer, and only those with *m/z*=1000 are allowed to pass through to the second mass analyzer. On their way there, they pass through a collision cell which fragments the selected ions into smaller pieces, which are then analyzed by the second mass spectrometer. Typically, the computer controlling the Tandem Mass Spectrometer automatically chooses precursor ions (*i.e.*, which *m/z* values to retain) by selecting those with high abundance, and

[1] For molecules heavier than about 4kD (kilo Daltons, where 1 dalton = 1 atomic mass unit = $\frac{1}{12}$ the mass of a carbon atom), the mono-isotopic peak is no longer the dominant one [16], as can be understood by looking at the binomial distribution which models the number of ^{13}C in a molecule.

which have not recently been analyzed by the second mass spectrometer.

Because the peptide fragmentation patterns are fairly well-understood, it is possible to make use of Tandem Mass Spectrometry to sequence peptides. For a fragment to be detected by the Mass Spectrometer it must be charged, and the most common charged ion fragments are the so-called 'b' and 'y' ions, which are the result of the peptide being cleaved at the CO-NH bond along its backbone (with the former retaining the charge on the N-terminus fragment, and the latter retaining the charge on the C-terminal fragment). When peptides are fragmented in this way, overlapping sequence fragments are produced. By calculating the mass difference between fragments differing by one amino acid, the full amino acid sequence can be teased out (provided that the ion types can be inferred). An alternative to this *de novo* sequencing is to instead use a database of already known, or theoretically generated peptide spectra with which to match the unknown peptide's spectral patterns, or to use a combination of *de novo* and database searching [64, 8].

3.2.3 Liquid Chromatography – Mass Spectrometry (LC-MS)

In LC-MS, a liquid chromatography step is added prior to injection of the sample into the mass spectrometer. This liquid chromatography step spreads out the parts of the sample solution over time on the basis of some property of the molecules, such as hydrophobicity. For example, in RV-HPLC (Reverse Phase High Performance Liquid Chromatography), a tubular column is packed with some material made up of hydrophobic molecules. The sample is then loaded onto the packed column, and sample molecules bind to the packed material with different affinities, depending on each molecule's hydrophobic properties. Next, solution is pumped through the column. Initially, the solution is mostly water, and is increasingly made to contain more and more organic solvent. As this organic solvent gradient is applied, molecules that are more weakly bound to the packed material come off first, and progressively, those that are more and more strongly bound come off [31]. As liquid comes off the column, it accumulates into a droplet, which is then vaporized and ionized as it is introduced into the mass spectrometer. Before being loaded on to the column, the protein sample is typically broken down into peptides by use of an enzyme called trypsin, so that peptides, not full proteins are ionized and measured by the mass spectrometer. Figures 4.3 and 4.6 show some LC-MS data. The first figure displays just the TICs,[2] while the second displays a zoom in of the data along both the LC and *m/z*domains.

The result of using LC-MS is that at each time point a less complex mixture (less complex

[2]TIC is an abbreviation for Total Ion Count and is a measure of the abundance of all ions at one time point, rather than separating out the abundances over different *m/z* values.

than if we had injected the whole sample at once into the MS machine) is fed into the mass spectrometer and a set of MS measurements at each time point is obtained. This LC step is helpful for two reasons: 1) peptides with the same *m/z* values are less likely to analyzed by the mass spectrometer at the same time, thus reducing ambiguity in the MS signal, and 2) the fewer the number of ions being simultaneously analyzed by the mass spectrometer at a time, the less ion suppression comes in to play. Ion suppression refers to one ion's signal suppressing the signal of another ion. With the presence of ion suppression artifacts, the abundance of any one species as detected by the MS instrument can depend on the abundance of other entirely different species; clearly this is undesirable effect. It is thought that ion suppression occurs when less volatile compounds change the efficiency of droplet formation just prior to ionization which in turn affects the amount of charged ion produced and ultimately the amount of analyte detected [7, 22], however it is not fully understood nor is it well-characterized.

While accuracy in measurements of *m/z* values in MS can typically be made to be quite good, the LC step results in a tremendous amount of variation in the time axis. For example, small variations in ambient pressure, ambient temperature, the organic gradient, *etc.* can cause the time axis to be variously shifted, compressed and expanded, in complex, non-linear ways. Furthermore, sampling of time is not necessarily constant so that even if the column were ideal, measurements would still come from different time points, as a result of the nature of the coupling to the MS instrument. As solution continually elutes off the column, it is continuously sprayed into the mass spectrometer, which allows a roughly constant amount of ions to pass through to the detector or collision cell. Which portions of this spray end up in the MS depends on the characteristics of the ions involved. Furthermore, when the first or second MS is busy, solution continues to be sprayed into the MS instrument, but some of it is inevitably wasted. When a tandem mass machine is analyzing ions only in the first detector it is said to be in *full scan mode*. A typical way to run a MS/MS instrument is in alternating mode where a full scan is performed (all *m/z* values), followed by use of the second detector for selective ion analysis. Thus the amount wasted depends on which mode the machine is being operated in. A precise understanding of how the solution is being sampled in time is not available. Thus to make use of the benefits of LC-MS, one needs to have effective ways of dealing with the inherent variability in the time axis. That is, alignment along the time/LC axis is a critical step in practical use of LC-MS.

3.2.4 Paradigms for Information Extraction from LC-MS

Tandem MS has proven to be extremely useful. However, running MS/MS is a time consuming process, taking around an order of magnitude (*e.g.*, 24 hours instead of 1 hour), times longer

than running just a single MS. Secondly, much time is wasted re-sequencing the same peptides over and over again, across many samples. Lastly, it is impossible to sequence every peptide in a sample solution because of the way the MS/MS works, selecting only one group of ions at any given time.

In the context of LC-MS/MS, one can imagine avoiding some of the less desirable properties associated with MS/MS just mentioned. For example, one can imagine building up a database of peptides previously sequenced by LC-MS/MS. Then, given a new solution to analyze, one could imagine running LC-MS, obtaining an abundance measurement for peaks centered at pairs of (*time, m/z*), and then matching these pairs into the database to determine what peptide might be likely to have those values. In this way, the same peptides need not be sequenced and re-sequenced, the time to run an experiment would be far less, and more peptides could be sequenced [39].

However, such an approach, if possible, would be complicated by a number of factors: 1) the liquid chromatography time axis would need to be corrected so that times in different experiments correspond to one another, and to the database measurements, 2) the same molecule in different samples may not behave the same way in the chromatography column due to differences in the make-up of the remainder of the sample, 3) there may be overlap of peptides in the (*time, M/Z*) space, and 4) the LC-MS system has a huge amount of noise and variability that is not well-characterized or accounted for. If, however, such an approach were possible, alignment of the LC axis would be a critical problem.

In the aforementioned approach, peaks are extracted and then identified, and only after these steps are performed does one pose queries related to the particular experiment at hand. An alternative approach to using LC-MS data would be to use a signal processing approach. That is, suppose one were interested in finding biomarkers for some disease, that is, finding some pattern in the LC-MS data which can differentiate healthy from diseased persons. Rather than starting with peak detection and peptide identification, one could simply treat the data as a matrix of signals, then make use of the many well-studied methods in signal-processing, statistics and machine learning to tease out interesting and relevant patterns from the data.

3.3 Analysis of LC-MS Data

Information extraction from LC-MS signal involves a sequence of operations, each typically treated in a largely independent manner from the others:

1. Quantization of *m/z* values.

2. Signal filtering and background subtraction.

3. Amplitude normalization.

4. Data transformations/error models.

5. Peak detection and quantification.

6. Alignment in time (and *m/z*).

7. Classification algorithms and biomarker discovery.

We now briefly review some of the issues related to Items 1, 3, 6 and 7 – those which pertain to Chapter 6 of this thesis. Our discussion of time alignment is restricted mainly to the field of LC-MS, with a more general discussion available in Chapter 2. A discussion of all steps are addressed in our paper [46].

Because many of the methods applied to non-chromatographic MS data (*e.g.*, MALDI and SELDI) can be carried over to LC-MS (which is really just a time series of MS spectra), and because more studies have been published with non-chromatographic data, we frequently report on the MS literature as well.

3.3.1 Quantization of *m/z* values

The data obtained from an LC-MS experiment, in its most raw form, consists of something like the following: a table of values, with the following columns: i) Scan Number, ii) Time off LC column, iii) *m/z* value, and iv) abundance. There is a one-to-one mapping between the scan numbers and the times, thus the scan number is simply an enumeration of the ordered time points.

So as to more easily manipulate the data, the LC-MS data can be converted into a data matrix with columns representing *m/z* values and rows representing time. This involves binning the *m/z* values, since retaining all possible values would lead to a huge and sparsely populated data matrix. The time values can normally be left unchanged since they are not too numerous, and since typically many *m/z* values correspond to a given time point. Entries in the matrix represent the relative abundance values at each combination of time and *m/z* bin, which we sometimes refer to in tuple format: (time, *m/z*).

An issue to consider here is how to bin the *m/z* values. For example, one could choose evenly-spaced bins in *m/z* space, or in $\log$ *m/z* space. In [80] peak-widths (ignoring the time axis) are observed to be reasonably constant in log *m/z* space. The width of the bin should be chosen to be large enough to keep the size of the matrix tractable, and not too sparse, but small enough that the *m/z* bin values remain informative (*i.e.*, not all collapsed on to each other); this trade-off will depend on the technology being used. It is not clear how to choose an optimal

width, nor how sensitive further calculations are to the bin width. In [68, 48], *m/z* values are rounded to the nearest integer while in [6, 84] evenly-spaced bins of width 0.2 *Th*are used. In [24], the bins are chosen to have a width which is a function of their machine's mass accuracy and resolution, resulting in bins size that increases linearly with *m/z* values.

3.3.2 Alignment in time (and *m/z*)

As mentioned in Section 3.2.3, the time/LC axis (*i.e.*, the time off the column, or scan header time) in LC-MS experiments is highly variable. The underlying mechanisms for this variability are not well characterized, but correction of it is necessary in order to use the data for most purposes. In some cases there may also be a bit of drift on the *m/z* axis of the experiment, though this seems to be far less of a problem than the variability in time. Algorithms applicable in one of these domains are in general applicable in the other as well. For LC-MS data, one could align only the time axes, or both the *m/z* and time axes independently, or simultaneously. Furthermore, one may wish to align a scalar time series (*e.g.*, the TICs), or align a vector time series (*e.g.*, the vector of all of the *m/z* values at each time point), or allow even more general schemes.

Alignment algorithms typically involve either i) maximizing some objective function over a parametric set of transformations (typically linear), or ii) non-parametric alignment by way of dynamic programming, or iii) some combination of these (*e.g.* piecewise linear transformations). Some of the main differences among algorithms are i) whether alignment is performed before or after peak detection (*i.e.*, on the raw signal, or on the peaks), ii) whether or not the amplitude of the signal is used in the alignment, and iii) whether or not changes in scale are corrected for at the same time as the alignment (*i.e.*, allowing for interplay between these two types of corrections). Most algorithms require a template to which all time series are aligned, to be specified *a priori*. Typically this is chosen to be one of the time series one wants to align. A poor choice in template might result in poor alignments and it may therefore be generally desirable to avoid this.

As mentioned in the previous chapter, Nielsen *et al* developed the COW (Correlated Optimized Warping) ([61]) algorithm for chromatographic data. This algorithm can be applied to scalar or vector time series, simply by defining the distance function (correlation or covariance) appropriately. Use of more than one component (*e.g.*, several *m/z* rather than one *m/z*) from the data vectors is shown to produce more stable alignments with respect to varying the free parameters, such as the maximum amount of warp allowed [61]. This is not surprising since use of more vector components provides more information, and we corroborate this in Chapter 6. In [61], they visually show on artificially constructed examples that their algorithm

is robust to varying numbers, heights and widths of peaks in the chromatogram and is superior to a global linear warp. An interesting evaluation is provided in [15], where PCA on the BPCs (Base Peak Chromatogram[3]), and PARAFAC (Parallel Factor Analysis - a generalization of PCA to three-way data) is performed. They show that the amount of variance explained by the top two principal components was 70% before alignment, and 98% after alignment. Similarly, with a seven-component PARAFAC model, the variance explained went from 60% to 97%. Thus the alignment is reducing some of the major sources of variation between the samples.

Again, as mentioned in the previous chapter, Radulovic *et al.* [68] develop an algorithm which allows different *m/z* blocks to have different time warps. However the algorithm is restricted to work on binary data.

Randolf *et al.* use scale-specific peaks extracted from a coarse scale in their multi-scale wavelet decomposition to align MALDI data along the *m/z* axis. Dominant peaks (those above some threshold) are used to compute a single optimal shift of all peaks in one sample, and thus the alignment is not very flexible. It is not clear whether or not features at different scales should be aligned differently.

Idborg *et al.* do not explicitly align their data, rather they compare detected components from their curve resolution procedure by way of cross-correlation (with only the seven most dominant peaks), allowing one component to be shifted relative to another to account for constant time shifts in one experiment relative to another. Components which correlate above some threshold are said to be the same component.

Hierarchical clustering is used in quite a novel way in [80] to perform alignment of mass spectrometry data (where instead of aligning in time, one is aligning in '*m/z*' space – see the next chapter for details). After finding peaks (that is, they find regions of *m/z* which are turned 'on', and other that are 'off' – those regions which are 'on' are said to be peaks, and tend to be small, local features) in their scalar 'time series', they use complete linkage hierarchical clustering to align peaks across experiments. Input to the algorithm is a list of all peaks (with associated amplitudes); distance between peaks is measured by Euclidean distance in $\log$ *m/z* space. After clustering is completed, the dendogram is cut off at a particular height, and all peaks belonging to a cluster are said to be the same peak. A "consensus" peak is extracted by taking the mean *m/z* value of each cluster. This approach is thus restricted to time series which can be sensibly broken down into 'peak' and 'non-peak' regions, and cannot be applied to continuous-valued time series data.

Overall, we see that in in LC-MS experiments, people concentrate on correcting the time axis, leaving the *m/z* axis alone. However, in SELDI and MALDI experiments, corrections

[3]This is the one-dimensional time series vector obtained by taking the maximum abundance value in each *m/z* slice (or UV slice for DAD).

are performed on the *m/z* axis, so it may be desirable to perform both of these corrections in LC-MS experiments.

An issue to consider is whether to align each *m/z* slice individually, or together, or something in between (*i.e.*, in a smoothly varying way). As one moves from a global alignment perspective to a more local one, regularization will play a larger role as less data will be available for each free parameter. Additionally, the more *m/z* slices used, the more computationally intensive will be a given algorithm. The optimal trade off will be determined by the informativeness of the LC-MS signal, and the type of misalignments present in the data. One could use the TIC, the BCP, individual *m/z* slices. One could also select the k slices with the highest ion count, or highest sum of second derivatives along *m/z*, or some dynamic binning of the *m/z* such that the ion count is evenly distributed among them, or one could use some dimensionality reduction technique on the *m/z* space, such as PCA, *etc*.

A related, although separate issue with LC-MS data arises when the mass spectrometer is run in alternating scan mode (that is, it is alternating with a second mass spectrometer to perform tandem mass spectrometry). In this context, sample coming off the chromatography column is lost during use of the second mass spectrometer, and thus sample "drop-out" occurs – that is, replicate biological samples are subject not only to variation arising from the chemistry of the column, but also to variation resulting from the stochastic sampling of the sample coming off of the column.

Amplitude normalization

Not only are MS signals corrupted by general background/baseline noise, they also frequently undergo systematic changes in their abundance measurements. That is, the amplitudes measured in one replicate may be systematically larger than in another. In such cases we need to normalize the amplitudes to make the experiments comparable. The simplest normalization would be to multiply all abundance values in one experiment by some constant factor, but in general, it may be necessary to perform more detailed corrections.

Normalization of MS data can be performed either by coercing *m/z* abundance values to be comparable across experiments, or by coercing the abundance values assigned to peaks to be comparable. Note that in general, we wish to normalize not only replicates, but also experimental data whose biological origin is not identical, such as data from cancer patients and healthy people.

Many of the normalization techniques used in MS are highly similar to those used in the normalization of microarray data. The underlying assumption behind the normalization techniques is that the overall MS abundance of either all features (peaks or time-*m/z* pairs) , or subset(s) of them, should be equal across different experiments. Given this assumption, one

can then determine the ratio of the overall feature abundance between two experiments for the chosen set of features, and then use this ratio as a multiplicative correction factor for one of the experiments. One can arbitrarily choose one experiment in a set of many experiments as a reference to which all others are normalized.

In the case where all features are simultaneously used to enforce this constraint, it is a global normalization, and in the case where only a subset of features are used at a time (often different subsets for different parts of the data), it is a local normalization. Locality can be defined by say similarity in *m/z* values, time (scan header) values, or abundance level. For example, in the case of a abundance-local normalization, those peaks or features with similar abundance values within one MS experiment would be scaled in a similar way, while those peaks or features with quite different abundance values could be scaled in a very different way, within the same MS experiment. If the mean of all features is made to agree across all experiments, we call it a global mean normalization, which is used for example in [74] on SELDI data. By plotting the point-wise log ratio of matched features for each *m/z* value, versus the *m/z* value, Sauve *et al*. show that visually there is no trend along the *m/z* axis [74] and thus that a *m/z*-dependent normalization is not needed for their data. Similarly, they show that no abundance trends exist and that the normalization need not be abundance-dependent.

With microarray data, one sometimes restricts the genes used for normalization to so-called 'housekeeping' genes which are thought to remain constant across the experimental conditions. Such an idea is applied to LC-MS data in [84], where a constant intensity ratio between pairs of experiments is computed using what are thought to be constant-abundance analytes, who's peaks must be found and quantified in the LC-MS signal. They then calculate a single normalization constant with respect to an arbitrarily chosen reference. Use of all peaks is also found to be adequate. (Baggerly *et al*. also consider using 'housekeeping' peaks to normalize, but stated that they were unable to find stable peaks across experiments [36].)

While many papers used a global abundance normalization, in the case of LC-MS data, it may be desirable to have normalization more local in time, since the chromatography step can produce fluctuations in signal.

3.3.3 Evaluation of Processing

The coefficient of variation (CV) of peaks/features is sometimes reported in the literature as a measure of the quality of the experiment and associated post-processing. For example, [84, 88, 68] respectively report CVs of less than 30%, 50%-60% (before pre-processing), and 25%. While these numbers have meaning on their own, and can be used to help assess the quality of the processing (for example by comparing the CV before alignment/normalization and after),

they are not absolute measures in any sense, and certainly much less so than in the case of microarray data, where the number of features is determined *a priori* by the chip. With MS experiments, feature detection is part of the game and this makes the interpretation of CVs drastically different. Consider the case where one decides to use very drastic pre-processing steps, for example where data is heavily smoothed and then binarized, allowing only strong peak signals to remain. In such an case, the CV would only be a reflection of those strong peaks that survived the pre-processing, and could have much lower CVs than CVs resulting from another technique which may have preserved far more information. Ultimately, what we are interested in is performance on some final goal, such as classification. If performance on this final goal is measurable, as in the case of classification, then it can provide one type of assessment of the processing steps, as for example is done in our paper [47] for microarray data to assess normalization schemes, and in Chapter 6 of this this thesis for the difference detection problem in LC-MS. Clearly, a feedback loop with the laboratory is also desirable.

3.3.4 Biomarker Discovery and Classification

The ultimate goals of most LC-MS based proteomics experiments are:

1. Classification (*e.g.*, can we distinguish chemotherapy-resistant tumours from other tumours?)

2. Low-level biomarker discovery: which points in the (time, *m/z*) space are responsible for the pattern differences between classes (or which peaks)?

3. High-level biomarker discovery: which peptides (or peaks) correspond to the biomarkers found in low-level biomarker discovery, and also which proteins correspond to these peptides?

4. What is the full list of proteins expressed in this sample, and at what level are they expressed?[4]

We have listed these tasks from most to least tractable. Building classification algorithms to tease apart class-dependent data is often feasible. However, determining the complete set of features (*i.e.*, peaks or peptides) responsible for the pattern differences, in a statistically sound way can be difficult (we will show this to be the case in Chapter 6). Frequently in machine learning methods, a step called *feature selection* is used to automatically select relevant

[4]This problem of finding all proteins is different in nature than the other problems listed; even if we knew the entire list of proteins contained in various samples, this would not directly lead to a solution to the other problems listed.

features. Sometimes this is performed independently from learning the classification model (feature selection is then called filtering), or interactively with learning (wrapper), or sometimes can fall out of the classification algorithm itself (*e.g.,* in the case of Decision Trees). Feature selection is in and of itself an extremely difficult problem and remains an open research question. Even if one could find the optimal set of features for a particular learning algorithm (optimal in the sense of providing the best generalization), this would not then mean that all statistically discriminative features had been found (or even only such features), because the optimal set of features for a particular learning algorithm is heavily dependent on the mathematical framework of the learning algorithm on hand. For example, if one were using a simple linear classifier, and found the optimal set of features for this classifier, any features which acted only in concert to provide discriminative power would not appear in this set (*e.g.,* imagine two proteins, which, if we knew the expression of only one of them, gave no information, but when both were either very highly expressed, or neither was expressed, would provide perfect discrimination). These artifacts, where features used by particular classifiers do not necessarily correspond to the set of features which most correlate with the classes, are present with every classifier which can be trained in finite time, and can offer explanation to the biologists as to why different analyses of the same data shed light on different features, without there being an error or problem in the analysis, as for example suggested in [22].

The statisticians take a different view of finding discriminative features. Typically they will perform some test deemed appropriate to the data on hand (*e.g.,* a t-test), that provides a value reflecting how much a feature discriminates between two classes. Then the distribution of these test values, under the assumption that no features are discriminative, is modeled. From this *null distribution*, they can then calculate the probability, for each of their features, that the feature would have the test value it has, or a more extreme test value, given that the feature is not discriminative. In the classical setting, these null distributions were theoretically developed, but currently, with computing power cheaply available, these distributions are often simulated, for example using permutations tests (Monte Carlo Permutation Procedure). An example of a permutation test would be to remove the labels from the samples (*e.g.,* cancer and healthy), and then to randomly assign labels to the samples, so that only spurious correlation exists between samples and lables. Then one can compute the value of the test function for every feature. By repeating this procedure many, many times, one can obtain an approximation to the null distribution for this test, for data that has the same kind of of structure as that on hand. These permutation tests are nice because they require no particular assumptions about the data, which most classical statistical tests do. Typically permutation tests are performed for tests which operate on one feature at at time. Since one is typically checking every feature in the data set, one could be performing thousands of significance tests. In such cases, the

false positive rate (FPR), which the p-value thresholds seeks to control, will be extremely large for commonly used p-values thresholds (*e.g.*, $p \leq 0.01$), and therefore some sort of *multiple testing* correction needs to be applied. Many methods have been developed for this type of correction, for example, the conservative Bonferroni correction which controls the probability that one or more false positives will be found over all tests (*i.e.*, it controls the family-wise error rate). However, in the context of most biological experiments, it is not in fact the FPR which we seek to control, rather it is the False Discovery Rate (FDR), as pointed out by Storey and Tibshirani [78]. While the FPR tells us how many of our features which are truly null, are expected to be called significantly different by our test, the FDR tells us how many of of the features that our test calls significant, are expected to in fact not be. Thus from a practical standpoint, the FDR tells us how many of our 'leads' (*e.g.*, a list of peaks that are different between two classes) we might expect to be false. Storey and Tibshirani develop the q-value, which can be thought of in a similar way to a p-value, except that it i) controls the FDR rather than the FPR, and ii) it automatically accounts for multiple-testing. If one chooses a q-value of 0.05, then one can expect that 5% of all features called significant will in fact not be. The q-value seems a more appropriate statistical measure than the p-value for biomarker discover in the present context.

We note also that permutation tests are sometimes used to assess the validity of a classification algorithm. In this usage, labels are randomly permuted (the number of samples in each class is maintained). Then the classifier is trained using the random labels, and its predictive power assessed. Such a method is used in [36, 85] and provides an easier way of reporting p-values seen in the biomedical literature (as opposed to the machine learning literature which typically does not use p-values).

Classification

One of the classic MS prediction papers is the ovarian cancer study of Petricoin *et al.* [62] (SELDI) in which a genetic algorithm is used to find a good set of predictive *m/z* values. The general idea was to evaluate sets of 20 *m/z* values (there were a total of $15,200$) as features by their ability to discriminate between cancer and healthy. Good sets are progressively combined until perfect discriminating power is obtained on the training data. Discriminating power is defined to be an ability to form two clusters with correct class membership, on the training data. Some sort of clustering is performed with a Euclidean metric. The final classifier is obtained after the optimal set of features is selected by the genetic algorithm, by clustering these features. New samples are classified by their proximity to each of the class clusters. It appears that the way they build progeny from the 'fitter' subsets is to only form new groups of *m/z* values, not to create completely new features. If this is indeed the case, then the final

classifier is simply a linear classifier in a subset of the original input space. Thus it might make more sense to use a linear discriminant classifier with some sort of regularization built in, such a shrinkage method like Nearest Shrunken Centroid [79, 80] After training they used a hold out set with which they achieved 100% specificity and 95% sensitivity. This paper sparked a debate due to the fact that the biomarkers discovered were not identified (biologically), with those arguing for the need to identify before publishing, and those in opposition to this [20, 19].

Similar ability to obtain near perfect classification has since been widely observed in the MS literature [62, 36, 77, 87, 73, 69]. But reproducibility of results has been limited at best. There is speculation that proper experimental methodology may partially be to blame, and that the patterns being observed are not in fact related to disease state, but instead to factors such as sample collection, preparation and storage [77, 1, 21, 22, 80]. Thus, it would seem that the most immediate goal in this area is not so much to develop new classification and biomarker discovery algorithms, but to instead be sure that the data sets are indeed representative of the biological problems being tackled.

Chapter 4

An EM-Based Continuous Profile Model

Synopsis: In this chapter we introduce a class of models called *Continuous Profile Models* (CPM), which is a class of probabilistic generative models for alignment and normalization of sets of related time series data. We then present one specific instance of CPMs– the EM-based CPM (EM-CPM), as well as a method to train the EM-CPM. We explore use of the EM-CPM, mainly in the context of a liquid-chromatography mass spectrometry (LC-MS) data set, but also with a speech data set. Lastly, we briefly explore a multi-class version of the EM-CPM as applied to a NASA solenoid valve data set.

4.1 Continuous Profile Models

When observing multiple time series generated by a noisy, stochastic process (such as liquid chromatography experiments), large systematic sources of variability are often present. For example, within a set of nominally replicate time series, the time axes can be variously shifted, compressed and expanded, in complex, non-linear ways. Additionally, in some circumstances, the scale of the measured data can vary systematically from one replicate to the next, and even within a given replicate.

We propose a class of models called Continuous Profile Models (CPM) for simultaneously analyzing a set of such time series. In this type of of generative model, first introduced by us in [48], each time series belonging to one class is generated as a noisy transformation of a single *latent trace*. If multiple classes of data exist, then there is one latent trace per class, and data is generated from the appropriate class-specific latent trace. A latent trace is an underlying, noiseless representation of the set of replicated, observable time series. Output time series are generated from this model by moving through a sequence of hidden states in a Markovian manner and emitting an observable value at each step, as in an HMM (Hidden Markov Model). Each hidden state corresponds to a particular location in the latent trace, and the emitted value

from the state depends on the value of the latent trace at that position. Addtionally, to account for changes in the amplitude of the signals across and within replicates, the latent time states are augmented by a set of scale states, which control how the emission signal will be scaled relative to the value of the latent trace. Thus the full hidden state space is the product state space of 'hidden times' and 'hidden scales'. Lastly, each time series can be scaled by a constant global scaling factor. Figure 4.1 illustrates a single-class CPM in action, while Figure 4.2 depicts the generative process of a single-class CPM. Together, these give an excellent overview and introduction to CPMs.

During training, the latent trace[s] is[are] learned, as well as the transition probabilities controlling the Markovian evolution of the scale and time states, the overall noise level of the observed data and the global scale factor. After training, the latent trace learned by the model represents a higher resolution 'fusion' of the experimental replicates.

The concepts at the core of a CPM are:

1. A latent trace representing the underlying noiseless representation of the set of time series.

2. A learned mapping between *observed time* (time that observed experimental observations are made in) and *latent time* (time as measured in a CPM – it can be thought of as a reference time). Mapping every observed time series into latent time by way of the CPM provides an alignment of all of the observed time series. This mapping is accomplished by way of an HMM.

3. Ability to simultaneous scale the signal (sometimes referred to as normalization) while performing alignment in time. This can be achieved in a number of different ways, as discussed in this chapter.

Note that a degeneracy can arise between the hidden scale states and the hidden time states. For any given observed time point in one time series, the CPM might choose to 'correct' it by moving it in latent time, or by changing its scale state. That is, the CPM may choose to change the way that point maps to latent time relative to other time points in that same time series, or it may choose to change the magnitude of the amplitude at that time point (or it may do both of these). However, with sufficient model regularization, or sufficient observed time series to train with, this degeneracy would disapear.

These concepts will become clearer when the full EM-CPM is formally introduced in Section 4.2. Since HMMs and some affiliated algorithms are heavily relied upon by CPMs, we refer the reader to Appendix B for a HMM reminder or quick introduction.

4.1.1 Etymology of the Name 'CPM'

The CPM is similar in spirit to Profile HMMs which have been used with great success for discrete, multiple sequence alignment, modeling of protein families and their conserved structures, gene finding [23], among others. Profile HMM are HMMs augmented by constrained-transition 'Delete' and 'Insert' states, with the former emitting no observations. They also have strict left-to-right transition rules (*i.e.*, one can only move forward along a sequence). Thus inference in a Profile-HMM is linear in the number of states (rather than quadratic in a general HMM). Multiple sequences are provided to the Profile HMM during training and a summary of their shared statistical properties is contained in the resulting model. The development of Profile HMMs has provided a robust, statistical framework for reasoning about sets of related discrete sequence data. One can think of the CPM as a continuous data, conditional analogue to the Profile HMM.

4.2 Single-Class EM-CPM

In the EM-based CPM (EM-CPM), first introduced in [48], all unobservables in the model, other than the HMM states, are treated as parameters – that is, we learn a point-estimate of each of these unobservables, using a penalized maximum likelihood framework. The HMM states are hidden variables that we fully integrate out during learning. This set-up constrasts with that in Chapter 5 where we introduce a fully Bayesian CPM in which a full posterior over all parameters but top-level hyper-parameters can be estimated.

We note that the model name, EM-CPM, is a slight abuse of terminology, since EM is an algorithm used to find a maximum likelihood solution, but the name EM-CPM refers to a particular model (which we happen to train with EM). The maximum likelihood parameters of the EM-CPM could of course be found by alternate means, such as direct gradient ascent.

We now formally introduce the EM-CPM model. We will do so using scalar time series data for clarity of exposition. Our extension to vector time series data (there are theoretically many such extensions) is a straight-forward one which we mention afterward. Suppose we have a set of K time series,

$$\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_{N^k}^k),$$

where the temporal sampling rate within each $\boldsymbol{x}^k$ need not be uniform, nor must it be the same across the different $\boldsymbol{x}^k$. For notational convenience, we henceforth assume $N^k = N$ for all k (*i.e.*, that all time series have the same number of observed points), but this is not a requirement of the model and it would be straight-forward to accomodate different N^k.

The CPM is set up as follows: We assume that there is a latent trace,

$$\boldsymbol{z} = (z_1, z_2, ..., z_M),$$

a canonical representation of the set of noisy, input, replicate time series. Any given observed time series in the set is modeled as a non-uniformly subsampled version of the latent trace to which local scale transformations have been applied. Ideally, M would be infinite, or at least very large relative to N so that any experimental data could be mapped precisely to the correct underlying trace point. Aside from the computational impracticalities this would pose, great care to avoid overfitting would have to be taken. Thus in practice, we have used $M = (2+\epsilon)N$ (double the resolution, plus some slack on each end) in our experiments and found this to be sufficient with $\epsilon < 0.2$. Because the resolution of the latent trace is higher than that of the observed time series, experimental time can be made effectively to speed up or slow down by advancing along the latent trace in larger or smaller jumps.

The subsampling and local scaling used during the generation of each observed time series are determined by a sequence of hidden state variables. Let a state sequence through the CPM hidden states be $\boldsymbol{\pi}$. Then each state in the state sequence maps to a time state/scale state pair:

$$\pi_i \rightarrow \{\tau_i, \phi_i\}.$$

Time states belong to the integer set $(1..M)$; scale states belong to an ordered set $(1..Q)$. (In our experiments we have always used Q=7, evenly spaced scales in logarithmic space). States, π_i, and observation values, x_i^k, are related by the emission probability distribution:

$$p(x_i^k|\pi_i, \boldsymbol{z}, \xi^k, u^k) \equiv \mathcal{N}(x_i^k|z_{\tau_i}\phi_i u^k, \xi^k), \tag{4.1}$$

where ξ^k is the noise level (variance) of the observed data. The u^k are real-valued scale parameters, one per observed time series, that correct for any overall (global) scale difference between time series k and the latent trace.

Tape Player Analogy It can be helpful to draw an analogy between generating an observed time series in the CPM and playing an audio cassette tape in a special type of tape player which in addition to the regular volume knob, also has a 'speed' knob. When playing a tape in this machine, one can keep one hand on each knob, controlling both the speed at which the audio is played, and also the volume at which it is played. Similarly, one can think of the generative process of a CPM as having a 'speed' knob and an 'amplitude' knob. When generating a single observed time series in a CPM, one can change either knob, or both at once. When the speed is

increased (by moving more quickly through latent time with the hidden time states), less of the underlying latent trace will be represented in the observed time series, and when the speed is decreased (by moving more slowly through latent time), more of that portion of the latent will be represented in the observed time series. The observed signal emitted at a particular latent time point is scaled by a factor proportional to the setting on the amplitude knob.

To fully specify our model we also need to define the state transition probabilities. We define the transitions between time states and between scale states separately, so that the joint state transitions factor as follows

$$T^k_{\pi_{i-1},\pi_i} \equiv p^k(\pi_i|\pi_{i-1}) = p(\phi_i|\phi_{i-1})p^k(\tau_i|\tau_{i-1}).$$

The constraint that time must move forward, cannot stand still, and that it can jump ahead no more than J time states is enforced. (In our experiments we used $J = 3$.) As well, we only allow scale state transitions between neighbouring scale states so that the local scale cannot jump arbitrarily. With these factored transitions the number of possible states that can be transitioned to at any one time is only $3J$. Thus the number of legal transitions is kept small and hence the transition matrix is very sparse. Additionally, we find these constraints to work well in practice. Each observed time series has its own time transition probability distribution to account for experiment-specific patterns. Both the time and scale transition probability distributions are given by multinomials:

$$p^k(\tau_i = a|\tau_{i-1} = b) = \begin{cases} \kappa^k_1, & \text{if } a - b = 1 \\ \kappa^k_2, & \text{if } a - b = 2 \\ \vdots & \\ \kappa^k_J, & \text{if } a - b = J \\ 0, & \text{otherwise} \end{cases}$$

$$p(\phi_i = a|\phi_{i-1} = b) = \begin{cases} s_0, & \text{if } D(a,b) = 0 \\ s_1, & \text{if } D(a,b) = 1 \\ s_1, & \text{if } D(a,b) = -1 \\ 0, & \text{otherwise} \end{cases}$$

where $D(a, b) = 1$ means that a is one scale state larger than b, and $D(a, b) = -1$ means that a is one scale state smaller than b, and $D(a, b) = 0$ means that $a = b$. The distributions are

constrained by: $\sum_{i=1}^{J} \kappa_i^k = 1$ and $2s_1 + s_0 = 1$.

J determines the maximum allowable instantaneous speedup of one portion of a time series relative to another portion, within the same series or across different series. If one portion of a time series uses only neighbouring time states as it moves through a hidden state sequence, while another uses the maximum allowable number, J, then the latter will be moving at a rate J faster than the former. However, the length of time for which any series can move so rapidly is constrained by the length of the latent trace; thus the maximum overall ratio in speeds achievable by the model between any two entire time series is given by $\min(J, \frac{M}{N})$.

We have now set up the core elements of the EB-CPM, but there remain two relatively minor additions which help to regularize the model. The first is a smoothness prior on the latent trace, and the second, a prior on the HMM state transition probabilities which ensures that all non-zero state-transition probabilities remain non-zero throughout learning.

To encourage the latent trace to be smooth, we penalize the likelihood with the following prior smoothing penalty

$$p(\boldsymbol{z}) = \prod_{j=1}^{M-1} \mathcal{N}(z_{j+1}|z_j, \tfrac{1}{2\lambda}) \tag{4.2}$$

$$\propto \prod_{j=1}^{M-1} \exp\left(-\tfrac{1}{2}\tfrac{(z_{j+1}-z_j)^2}{\frac{1}{2\lambda}}\right) \tag{4.3}$$

$$= \exp\left(-\lambda \textstyle\sum_{j=1}^{M-1}(z_{j+1}-z_j)^2\right), \tag{4.4}$$

Clearly, this penalty directly discourages the latent trace from varying rapidly point-to-point. The higher the value of λ, the stronger is the discouragement. It will not be possible to directly learn an appropriate value of λ during training, rather we will have to appeal to cross-validation (in practice we use a hold out set because of the high computational burden) to find a good setting for λ. Although we initially used the penalty in Equation 4.2, we subsequently modified it slightly, as follows, and it is this form which we use in the final model,

$$p(\boldsymbol{z}) \propto \exp\left(-\lambda\bar{u}\sum_{j=1}^{M-1}(z_{j+1}-z_j)^2\right), \quad \text{where } \bar{u} \equiv \tfrac{\sum_k {u_k}^2}{K}. \tag{4.5}$$

We have simply added a factor $\bar{u}$[1] to the penalty in Equation 4.2, where $\bar{u}$ is the mean global scaling factor across all states. The motivation for adding this term is that the model, without any smoothness penalty, has a degeneracy: the latent trace, $\mathbf{z}$ has the freedom to be of any arbitrary scale because whatever scale it takes on, if this scale is far from the scale of the observed

[1]That we use $\bar{u} \equiv \frac{\sum_k {u_k}^2}{K}$ instead of say $\bar{u} \equiv (\frac{\sum_k u_k}{K})^2$ or the geometric mean has to do with maintaining analytical tractability in an earlier, related model, although this issues does not apply here since we have already lost analytical tractability in the u^k M-step updates.

data, the model will compensate by adjusting $\{u^k\}$ to offset it. Now given this degeneracy, when we add the smoothing penalty in Equation 4.2, the degeneracy disappears, and there is an immediate pressure to make the latent trace of very small (and equivalently the scale factors, u^k very large). Thus to counteract this effect, we add the factor $\bar{u}$ which, with the smoothing penalty, has the net effect of removing the degeneracy, and encouraging the scale of the latent trace to match the scale of the observed data. And in fact, to further emphasize that we want the scale of the latent trace to match that of the observed data, we also add a log-normal prior for the u^k,

$$p(\log u_k) = \mathcal{N}(\log u_k | 0, w),$$

where w is the variance (not to be confused with the class labels ω^k). Alternatives to these breakers of degeneracy would have been to constrain the latent trace values, or the global scaling constants, to sum to some constant, $\sum_j z_j = \mathcal{C}$, or $\sum_j u^k = \mathcal{C}'$.

Lastly, we add Dirichlet priors for the scale and time transition probabilities to ensure that all non-zero transition probabilities remain non-zero.

$$p(\boldsymbol{\kappa^k}) = \mathcal{D}(\boldsymbol{\kappa^k}|\boldsymbol{\eta}) \propto \prod_{v=1}^{J} (\kappa_v^k)^{\eta_v - 1}$$

$$p(\boldsymbol{s}) = \mathcal{D}(\boldsymbol{s}|\boldsymbol{\eta'}) \propto \prod_{v=1}^{2} (s_v)^{\eta'_v - 1}$$

where v indexes the dimensions of each multinomial. η_v and η'_v can be thought of as pseudo-count data. That is, one can imagine that we, *a priori*, have observed some number of each type of transition, and that we will always add these to our counts during training. This interpretation becomes more obvious in Appendix C when we show the training updates for the state transition parameters.

Likelihood Function for Single-Class, Scalar Time Series

We write the log likelihood of K observed time series, $\boldsymbol{x}^k$ as $\mathcal{L}^p \equiv \mathcal{L} + \mathcal{P}$. $\mathcal{L}$ is the likelihood term arising from the (conditional) HMM model, and can be efficiently computed using the Forward-Backward algorithm (see Appendix B). It is composed of the emission and state transition terms. $\mathcal{P}$ is the log prior (or penalty term), regularizing various aspects of the model.

The two components of the log likelihood are

$$\mathcal{L} \equiv \sum_{k=1}^{K} \log \sum_{\boldsymbol{\pi} \to \{\tau_i, \phi_i\}} p(\pi_1) \left(\prod_{i=1}^{N} \mathcal{N}(x_i^k | z_{\tau_i} \phi_i u^k, \xi) \right) \left(\prod_{i=2}^{N} T^k_{\pi_{i-1}, \pi_i} \right) \tag{4.6}$$

$$\mathcal{P} \equiv -\lambda \bar{u} \sum_{j=1}^{M-1} (z_{j+1} - z_j)^2 + \sum_{k=1}^{K} \log \mathcal{D}(\kappa_v^k | \{\eta_v^k\}) + \log \mathcal{D}(s_v | \{\eta_v'\}) + \sum_{k=1}^{K} \log \mathcal{N}(\log u_k | 0, w), \tag{4.7}$$

where $p(\pi_1)$ are priors over the initial states. Note that a hidden state path in which the latent trace is "used up" before all points in an observed time series have been "emitted", has zero probability since there are no time states that can be transitioned to from the last latent time state.

Having fully specified the model, we then use EM to estimate the MAP parameter settings for our model, as discussed in Section 4.5. For convenience, a summary of all the equations is provided on page 43.

4.3 Extension to Multiple Classes

In aligning observed time series from different, yet related classes (such when looking for biomarkers in LC data), we need to have a different latent trace for each class (*class-specific latent traces*). We assume that the data across classes are very similar, but systematically differing at a few times points. We also assume that class labels, denoted here, ω^k ($\omega^k \in [1..C]$), are available to us, as would be the case in a biomarker discovery paradigm. We discuss alternatives to such a supervised scenario in Chapter 7. To represent our knowledge the class-specific traces are similar to one another, we will add yet another penalty term to the log likelihood, one that penalizes differences between class-specific latent traces. Letting $\boldsymbol{z}^c$ be the class-specific latent trace for class c, the pentaly term in the log likelihood for the latent traces now consists of our earlier smoothing penalty, as well as an inter-class penalty,

$$\log p(\{\boldsymbol{z}^c\}) \propto -\lambda \bar{u} \sum_{c=1}^{C} \sum_{j=1}^{M-1} (z_{j+1}^c - z_j^c)^2 - \nu \sum_{c=1}^{C} \sum_{c'<c} \sum_{j=1}^{M} \log \left(1 + \frac{(z_j^c - z_j^{c'})^2}{\varsigma} \right) \tag{4.8}$$

$$= \sum_{c=1}^{C} \log F(\boldsymbol{z}^c) + \sum_{c=1}^{C} \sum_{c'<c} \log F'(\boldsymbol{z}^c, \boldsymbol{z}^{c'}). \tag{4.9}$$

This new second term in the penalty on the log likelihood can be thought of as approximately representing a product of t-distributions with 1 degree of freedom, (also called a Cauchy dis-

tribution) at each latent time, in each latent trace, centered on the value of another class's latent trace at the same latent time. We use a t-distribution-like penalty here, rather than the Gaussian-like penalty used for the trace smoothness penalty since we expect that occasionally, different classes will have substantial differences from one another, and thus a distribution with heavier tails than a Gaussian is suitable, such as a t-distribution. As with the smoothing penalty parameter λ, we will again have to appeal to a hold out set to find a good setting for ν. And if we wish to learn both λ and ν, then we will have to evaluate the hold out set over a grid of joint values for these two parameters. This may seem inconvenient, and indeed is not an ideal situation. While using a hold out set over one parameter is not much of a stumbling block, the more parameters we need to find by way of a hold out set (and which depend upon each other's settings), the larger a space we need to explore, just as when sampling a probability distribution with many dimensions. Presumably one could devise a more intelligent method than a simple grid search, but we do not explore that issue here. Of course, if one uses the model on the same kind of data over and over, then one thorough search of parameter space would reduce the need to do so again on similar, future data sets. The fully Bayesian model presented in Chapter 5 removes the need for hold out sets, and addresses the multi-class alignment in what we consider to be a nicer way. We therefore do not pursue the multi-class EM-CPM in too much detail.

4.4 Extension to Vector Time Series

We can easily extend the CPM to handle vector time series, in the simplest possible way. That is, we force all compoments of a vector at a particular time to move together and we ignore correlations between components.

Let the k^{th} vector time series be $\boldsymbol{\mathcal{X}}^k = (\boldsymbol{x}_1^k, \boldsymbol{x}_2^k, ..., \boldsymbol{x}_N^k)$, where $\boldsymbol{x}_i^k$ denotes a vector, entries of $\boldsymbol{\mathcal{X}}^k$ are written $\mathcal{X}_{id}^k$, and where i runs from 1 to N (length of the time series), d runs from 1 to D (the dimensionality of the vector at each time point), and k runs from 1 to K (the number of experimental observed time series). This change to multivariate time series manifests itself in a multivariate HMM emission probabilty distribution with a diagonal covariance matrix, $\boldsymbol{\Xi}^k = \boldsymbol{I}\boldsymbol{\xi}^k = \boldsymbol{I}\left[\xi_1^k, \xi_2^k, \ldots, \xi_D^k\right]$, where $\boldsymbol{I}$ is the identity matrix, and ξ_d^k is the HMM emission variance for the d^{th} dimension of the k^{th} time series. Additionally, the latent traces must also be changed to reflect the same number of dimensions present in the data. Thus we let $\boldsymbol{\mathcal{Z}}^c = (\boldsymbol{z}_1^c, \boldsymbol{z}_2^c, ..., \boldsymbol{z}_N^c)$ be a multi-variate, class-specific latent trace, with entries $\mathcal{Z}_{id}^c$, where i runs from 1 to M (the number of hidden time states), d runs from 1 to D (the dimensionality of the vector at each time point), and c runs from 1 to C (the number of classes). The HMM emission

probability for one a data of vector at one time point is thus given by

$$p(\boldsymbol{x}_i^k|\pi_i, \boldsymbol{\mathcal{Z}}^{\omega^k}, \Xi^k, u^k, \omega^k) \equiv \mathcal{N}(\boldsymbol{x}_i^k|\boldsymbol{z}_{\tau_i}^{\omega^k}\phi_i u^k, \Xi^k) \tag{4.10}$$

$$= \prod_{d=1}^{D} \mathcal{N}(\mathcal{X}_{i,d}^k|\mathcal{Z}_{\tau_i,d}^{\omega^k}\phi_i u^k, \xi_d^k), \tag{4.11}$$

where u^k and ϕ_i^k remain scalars (that is, the same scaling is performed across all components of a vector time point), and where the multi-variate normal unrolls into a product of scalar Gaussians because of the diagonal covariance matrix.

Likelihood Function For Multi-Class, Vector Time Series

We can now write the general CPM log likelihood function using the same notation, $\mathcal{L}^p \equiv \mathcal{L} + \mathcal{P}$, where again, $\mathcal{L}$ is the likelihood term arising in a (conditional) HMM model, and $\mathcal{P}$ is the log of the penalty/prior terms. Thus the two components of the general CPM log likelihood are

$$\mathcal{L} \equiv \sum_{k=1}^{K} \log \prod_{\boldsymbol{\pi}\to\{\tau_i,\phi_i\}} p(\pi_1) \left(\prod_{i=1}^{N}\prod_{d=1}^{D} \mathcal{N}(\mathcal{X}_{i,d}^k|\mathcal{Z}_{\tau_i,d}^{\omega^k}\phi_i u^k, \xi_d^k)\right)\left(\prod_{i=2}^{N} T_{\pi_{i-1},\pi_i}^k\right) \tag{4.12}$$

$$\mathcal{P} \equiv -\lambda \sum_{c=1}^{C} \bar{u}^c \sum_{j=1}^{M-1}\sum_{d=1}^{D} (\mathcal{Z}_{j+1,d}^c - \mathcal{Z}_{j,d}^c)^2 - \nu \sum_{c=1}^{C}\sum_{c'<c}\sum_{d=1}^{D}\sum_{j=1}^{M} \log\left(1 + \frac{(\mathcal{Z}_{j,d}^c - \mathcal{Z}_{j,d}^{c'})^2}{\varsigma}\right) + \ldots$$

$$\ldots + \sum_{k=1}^{K} \log \mathcal{D}(\kappa_v^k|\{\eta_v^k\}) + \log \mathcal{D}(s_v|\{\eta_v'\}) + \sum_{k=1}^{K} \log \mathcal{N}(\log u_k|0, w). \tag{4.13}$$

Summary of the Single Class EM-CPM

Notation:

$\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_{N^k}^k)$ *observed time series*

$\boldsymbol{z} = (z_1, z_2, ..., z_M)$ *latent trace*

ϕ_i^k *scale state for i^{th} time point in the k^{th} time series*

τ_i^k *time state for i^{th} time point in the k^{th} time series*

ξ^k *HMM emission variance*

u^k *global scaling factor for observed trace k*

HMM portion of the model:

$$\log p(\boldsymbol{x}^k) = \log p(\pi_1^k) + \sum_{i=1}^{N} \log \mathcal{N}(x_i^k | z_{\tau_i^k}^{w^k} \phi_i^k u_{\tau_i^k}^k, \xi^k) + \sum_{i=2}^{N} \log T^k_{\pi_{i-1}^k, \pi_i^k}$$

$$T^k_{\pi_j,\pi_i} \equiv p(\pi_i|\pi_j) = p(\phi_i|\phi_j)p^k(\tau_i|\tau_j) \quad \textit{HMM hidden state factorization}$$

$$p^k(\tau_i|\tau_{i-1}) = \text{Multinomial}(\kappa_1^k, \kappa_2^k, \ldots, \kappa_J^k) \quad \textit{HMM time transition distribution}$$

$$p(\phi_i = a|\phi_{i-1} = b) = \text{Multinomial}(s_1, s_0, s_1) \quad \textit{HMM scale transition distribution}$$

Priors:

$$p(\boldsymbol{z}) \propto \exp\left(-\lambda \bar{u} \sum_{j=1}^{M-1} (z_{j+1} - z_j)^2\right) \quad \textit{prior on latent trace,}$$

$$\text{where } \bar{u} \equiv \frac{\sum_k {u_k}^2}{K}$$

$$p(\log u_k) = \mathcal{N}(\log u_k | 0, w), \quad \textit{prior on global scaling factor}$$

$$p(\boldsymbol{\kappa}^k) = \mathcal{D}(\boldsymbol{\kappa}^k|\boldsymbol{\eta}) \quad \textit{prior on time transition multinomial parameters}$$

$$p(\boldsymbol{s}) = \mathcal{D}(\boldsymbol{s}|\boldsymbol{\eta}') \quad \textit{prior on scale transition multinomial parameters}$$

4.5 Training the CPM with EM

To train the EM-CPM – that is, to find a good setting of the parameters – we will maximize the penalized likelihood, $\mathcal{L}^p$, with respect to the parameters we wish to fit, namely, $\{\boldsymbol{\mathcal{Z}}^c\}$, $\{u^k\}$, $\{\kappa_i^k\}$, s_0, s_1 and $\{\xi_d^k\}$ (integrating out the HMM hidden states). Of course this assumes that ν and λ are fixed and that all hyper-parameters have been suitably chosen. One could perform

the optimization directly by appealing to a numerical solver[2] such as conjugate gradient and relying on properties of the HMM to efficiently compute the log likelihood function and its derivative for particular parameter settings (see for example [72]). However, we appeal to the well known Expectation-Maximization algorithm, for which we have provided a refresher in Appendix C. Of course any method one chooses to use for maximizing the penalized likelihood will be subject to problems of local minima.

In deriving an EM algorithm specific to the EM-CPM, we need an expression for the expected complete log likelihood of our model, $\left\langle \mathcal{L}^p_{\text{comp}} \right\rangle$ which is the average of the penalized joint log likelihood with respect to the posterior distribution over hidden states, $p(\{\boldsymbol{\pi}^k | \{\boldsymbol{\mathcal{X}}^k\})$. If we enumerate the hidden states, $s = 1, \ldots, S$, then the expected complete log likelihood is given by

$$
\begin{aligned}
\left\langle \mathcal{L}^p_{\text{comp}} \right\rangle &\equiv \langle \mathcal{P} \rangle + \sum_{k=1}^{K} \left\langle \log p(\boldsymbol{\mathcal{X}}^k, \boldsymbol{\pi}^k) \right\rangle \\
&= \mathcal{P} + \sum_{k=1}^{K} \left\langle \log p(\pi_1^k) + \sum_{i=1}^{N} \sum_{d=1}^{D} \log \mathcal{N}(\mathcal{X}_{i,d}^k | \mathcal{Z}_{\tau_i,d}^{\omega^k} \phi_i u^k, \xi_d^k) + \sum_{i=2}^{N} \log T^k_{\pi^k_{i-1}, \pi^k_i} \right\rangle \\
&= \mathcal{P} + \sum_{k=1}^{K} \left\langle \log p(\pi_1^k) \right\rangle + \sum_{i=1}^{N} \sum_{d=1}^{D} \left\langle \log \mathcal{N}(\mathcal{X}_{i,d}^k | \mathcal{Z}_{\tau_i,d}^{\omega^k} \phi_i u^k, \xi_d^k) \right\rangle + \sum_{i=2}^{N} \left\langle \log T^k_{\pi^k_{i-1}, \pi^k_i} \right\rangle \\
&= \mathcal{P} + \sum_{s=1}^{S} \sum_{k=1}^{K} p(\pi_1^k = s | \boldsymbol{\mathcal{X}}^k) \log T^k_{0,s} + \ldots \\
&\ldots + \sum_{s=1}^{S} \sum_{i=1}^{N} \sum_{d=1}^{D} p(\pi_i^k = s | \boldsymbol{\mathcal{X}}^k) \log \mathcal{N}(\mathcal{X}_{i,d}^k | \mathcal{Z}_{\tau_i,d}^{\omega^k} \phi_i u^k, \xi_d^k) + \ldots \\
&\ldots + \sum_{s=1}^{S} \sum_{s' \neq s} p(\pi_{i-1}^k = s, \pi_i^k = s' | \boldsymbol{\mathcal{X}}^k) \sum_{i=2}^{N} \log T^k_{s,s'},
\end{aligned}
$$

where $\mathcal{P}$ is defined in Equation 4.13 and $\langle \mathcal{P} \rangle = \mathcal{P}$ because it does not depend on the hidden states.

4.5.1 E-Step

In the E-Step of our EM algorithm we can exactly and efficiently compute the marginals of the posterior, $p(\pi_i^k = s | \boldsymbol{\mathcal{X}}^k)$ and $p(\pi_{i-1}^k = s, \pi_i^k = s' | \boldsymbol{\mathcal{X}}^k)$. This is accomplished by the well-known Forward-Backward algorithm [65] (an instance of belief propagation) as shown

[2] We briefly explored this avenue for the single-class, scalar CPM, and found that it did not systematically increase (or decrease) the speed or quality of the solution found, though we do not report detailed results here.

in Appendix B. If we decide not to learn the state transition parameters, then we can omit computing the pair-wise marginals of the posterior, $p(\pi_{i-1}^k = s, \pi_i^k = s' | \boldsymbol{\mathcal{X}}^k)$. Note that because of the sparse transitions we have chosen, the complexity of this algorithm is linear in the number of states, rather than quadratic as in the general case.

4.5.2 M-Step

Each parameter or group of parameters has its own portion of the M-Step. For those that parameters that have analytical M-Steps we need to find the parameter setting which maximizes $\left\langle \mathcal{L}_{\text{comp}}^p \right\rangle$. Thus, given posteriors from the E-Step, we first compute $\frac{\partial \left\langle \mathcal{L}_{\text{comp}}^p \right\rangle}{\partial a}$ and set it equal to zero. If the parameter being updated can be solved for analytically, then we are done. If it cannot be solved for analytically, we need to resort to a numerical optimizer routine to perform the desired partial maximization (partial because it is over only some of the parameters), in which case we usually use both the expected complete log likelihood function values, as well as its derivatives.

The M-Step updates for u^k, ξ_d^k, and $\boldsymbol{\mathcal{Z}}^c$ are coupled. Thus we arbitrarily pick an order to update them and as one is updated, its new values are used in the updates for the coupled parameter updates that follow it (*i.e.*, coordinate optimization). In our experiments we updated in the following order: ξ_d^k, $\boldsymbol{\mathcal{Z}}^c$, u^k. The other parameters, κ_v^k and s_v, are completely decoupled.

We now go over each portion of the M-Step, but leave the detailed derivations and equations to Appendix D. The marginals of the posteriors are abbreviated as $\gamma_s^k(i) \equiv p(\pi_i^k = s | \boldsymbol{\mathcal{X}}^k)$ and $\xi_{s,s'}^k(i) \equiv p(\pi_{i-1}^k = s, \pi_i^k = s' | \boldsymbol{\mathcal{X}}^k)$. These are computed during the E-step, and held constant during the M-step.

Latent Trace M-Step

For the single-class case, with $\lambda = 0$, the update for each scalar value $\mathcal{Z}_{d,i}$ is independent of all others, and can be analytically calculated from one equation in one unknown. If, however, $\lambda \neq 0$, then values within the one class become coupled, and a tri-diagonal linear system needs to be solved, but the solution remains analytical. For the general, multi-class model, all latent trace values, $\mathcal{Z}_{i,d}^c$, both within a class-specific trace (for fixed c), and across class-specific traces, are coupled together non-linearly (although are independent across different dimensions d). At this point, the update is no longer analytical, and we require a numerical solver.

HMM Emission Variance M-Step

The updates for HMM emission variance, without any constraints, are analytical, and given by

$$\xi_{d'}^{k'} = \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} \gamma_s^k(i) (\mathcal{X}_{i,d'}^{k'} - \mathcal{Z}_{\tau_s,d'}^{\omega^{k'}})^2. \tag{4.14}$$

This update is the posterior-weighted sample variance of the observed time points relative to their latent trace mean – the usual type of intuitive outcome for analytical M-Step updates, in which the update we would have achieved had we known the hidden state values is slightly altered so as to account for our uncertainty in these states. However, we do impose a constraint – for fixed d, across all k, k' in class c, $\frac{\xi_{d'}^{k'}}{\xi_{d'}^{k}} < F$ (we used $F = 4$ in our experiments). That is, the HMM emission variances, for the observed time series within a class, cannot be more than a factor F apart. An alternative view of this constraint is as a set of linear constraints on the log of the variances ($\log \xi_{d'}^{k'} - \log \xi_{d'}^{k} < \log F$). We imposed this constraint because in early experiments, we found that occasionally the latent trace would otherwise latch on to one of the observed time series, and that that time series' HMM variance would become orders of magnitude smaller than the others.

The M-step for the variances are easily modified to account for this constraint, assuming that they are initialized so as to obey this constraint. The idea is to first calculate the unconstrained updates, and then for each ξ_d^k in the group being updated, see if its unconstrained update is acceptable. If it is, then update it. If it isn't, update in the correct direction, as much as possible (so as not to violate the constraint). Continue to cycle through the set of ξ_d^k until none of them can be changed (or stopping earlier for a partial M-Step), at which point the algorithm stops. This algorithm is guaranteed to not decrease the expected complete log likelihood, since every time we change a value of ξ_d^k, we necessarily increase the expected complete log likelihood. Furthermore, the constraint is guaranteed to be satistified, since it is satisfied at the start (by our initialization condition), and since we never make an update that violates it. Lastly, the algorithm is guaranteed to stop since at every iteration, we move one of the variances closer to its unconstrained update. If every variance reaches its unconstrained update, the algorithm stops. If every variance does not reach its unconstrained update, then it must have stopped changing, since each parameter always moves in the same direction (toward its unconstrained optimum).

State Transition Parameters M-Step

The parameters for both the time state transition probability distributions and those of the scale state transition probability distribution are analytical, and given by:

$$\kappa_v^k = \frac{\eta_v^k + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s''}^k(i)}{\sum_{j=1}^{J} \eta_j^k + \sum_{j=1}^{J} \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=j\}} \sum_{i=2}^{N} \xi_{s,s''}^k(i)}$$

$$s_v = \frac{\eta'_j + \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{\{s''\in H(s,v)\}} \sum_{i=2}^{N} \xi_{s,s''}^k(i)}{\sum_{j=1}^{2} \eta'_j + \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{\{s''\in H(s,1),H(s,0)\}} \sum_{i=2}^{N} \xi_{s,s''}^k(i)}$$

where $H(s,j) \equiv \{s'|s' \text{is exactly } j \text{ scale states away from } s\}$, and $v \in 0..1$. Notice that the parameters of the Dirichlet priors serve as pseudo-counts, and that the updates are once again, intuitively pleasing.

Global Scaling Parameter M-Step

The update for each u^k is independent of the others. With our u^k prior the update is not analytically tractable, although without the prior, it is given by

$$u_k = \frac{\sum_{d=1}^{D} \sum_{s=1}^{S} z_{\tau_s} \phi_s \sum_{i=1}^{N} \gamma_s^k(i) x_i^k}{\sum_{d=1}^{D} \sum_{s=1}^{S} (z_{\tau_s} \phi_s)^2 \sum_{i=1}^{N} \gamma_s^k(i)}.$$

However, since we use the prior, we resort to a numerical solver for these parameters. The derivatives of the expected complete log likelihood can be found in Appendix D.

4.5.3 Initialization and Setting of Hyper-parameters

During our experiments, unless otherwise noted, we have set the initial parameters as follows. The latent trace is set to be a naively upsampled (simply repeating every measurement, next to itself), and then smoothed version of one of the observed time series, where smoothing is performed by a moving average filter with a window that is 20% the length of the vector being smoothed. The HMM emission noise for each dimension is set to be standard deviation of all data points across all observed time series for that dimension. The probability of moving ahead 1, 2, or 3 time states is set to respectively 30%, 40% and 30%. The probability of staying in the same scale state is 90%, and all scaling parameters are set to unity. For the state transition hyper-parameters, we set $\eta_v = \eta'_v = 5$ (for all v), and for the scaling hyper-parameters, $w = \ln(1.5)$. The probability for starting in any one state $T_{0,\pi}$ was the same for

all states π (albeit some starting states will be too close to the end of the latent trace to ever produce a feasible path, and these will be ruled out by the forward-backward algorithm).

4.6 Obtaining an Alignment after Training

After training, one may examine either the latent trace or the alignment of each observable time series to the latent trace (and to each other). Such alignments can be achieved by several methods, including use of the Viterbi algorithm to find the highest likelihood path (MAP) through the hidden states [65], or sampling from the posterior over hidden state sequences (see Appendix B for details of these computations). Generally we use Viterbi alignments since we have found these to be very reasonable, although we investigate the posterior in Section 4.9.7. Note that when discussing an 'alignment' resulting from a CPM, we by default mean alignment *and* scaling, unless otherwise noted.

4.6.1 Unrolling an Alignment

When the EM-CPM is used with inherently vector time series data, such as with LC-MS (or any hyphenated LC technique), we normally model a reduced dimensionality version of the data for alignment. That is, we reduce the dimensionality of the signal at each time point. For example, with LC-MS data, we have used the TIC (Total Ion Count), which is a scalar version of the inherently vector time series obtained by summing out over all dimensions to obtain a scalar value for each time point. In these instances, although we seek to achieve a good alignment in the original space (*e.g.*, time $\times$ *m/z* space in LC-MS), for computational efficiency we use data in a reduced space. In such cases, we will refer to unrolling the 1D alignment to a 2D alignment, by which we mean that for a particular time series, we take its Viterbi alignment (or however one obtained an alignment), found using the EM-CPM with reduced dimensionality data, and systematically apply this alignment to each of the original features (*e.g.*, *m/z* slices). This gives us the most naive of vector time series alignment, where each compoment of one vector gets warped together with the others at the same time point.

4.6.2 Alignment Scores

In the Section 4.9 where we explore the behaviour of the EM-CPM under various conditions, it will be useful to define what we refer to as a 1D score and a 2D score which reflect on how tightly the observed time series are aligned (and scaled), both in the space in which they are aligned, and in the desired alignment space. These are computed by aligning the data in the alignment space (which will give the 1D score), or in the unrolled space (which will give the 2D

score). Since an alignment using the EM-CPM produces a sparse mapping from observed time to latent time, we need to fill in the gaps, which we do by way of linear interpolation. Thus, if a particular latent time has not been mapped for one particular observed time series then we assign it the value of the distance-weighted linear combination of its two nearest neighbours in latent time. After obtaining this aligned and 'filled in' data set, we next compute the point-wise standard deviation across replicates at every point in this space.[3] The score is then the mean of these point-wise standard deviations. One might be tempted to think that these scores measure the quality of an alignment, and that we could use them to judge how well various modelling choices were working relative to one another. However, these scores only reflect how well observed time series have been forced to match each other, and do not take into account possible overfitting. This is definitely the case for the 1D scores, and potentially also so for the 2D scores. However, the 2D scores give us the possibility of a more unbiased opinion, if the full set of data is not used for alignment. For example, in using the TICs to align LC-MS data, one might consider the 2D scores as almost a hold out measure representative of the quality of alignment (though not one which could be used to assess how using increasing dimensions in the alignment affects quality). In any case, we should be careful in interpreting better 1D or 2D scores as better alignments. We simply use these scores as one extra measure to help us characterize our experimental results.

Alignment Score Definition

Formally, the scores are defined as follows. Let $\boldsymbol{\mathcal{Y}}^k = (\boldsymbol{y}_1^k, \boldsymbol{y}_2^k, \ldots, \boldsymbol{y}_N^k)$ be the k^{th} original time series data that we wish to align in its full vector space, where $\boldsymbol{y}_i^k = (\mathcal{Y}_{i,1}^k, \mathcal{Y}_{i,2}^k, \ldots, \mathcal{Y}_{i,W}^k)$ is, for example, the vector containing the ion abundance at all *m/z* values for time point i. In other words, one non-reduced LC-MS experiment is represented as the $N \times W$ data matrix $\boldsymbol{\mathcal{Y}}^k$. Now assume that for the purposes of alignment we have reduced the dimensionality of each of the $\boldsymbol{\mathcal{Y}}^k$ into $\boldsymbol{\mathcal{X}}^k = (\boldsymbol{x}_1^k, \boldsymbol{x}_2^k, \ldots, \boldsymbol{x}_N^k)$, where $\boldsymbol{x}_i^k = (\mathcal{X}_{i,1}^k, \mathcal{X}_{i,2}^k, \ldots, \mathcal{X}_{i,D}^k)$, (*i.e.,* the *m/z* values have been reduced to D bins).

Now define the function $\textit{align}(\boldsymbol{\mathcal{G}}, \boldsymbol{\pi}, u) = \hat{\boldsymbol{\mathcal{G}}}$ which takes as input a vector/scalar time series with N time points, each of which has a vector of measurements of length V, and also a path through the hidden states of the CPM, $\boldsymbol{\pi}$. The function returns a time-upsampled, aligned and scaled version of $\boldsymbol{\mathcal{G}}$ that has M time points (where M is the length of the latent trace used in

[3]The point-wise standard deviation is the same as the root mean squared (RMS) deviation, except that $N-1$ is used in the denominator for the former, rather than N in the RMS. We simply use this measure as a guiding heursitic, and either of these statistics would serve our purpose.

the CPM). Specifically, $\forall l, \forall j \in 1..M$, if $\exists t, \pi_t = j$, then

$$\hat{\mathcal{G}}^k_{j,l} = \frac{\mathcal{G}^k_{t,l}}{u\phi_t},$$

and $\forall d \in 1..D, \forall j \in 1..M$, if $\forall t, \pi_t \neq j$, then $\hat{\mathcal{G}}^k_{j,d}$ is obtained by linear interpolation between $\hat{\mathcal{G}}^k_{j',d}$ and $\hat{\mathcal{G}}^k_{j'',d}$ where j' and j'' are the nearest two values for which $\exists t, \pi_t = j'$, $\exists t, \pi_t = j''$.

Finally, let a hidden state sequence through a CPM for the k^{th} time series be $\boldsymbol{\pi}^k$, let $\hat{\boldsymbol{\mathcal{X}}}^k = \textit{align}(\boldsymbol{\mathcal{X}}^k, \boldsymbol{\pi}^k, u^k)$ be the aligned, scaled and interpolated data in the reduced dimensionality data space, and let $\hat{\boldsymbol{\mathcal{Y}}}^k = \textit{align}(\boldsymbol{\mathcal{Y}}^k, \boldsymbol{\pi}^k, u^k)$ be the unrolling of that alignment, scaling and interpolation to the original (not reduced) data space. Then the 1D and 2D scores over K time series are given by

$$\text{1D Score}(\{\hat{\boldsymbol{\mathcal{X}}}^k, u^k, \boldsymbol{\pi}^k\}) = \frac{\sum_{i=1}^{N}\sum_{d=1}^{D} \text{std}_k(\hat{\mathcal{X}}^k_{i,d})}{ND}$$
$$\text{2D Score}(\{\hat{\boldsymbol{\mathcal{Y}}}^k, u^k, \boldsymbol{\pi}^k\}) = \frac{\sum_{i=1}^{N}\sum_{v=1}^{V} \text{std}_k(\hat{\mathcal{Y}}^k_{i,v})}{NV}$$

where $\text{std}_k(\mathcal{G}^k_{i,d})$ is the unbiased sample standard deviation at time point i, for component d, of replicate k.

4.7 Scaling Spline Alternative

The EM-CPM HMM scale states introduced in Section 4.2 are computationally expensive – the number of hidden states (*i.e.,* joint scale/time states) scales linearly with Q, the number of scale states. In turn, the time complexity of the Forward-Backward algorithm scales as Q (if not for the sparse transitions, it would scale as Q^2). If we and want to also learn the state transition parameters during EM, then time complexity scales as Q^2. Even if we choose not to learn the state-transition parameters, the increase in time with the number of scale states may not be desirable. Thus we develop a modified EM-CPM in which we eliminate the hidden scale states, and instead introduce scale flexibility by way of a set of spline parameters. Thus the number of hidden states is decreased by a factor Q, but with an added cost of having to compute the MAP estimate of the scaling spline parameters during the M-Step. Depending on the number of control points in the spline, this may save us quite a bit of CPU time. In Section 4.9.6 we explore these alternatives experimentally.

Each spline tells us by what amount to scale each observed point relative to the latent trace from which it is being generated. The more control points we use, the more local the scaling. The same scaling spline is used for all dimensions, d, but is unique to each observed time

series. For each spline we use G control points, evenly spaced in latent time. Note that the position of the control points is fixed ahead of time and never changes during learning. The first control point is located at the first latent time point (*i.e.*, 1), and the last control point at the last latent time point (*i.e.*, M). For any latent time, j which does not have a control point, we linearly interpolate between its two closest control points. One could of course use more expressive splines instead, but we felt that linear splines would sufficiently serve our purpose.

More formally, let c_j^- be the latent time of the control point immediately to the left of time state j and c_j^+ be the latent time of the control point immediately to the right of hidden time state j. Let μ_m^k be the value of the control point occuring at latent time m (only defined for those latent times which have a control point), then the scaling value defined by the spline at any latent time, j, for observed time series k, represented by u_j^k, and is given by[4]

$$u_j^k = \frac{(j - c_j^-)\mu_{c_j^+}^k + (c_j^+ - j)\mu_{c_j^-}^k}{c_j^+ - c_j^-},$$

unless j itself has a control point, in which case $u_j^k = \mu_j^k$.

This change does little to alter the EM algorithm for the EM-CPM. Our E-Step will be less computationally burdensome since we have removed many states. We will need to modify our M-Step for the global scaling parameters, u^k, to instead update the values at our spline control points. The emission probabilities change slightly – replacing u^k with u_j^k as follows, $p(\mathcal{X}_{i,d}^k|\pi_i, \boldsymbol{z}, \xi, u_{\tau_i}^k) \equiv \mathcal{N}(\mathcal{X}_{i,d}^k|\mathcal{Z}_j^{\omega^k}\phi_i u_{\tau_i}^k, \xi)$.

Updating the scaling spline parameters in the M-Step can be performed seperately for each observed time series (but simultaneously across all control points for a given observed time series), but the update is not analytical. Thus we can make use of the derivative $\frac{\partial\langle\mathcal{L}_{\text{comp}}^p\rangle}{\partial\mu_m^k}$ (applying the same log normal prior we applied to the u^k, only now applied to the μ_m^k), as provided in Appendix D.

4.8 Modeling Choices in the Design of the EM-CPM

There are several design decisions we have made which introduce approximations to what we believe to be the true generative process underlying the type of data for which we might use the model. Here we discuss these choices, why we made them, and what other choices could have been made.

- One could argue that the smoothness prior on the latent trace (taking the form of a 1D

[4]This is a slight abuse of notation, as previously we used u^k to denote the global scaling parameter, whereas we have overloaded the meaning here, so that u_j^k is the scaling value at time j as assigned by the spline.

Gaussian Markov random field) is rather weak, and unlikely to generate the kind of latent traces we expect for our data. A Gaussian Process could provide a more suitable prior in this regard, however, the added computational cost could prove to be a large burden. In any case, with enough data, our weak prior is sufficient, and in practice has so far worked well.

- The HMM emission variances, ξ_d^k, are independent of the intensity of the signal they model (since the noise is additive) – that is, emission probability distributions are homoscedastic, when we might expect that in reality these are heteroscedastic, and more specifically, that the variance would get larger as the value of the signal gets larger, perhaps in a way such that the relative noise remains constant (*i.e.*, multiplicative noise). This issue could be readily addressed by taking the log transform of the observed time series before using the CPM. However, such a solution may compromise other parts of the model (*e.g.*, the multiplicative scaling). A compromise solution allowing some heteroscedasticity while retaining tractability would be to use a finite set of $\{\xi_{d,j}^k\}$ where j would correspond to a bin of signal intensities. If $|j|$ were not too large, this would remain a feasible option both computationally, and from a modeling perspective (in that there would still be enough data to fit each ξ_j^k). One could go further with this and use a spline to interpolate the variance over the full range of signal intensity, although this would complicate training. In this thesis, we do not change the model in these ways, except to sometimes use the log transform of the data.

- Somewhat similarly to the issue above, our smoothing penalty does not take into account the magnitude of the signal, and thus portions of the latent trace which are very non-smooth, but lie in low magnitude region will not be penalized much relative to an equally varying signal in a higher magnitude region. Again, one would ideally like to use differences which are proportional to the magnitude, or use something altogether different, such as the sum of the point-wise finite difference second derivatives over the latent trace, but these would make the M-Step more complicated (changing it from analytical to non-analytical in the $\lambda = 0$ case).

- We have factored the state transitions into scale state transitions and time state transitions, but one could imagine that there are certain regions in time in which the scaling correction changes more rapidly than in other regions of time. Coupling these two types of states together would not alter the sparseness of the transition probability matrix, nor cause much extra computational burden, though we would rely more on a prior for learning the transitions, as less data per parameter would be available. However, we don't feel that this refinement would actually provide much benefit in terms of modeling power, as

the factored assumption does not seem particularly strong or inappropriate in our data sets (although it could be).

- The way in which we chose to extend the scalar time series CPM to vector time series was the simplest possible, and richer extensions would likely provide better results, but at greater computational cost. For example, one shortcoming of our method is that we ignore correlation among the vector components. This could potentially be overcome by pre-processing the data to remove such correlations, for example using PCA (Principle Components Analysis), ICA (Independent Components Analysis) or FA (Factor Analysis), or by using a non-diagonal covariance in the HMM emission probability distributions (thereby coupling together all dimensions and making training and inference more expensive). Another shortcoming of our method is that it might be the case that in some domains, different vector components require different alignments and/or normalizations, while we align/normalize all components in an identical manner.

- In some domains, such as LC-MS, in addition to having the ion abundance measurements, we also have time stamps for each experiment. For example, for a TIC, $\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_{N^k}^k)$, we also usually have access to time stamps for each associated data point, $\boldsymbol{t}^k = (t_1^k, t_2^k, ..., t_{N^k}^k)$. Thus, one could consider devising a model which also incorporates this information. However, the major problem we are trying to address is that of non-calibrated time, both between experiments and within. Hence the time stamps, though they may contain some information, likely do not contain very much that could help us. In fact, the more a data set requires use of a time alignment algorithm, the less information these time stamps will convey. Hence we have chosen to ignore them altogether in our model. One scenario in which we might make use of time stamps would be when we believe, *a priori*, that different time series were generated at very different time sampling frequencies. In such a situation, one could naively alter the model to allow for more extreme warping (for example by allowing any point in any time series to jump ahead a large number of time steps). However, this could be computationally expensive (since the transition matrix would be less sparse), and also prone to overfitting. One might instead tailor the time transitions priors to account for our prior knowledge by roughly constraining densely sampled time series (high frequency sampling) to jump relatively few steps ahead in time, while those sampled at a much lower frequency could be constrained to jump ahead only in much larger jumps. In other words, one could use the observed time stamps to provide a prior on the time transition probabilities.

4.8.1 Relation to Input-Output HMMs

The EM-CPM has some structural similarities (in the sense of the graphical model set-up) to Input/Output HMMs (IOHMMs), also called Conditional HMMs [11], though the problems these two models address are quite different. IOHMMs extend standard HMMs [65] by conditioning the emission and transition probabilities on an observed input sequence. Each component of the output sequence corresponds to a particular component of the input. Training of an IOHMM is supervised — a mapping from an observed input sequence to output target sequence is learned. Our EM-CPM also requires input and thus is also a type of conditional HMM. However, the input is unobserved (but crucially it is shared between all replicates) and hence learning is unsupervised in the EM-CPM. One could also take the alternative view that the EM-CPM is an HMM with an extra set of parameters, the latent trace, that affect the emission probabilities and which are learned by the model.

4.9 Single-Class Experiments

We mainly use one data set used to explore the single-class EM-CPM – a set of 11 experimental replicates of LC-MS data.[5] Protein was extracted from lysed *E. coli* cells, then digested, and finally, subjected to capillary-scale LC-MS coupled on-line to an ion trap mass spectrometer. The same LC column was used throughout.

To aid convergence to a good local optimum and to speed up training, we pre-processed the LC-MS data set by coarsely aligning and scaling each time series as follows: We 1) translated each time series so that the center of mass of each time series was aligned to the median center of mass over all time series, 2) scaled the abundance values such that the sum of abundance values in each time series was equal to the median sum of abundance values over all time series. Additionally, prior to this pre-processing, all *m/z* values were mapped to bins of width $\frac{1}{2}$ *Th*[6] which are retained throughout all other processing (similarly done in [68, 86, 48, 50]). In our data set, we had roughly 1000 rows corresponding to 1000 time points, and 2400 columns corresponding to *m/z* bins of width 0.5 *Th* spanning 400-1600 *Th*. This same pre-processing was applied before using other algorithms such as DTW and COW.

In some cases we use the TICs for alignment, while in other cases we use several *m/z* bins. In the latter case, each bin is designed to contain roughly equal ion abundance over all samples being aligned. The TICs (Total Ion Count) of the pre-processed data are shown in Figure 4.3. For this data set, we use 880 latent times into which the 400 real time points can map. With 7

[5] We thank Andrew Emili for providing us with this data. All LC-MS data reported in this thesis was generated in his laboratory.

[6] *Th* stands for Thompson and represents the units mass over charge (*m/z*).

scale states, this gives $7 \times 880 = 6160$ hidden states.

Most experiments we performed here are very data set dependent. We do not present their conclusions as hard and fast rules, but rather present them as being representative of this type of LC-MS data. We also present them to show what modeling choices should be given consideration.

At the end of this section, we give a brief demonstration of the EM-CPM applied to the alignment of speech data, to demonstrate the potential breadth of application of CPMs, though we do not investigate this domain in depth.

Occasionally during our experiments we will want to use a hold out set, as a less expensive proxy to cross-validation. When using such a hold out set, we fix the parameters which are not specific to a particular observed time series (that is, the latent traces, $\{\boldsymbol{\mathcal{Z}}^c\}$, and the parameters, s_0, s_1) to the value learned during training, and then run a restricted version of our EM algorithm in order to learn the other parameters, namely, the scaling parameters (u_k or μ_τ^k), the emission noise (ξ_d^k), and the state transition parameters (κ^k). Note that this is not a truly unbiased validation, since our algorithm has the potential to overfit during restricted EM, albeit to a much lesser extent than during training when the latent trace can also change. During this restricted EM, no constraints are place on the ξ_d^k (as opposed to during training where they are restricted to be within a factor of each other). The motivation for these constraints was to prevent the latent trace from latching on to one observed time series. If the latent trace has been learned, as in this restriced EM, then the constraints serve no purpose.

4.9.1 Demonstration of the Model in Action

To get a feel for the EM-CPM we start off with a general overview of the results of using the single class, scalar EM-CPM on the TICs of the 11 replicates of LC-MS data, with no smoothing prior ($\lambda = 0$). Figure 4.4 contrasts the data before and after alignment (using Viterbi/MAP alignment), while Figure 4.5 shows one of the 11 observed time series superimposed on the final latent trace. Figure 4.6 shows how the scalar TIC alignment unrolls into a full two-dimensional alignment of the LC-MS data for a zoomed in portion of the data.

4.9.2 Convergence

In order to get a feel for how the EM-CPM converges, we trained the scalar, single class model on six of eleven LC-MS TIC data, with no smoothing ($\lambda = 0$), for 1000 iterations, using scaling states (and no scaling spline). Figure 4.7 shows the fractional change in the parameters,[7] as

[7] We define the fractional change, for positive parameter p, between time steps t and $t-1$ to be $\frac{abs(p^t - p^{t-1})}{(p^t + p^{t-1})/2}$.

well as the 1D and 2D scores. These scores are essentially a summary statistic of the point-wise standard deviation, in 1D (in TIC space), or in 2D (after unwrapping the TIC alignments into the full LC-MS space) and were introduced in Section 4.6.2.

Within 300 iterations, none of the fractional changes are larger than 10^{-6}, and this is also the point at which maximal convergence appears to occur. The time transition parameters (κ_v^k) converge most tightly (lowest fractional change), except for the scale transition parameters (s_0, s_1) which completely stop changing after 5 iterations and can't be seen on the graph. The HMM emission noise converges next most tightly (ever so slightly above the time transition parameters). The rest all converge less tightly and of a similar tightness to one another. These include the 1D and 2D alignment scores, the global scaling factors (u^k), and the latent trace (z), which can't be seen well because it is below the 1D and 2D score curves.

Note that of all the parameters, the u_k (global scaling), z (latent trace), and the 1D and 2D scores all coverge to a similar fractional value ($\sim 10^{-6}$), while the state transition parameters and the HMM emssion converge more tightly. The fact that the u^k don't seem to converge as tightly as the other parameters might due to the fact that its M-step update is not analytical, rather it uses an iterative numerical solver. That the latent trace converges less tightly may be a reflection of the fact that we used no smoothing $\lambda = 0$ and hence the very ends of the latent trace are essentially unconstrained since no observed points map to them.

Of course convergence properties may be different under different circumstances, such as with smoothing ($\lambda \neq 0$), with vector time series alignment, use of a scaling spline and fewer or more observed time series. In other experiments in this chapter, we take convergence to occur when the log likelihood changes by no more than 10^{-3} (on order of 100 iterations). However, we have previously used the EM-CPM on similar type data for 10-20 iterations with seemingly good results (visually).

4.9.3 Stability with Respect to Initialization

One might wonder how large a role the EM initialization plays in uncovering a good local maximum, or how different local maxima compare to one another. The parameter which might most affect this outcome is the latent trace, and so, using the same experimental conditions as in the previous section, we trained the EM-CPM using 13 different initializations for the latent trace. For the first six initializations, we used a smoothed, upsampled version of each of the 6 TICs being aligned, in turn. Smoothing was accomplished by way of a moving average smoother, using a span of 20% the length of the latent trace. We repeated this with a span of 10% to obtain another 6 initializations. Finally, we initialized with a flat trace, of magnitude equal to the mean of the six TICs over all time points. Each of these initial latent traces is

shown in Figure 4.8, and the corresponding recovered latent traces are shown in Figure 4.9. Additionally, the log likelihood, the 1D and 2D scores (using Viterbi alignments), the HMM emission noise (ξ^k), the number of iterations used, as well as all final latent traces superimposed are shown in Figure 4.10.

The log likelihood of each of the resulting fitted models is almost identical, however, the 1D and 2D scores and the HMM emission noises do vary, though not dramatically. The flat initial trace requires more iterations of EM to converge than most of the smooth trace initializations, though two of these take even longer. The recovered latent traces, though highly similar in nature, with similar features, do differ in fine detailed structure, in magnitude, and in relative position of features. However, it would seem that any one of these initializations would have provided us with a reasonable answer, even if not a completely unique one. In Chapter 6, on a different data set, we show that many different initializations do indeed lead to good quality solutions.

4.9.4 To Learn or Not To Learn the State Transition Probabilities

Learning the state transition probabilities is expensive. To learn them, we need to gather all pairwise marginals of the posterior over the hidden state sequence, making the algorithm more expensive. To examine the effects of learning the state transition parameters rather than not learning them, we trained the CPM on 6 of the 11 LC-MS TICs, and used 3 others as a hold out set, with no smoothing prior. The results are shown in Figure 4.11. Note that for the experiment where we did not learn the state transitions, these were fixed, *a priori*, to have a 0.3 probability of jumping ahead 1 or 3 time states, and 0.4 of jumping ahead 2, and also a 0.9 probability of staying in the same scale state rather than changing to an adjacent one.

From these results, it seems entirely reasonable to omit the learning of the state transition parameters for this type of data in order to achieve a large speed-up, when this is desired. Although the 1D-scores and and emission noise are not as good when we don't learn the transition parameters, the 2D-scores are comparable, as is the log likelihood, and the time saving is a factor of eight. Additionally, the contribution that the Dirichlet priors on the state transitions make to the overall log likelihood in the trained model are negligible.

4.9.5 Smoothing Parameter

We now explore how using a different amount of smoothing penalty (λ) affects the results, both on a training set as well as on a hold out set. Again, we use the TIC LC-MS data set, with 6 samples for training, and 3 as a hold out set. The recovered latent traces are shown in Figure 4.12, while other results are in Figure 4.13. Note that it is not meaningful to compare between

the hold out and the train set since these contained a different number of data points (not to mention different data points).

As λ becomes larger (10^{-14}), the running time per iteration increases. This could be due to the fact that the likelihood surface is much flatter, and thus components of the M-step requiring an iterative solver might take longer to reach the same level of convergence. Based on these results, for this type of data, we are comfortable using no smoothing, although we may want to use a tiny amount so as to avoid numerical issues arising from the ends of the latent trace being completely unconstrained if no observed time points map to them (or other latent trace points, although we have only observed this phenomenon at the ends of the latent trace).

Note also that there are actually two ways in which we can regularize the latent trace. One, as mentioned, is with the smoothing parameter. The other way would be to vary the ratio of observed time points to latent time points. As this ratio is increased, the model becomes more constrained (and hence regularized). We did not investigate this alternative.

4.9.6 Scale States Verus Scaling Spline

We now examine how results are affected by using different forms of scaling. In particular, we compare i) no scaling whatsoever (either with scale states or a scaling spline, or a global scaling factor), ii) using a scaling spline with 1, 3, 7, 10, 15, 20, 50, 100 control points, or iii) using the HMM scale states without a spline, but with a global scale factor. Various results are shown in Figure 4.14. We have used the LC-MS TICs, 6 for training and 3 as a hold out set, with no smoothing, and did not learn the state transition parameters.

Note that a maximum of 500 iterations of EM was permitted, and that for the scaling spline with anywhere between (and including) 3 to 100 control points, the algorithm did not converge to the specified threshold (relative change of 10^{-8} on the log likelihood) which may have to do with an increased reliance on iterative M-steps.

Use of hidden scale states provides a better 1D score, as does using increasingly more spline control points. However, the 2D score does not seem similarly improved – perhaps indicating that the scaling is overfitting, or perhaps that scaling should be performed differently at different *m/z* values.

The CPU times per iterations shown at the bottom of Figure 4.14 are plotted on a logarithmic scale. The ratios of CPU time per iteration of i) the HMM scale states over 2) each of 1,3,7,10,15,20,50 control points, are respectively 0.17, 0.18, 0.23, 0.29, 0.44, 0.66, and 3.66.

It's difficult to make any hard rules from these experiments, but it would seem that if one feels that much local scaling is required, then one might as well use the full HMM scale states, since the computational cost required to do so is less than using only 50 spline control points.

However, if one felt that only more global trends were likely to occur, then a scaling spline with just a few control points may be sufficient, and faster to train.

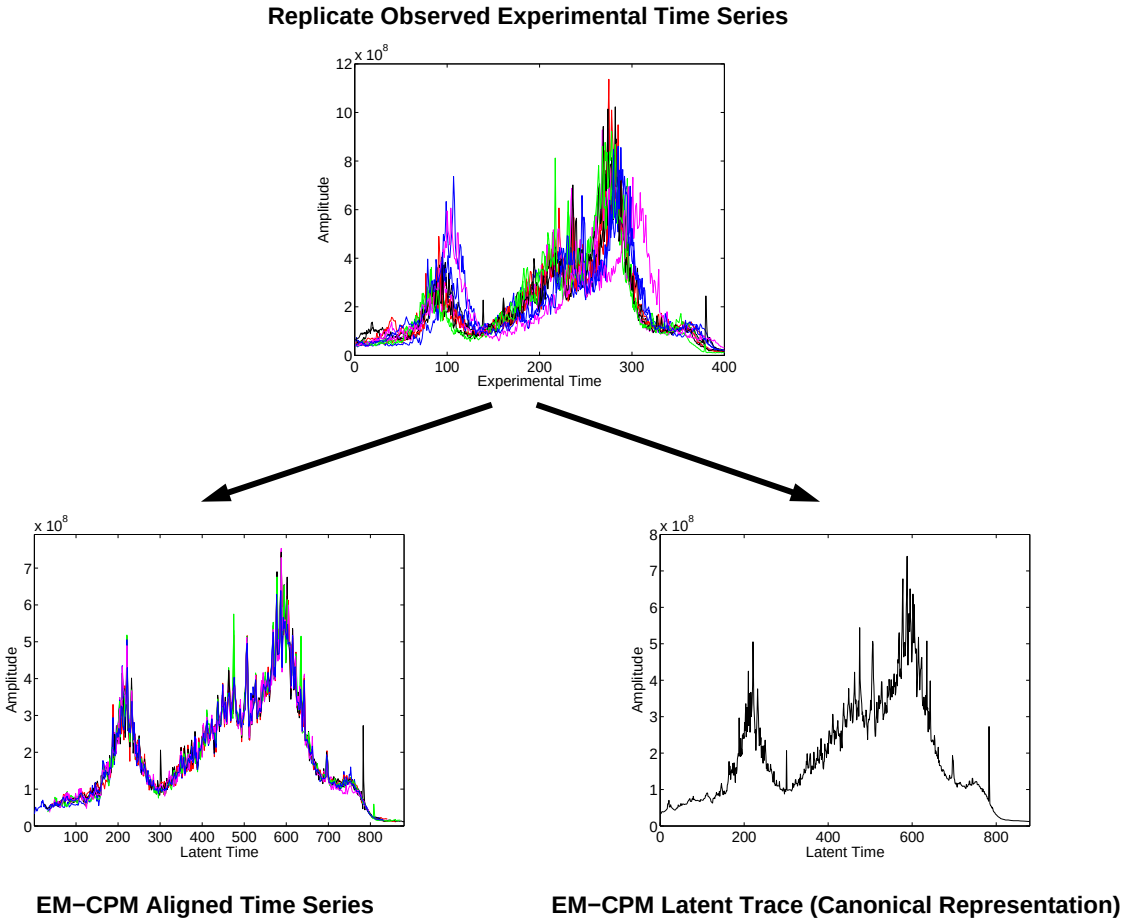


Figure 4.1: The motivation and desired end goal of a single-class Continuous Profile Model are shown. A set of eleven experimental time series are shown. These are measurements of the same underlying signal, but because of time sampling issues in the instrumentation, as well as noise inherent in the experimental machinery, the time series differ from one another. In particular, time both within and across samples is not consistent, and also the scaling of the signal changes, both within, and across the experimental replicates. To be able to leverage all of the information contained in these noisy experimental observations, it is desirable to align and scale them in a principled way, as well as to synthesize a summary of their shared properties, in the form of a canonical representation. Data shown are actually real data from an LC-MS experiment, aligned using our EM-CPM, with a 'latent trace' canonical representation provided by the trained EM-CPM. This experiment is described in more detail in Section 4.9.

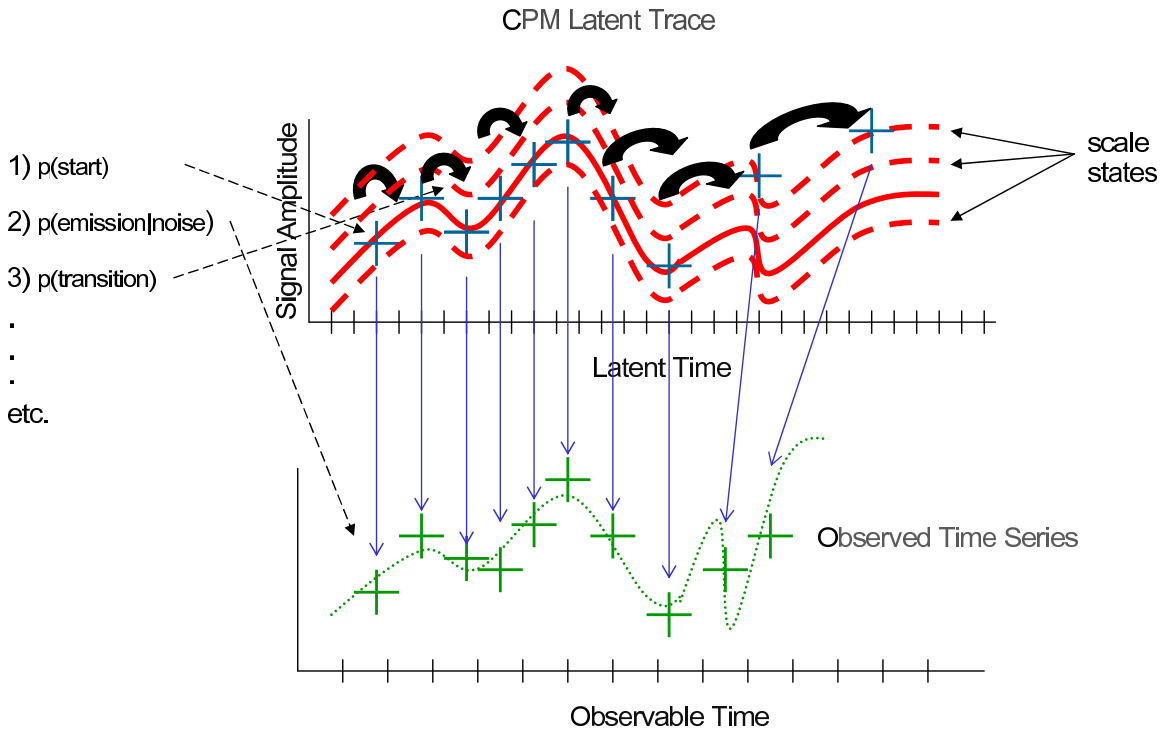


Figure 4.2: A graphical depiction of a single-class Continuous Profile Model (CPM) is shown. The single-class CPM is a generative model for experimental time series replicates which allows the scaling and time to be corrected, both within a time series, and between time series. This is accomplished by hypothesizing (and then learning) a so-called *latent trace*, shown as the solid red line. An observed time series is generated in the model by probabilistically choosing a starting *latent time* (time as indexed by the latent trace), and then emitting an observed symbol which is the same as the latent trace at that time (with Gaussian noise). Next, a transition is probabilistically made to a second latent time point (constrained to be further ahead in time than the first one, but not too far ahead), and again, a symbol is emitted which is Gaussian in the latent trace at that latent time. This is repeated until an entire observed time series has been generated. However, since we want to also scale the data, while aligning it, we augment the 'latent time states' by 'latent scale states' (shown as dashed red lines). So rather than emitting a symbol Gaussian in the latent trace value, a symbol is emitted which is Gaussian in the latent trace value *times a scale factor*; instead of transition from one latent time to the next, a transition is made from a (time,scale) state, to another (time',scale') state.

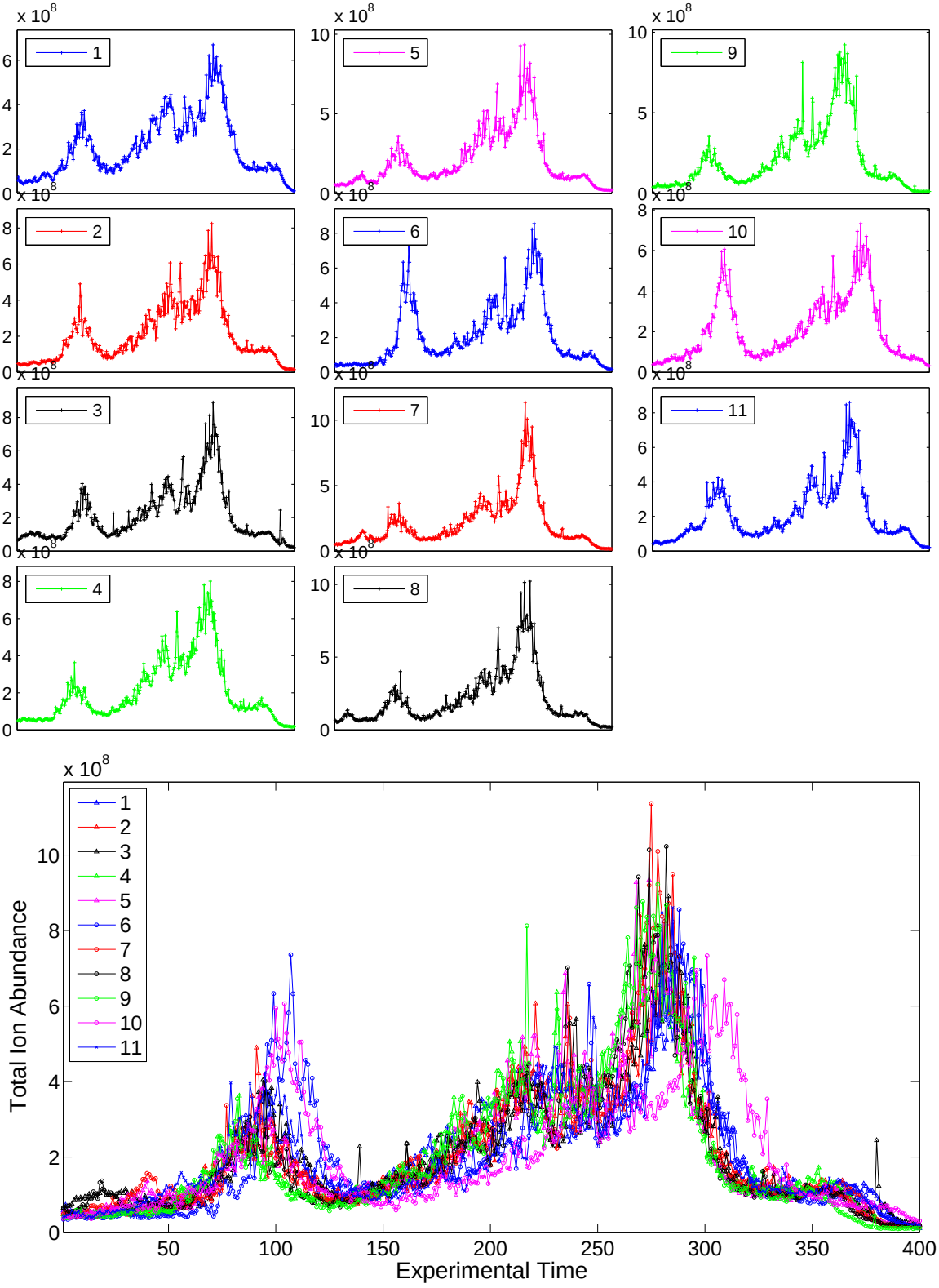
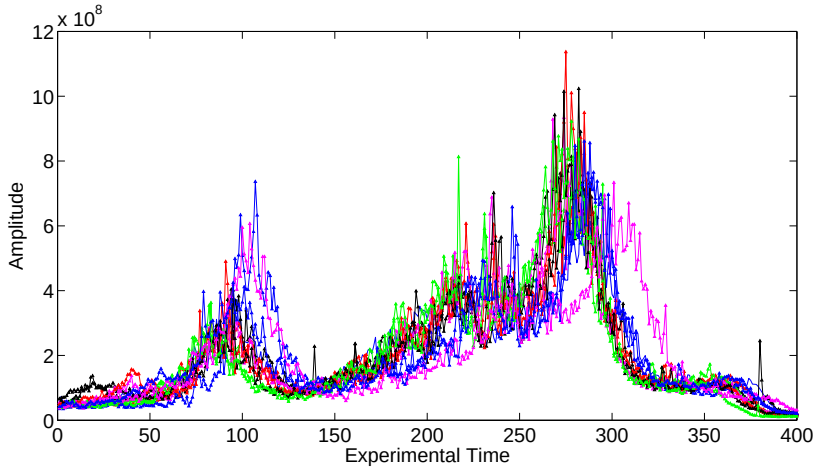
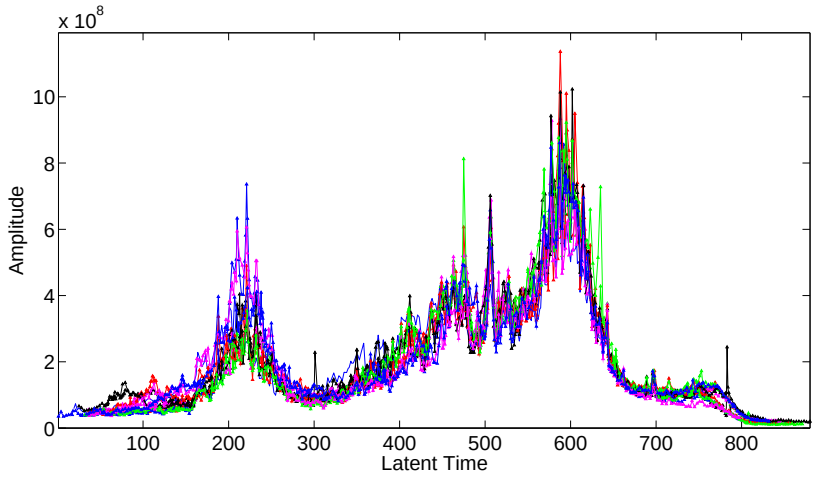


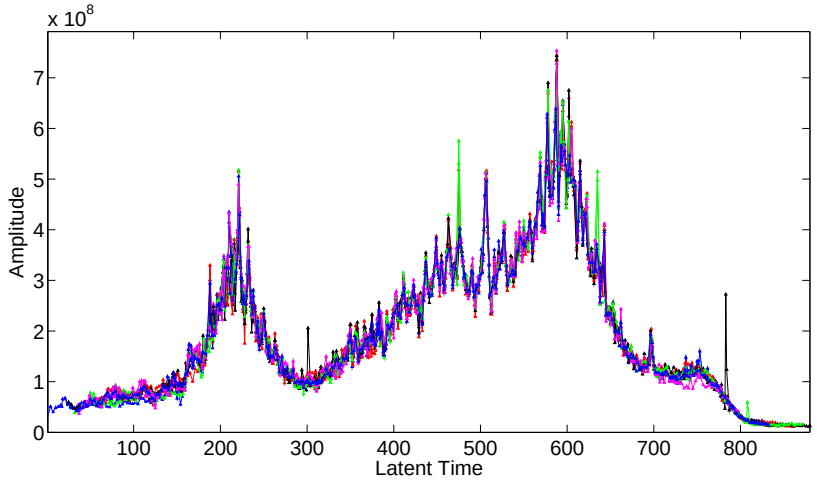
Figure 4.3: The raw, pre-processed LC-MS data set, shown individually in the top figure, and superimposed in the bottom figure. (It is the TIC of the data shown, which is the original LC-MS data reduced to be scalar time series by adding together the ion abundance at all *m/z* values (feature values) to create a single, scalar feature.



a)



b)



c)

Figure 4.4: a) the TICs of the raw, pre-processed, but unaligned data, b) EM-CPM-aligned, but not scaled version of this data, and c) EM-CPM-aligned and version of the data. Note that b) results from training the EM-CPM with both alignments and scaling, but shows only the alignment portion.)

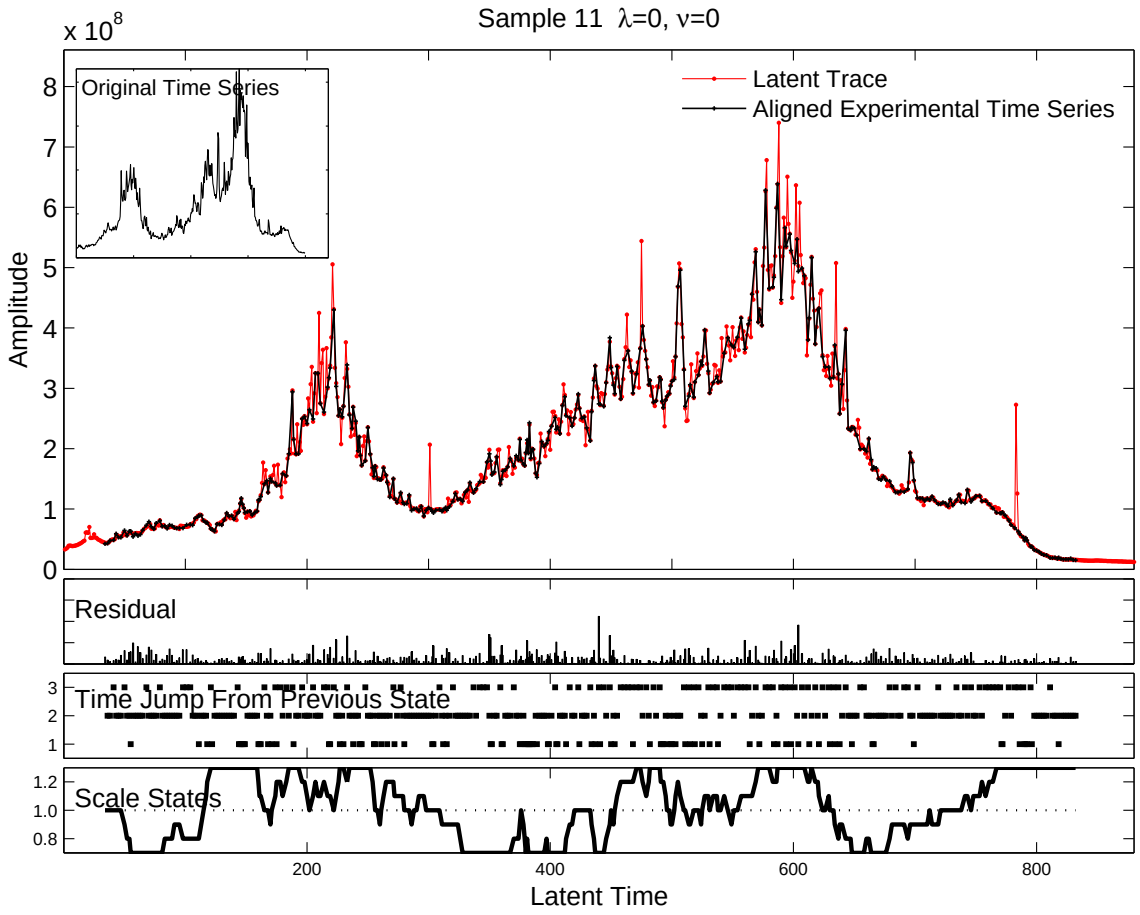
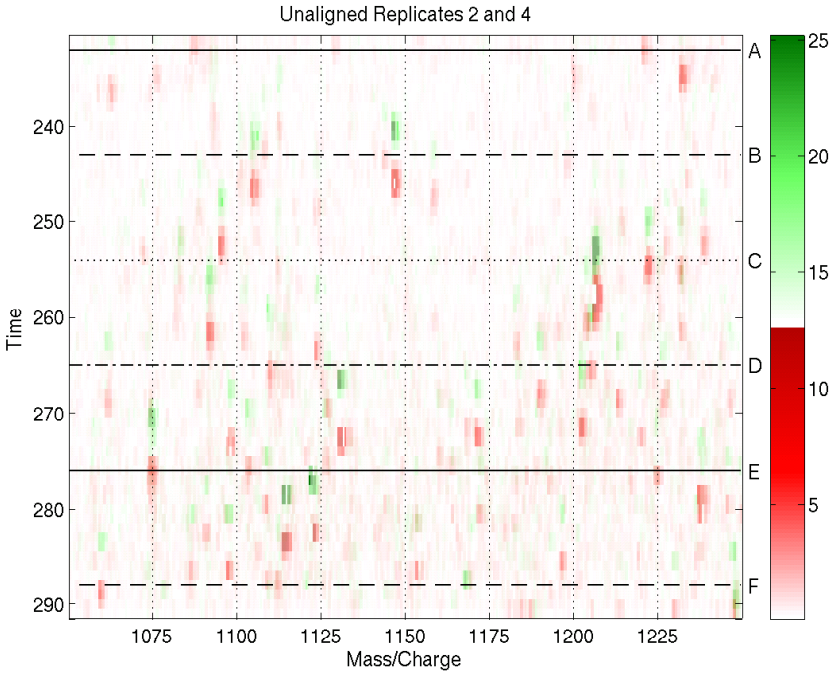
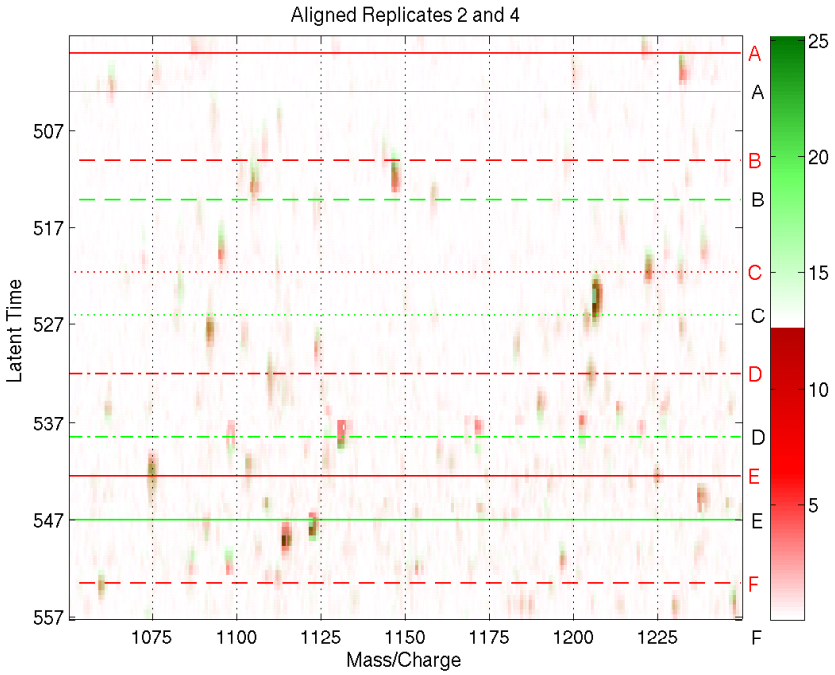


Figure 4.5: The 11^{th} TIC replicate Viterbi aligned to the learned latent trace (inset shows the original, unaligned). Below are three strips showing, from top-to-bottom, i) the error residual, ii) the number of time states moved between every two states in the Viterbi alignment (for example, a mark at latent time 400 with a time jump of 3 indicates that the previously used latent time was 400-3=397), and iii) the local scaling applied at each point in the alignment (vertical scale is in log space). The 7 scale states are evenly space in $\log$ space, between a scaling factor of $2^{-0.45} = 0.73$ and $2^{0.45} = 1.336$. The global scaling factor, $u^k = 0.987$



a)



b)

Figure 4.6: Zoom in of observed time series #2 (in red) and #4 (in green) superimposed. a) shows the pre-processed, but unaligned data, while b) shows the unrolled alignment. Marker lines labeled A to F show how observed time in a) was mapped (non-linearly) to latent time using the Viterbi alignment.

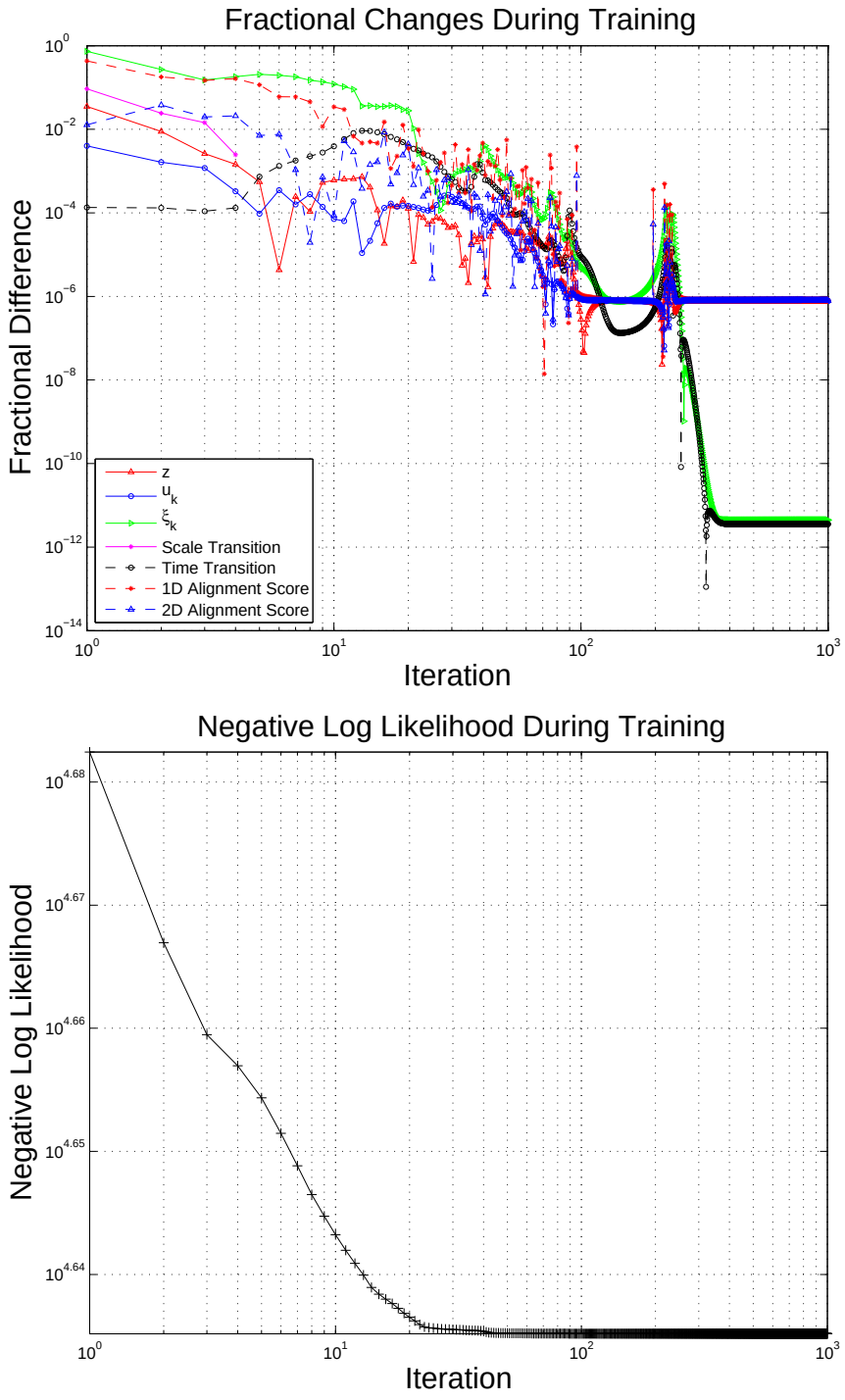


Figure 4.7: Top: Fractional change in paramaters, and 1D, 2D scores over 1000 iterations of EM. For groups of parameters such as u_k, ξ^k, κ_v^k, z, the mean fractional change is shown. Bottom: how the negative log likelihood changes during the same period.

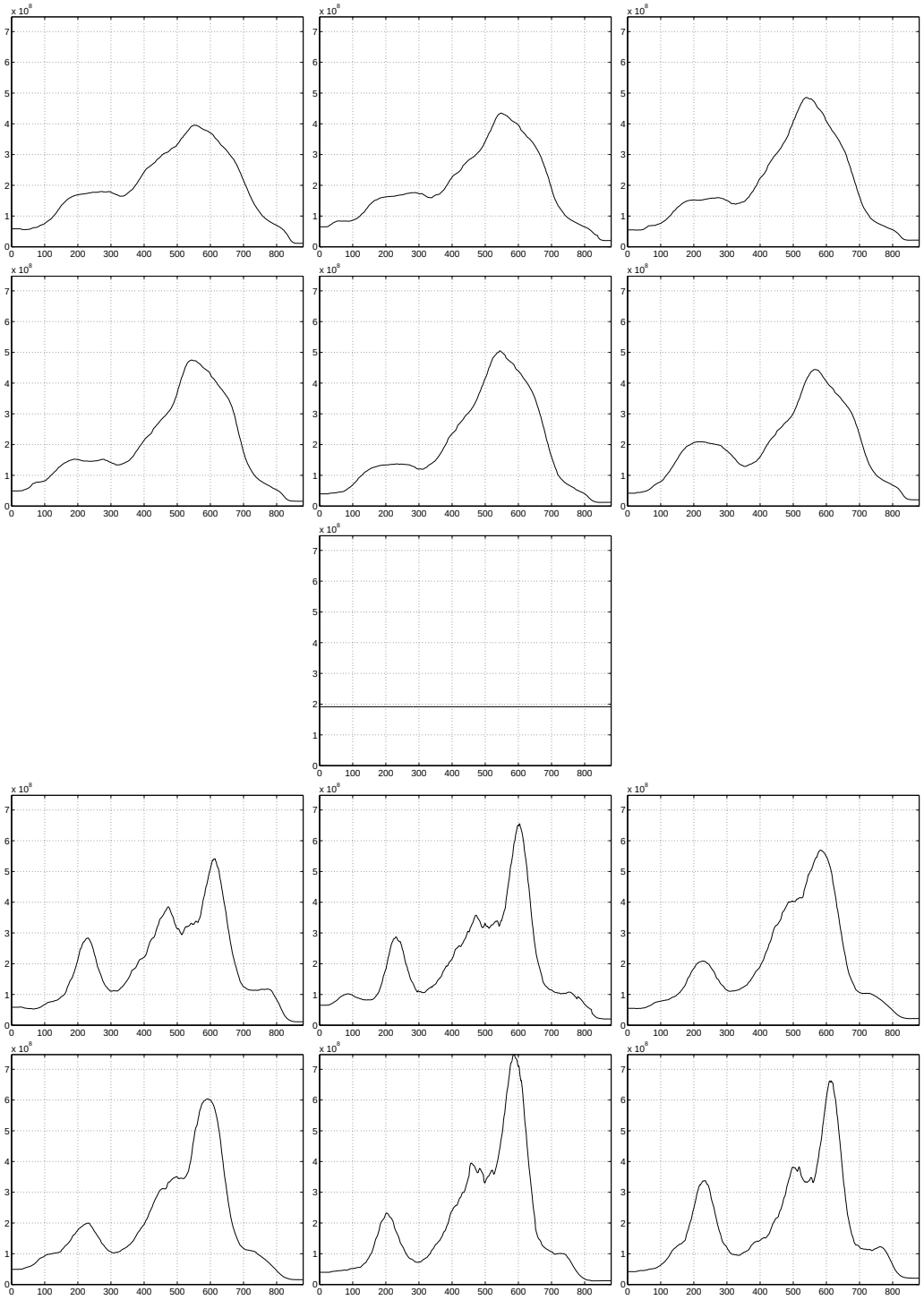


Figure 4.8: Initial traces used for each of 13 EM runs on 6 TICs of the LC-MS data set. The first six show the 20%smoothing initializations, while the last 6 show the 10%smoothing initializations.

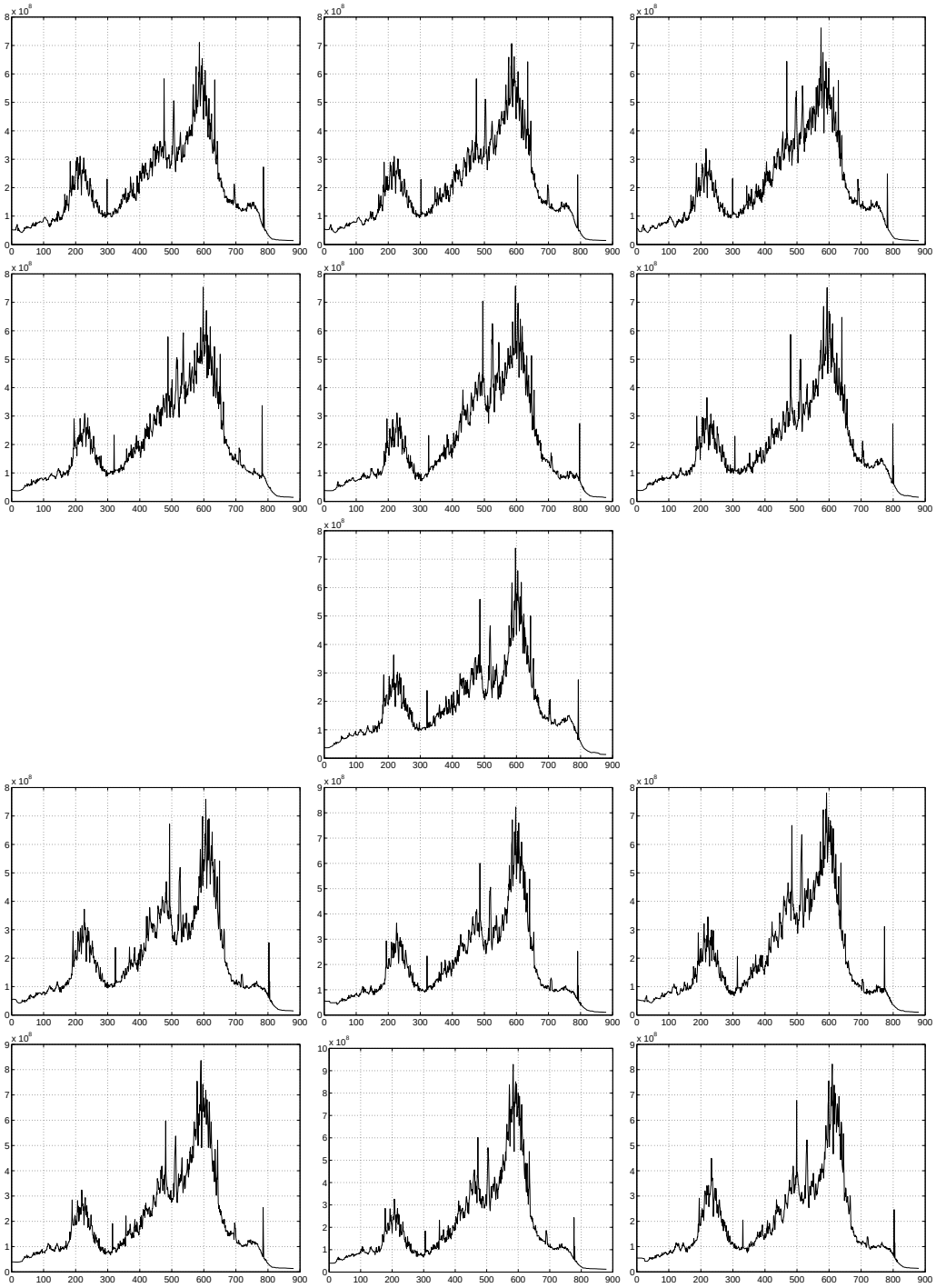


Figure 4.9: Final latent traces after EM, for each of the initializations shown in Figure 4.8

.

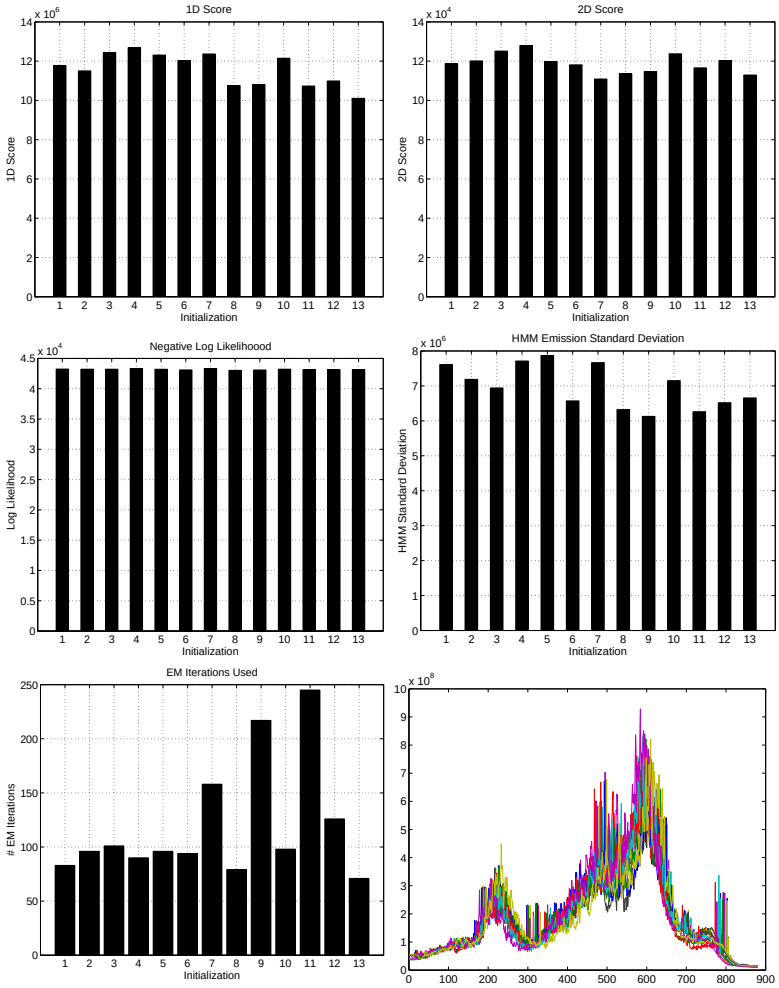


Figure 4.10: Comparison of various values across 13 initializations. The x-axis labels these initializations 1-13. These are respectively (1-6) each of the six traces with 20% smoothing, followed by (7) the flat trace, followed by (8-13) six traces with 10% smoothing. The bottom-right plot shows all of the learned traces superimposed.

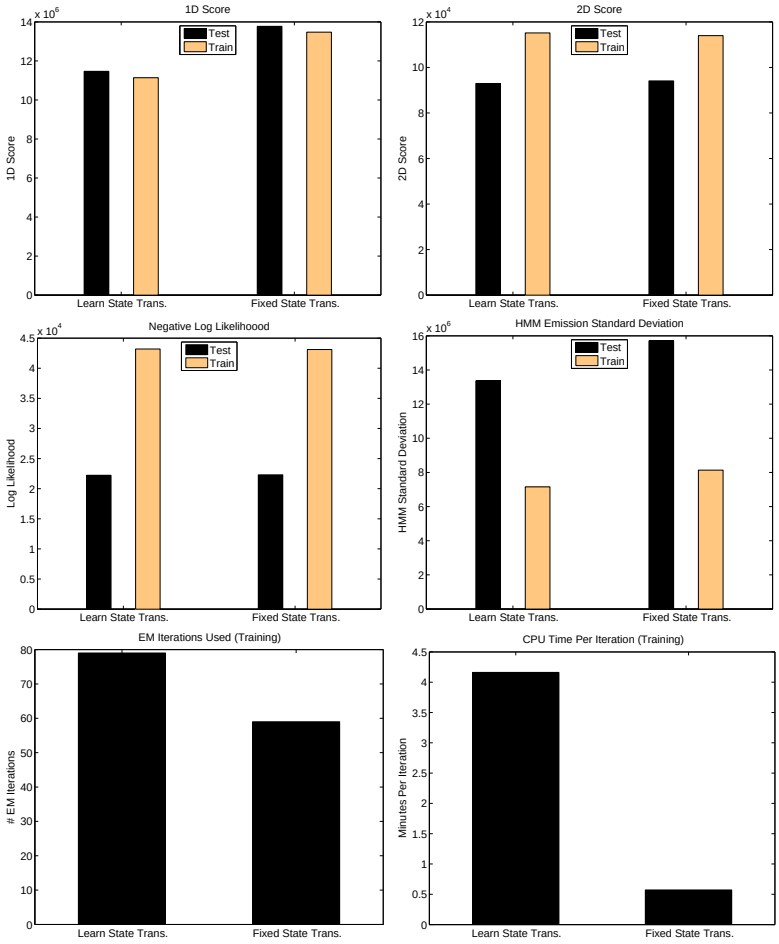


Figure 4.11: Comparison between learning the state transition parameters, or not learning them. The quality of alignments between these is comparable, but the running time is much faster if we omit learning them, reducing each iteration from four minutes to half a minute on this data set. Note that 1D/2D scores are not comparable between hold out and training since different numbers of observed time series were in each, and this can not be accounted for in a straight-forward manner.

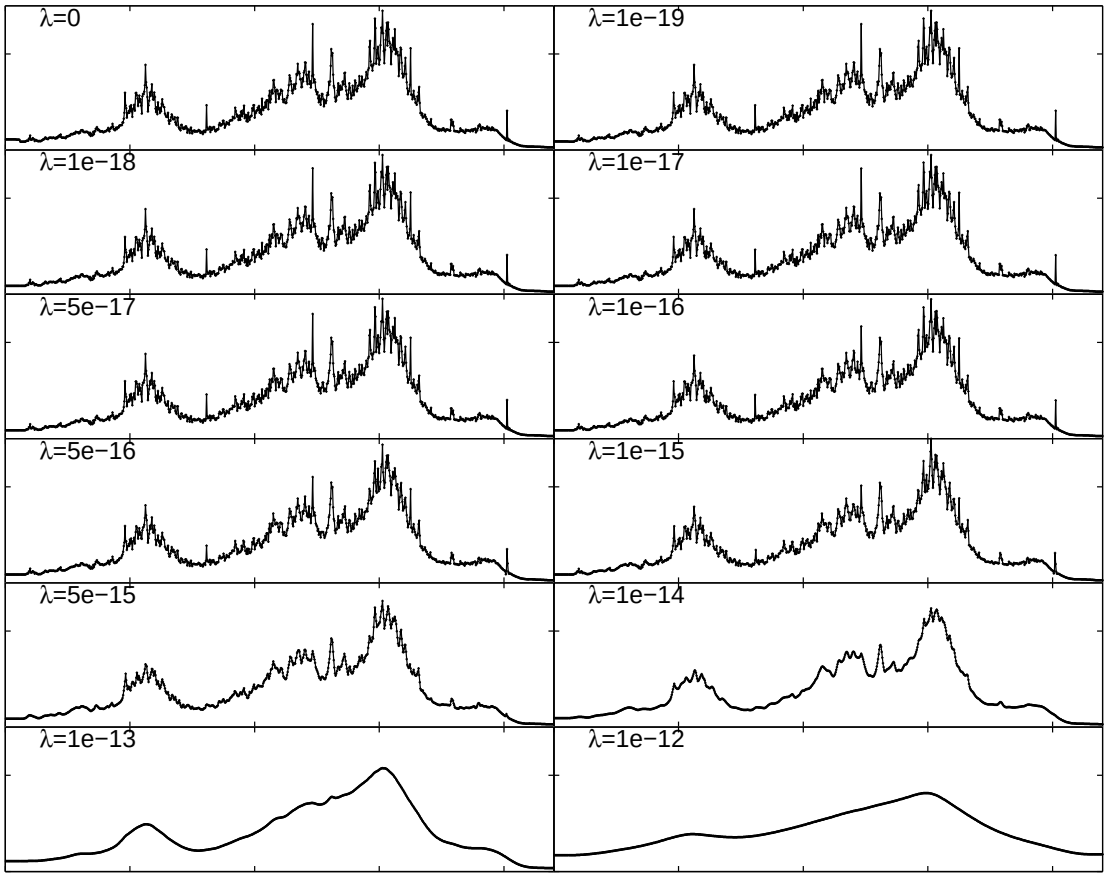


Figure 4.12: Latent traces recovered using different smoothing penalties on the latent trace (λ).

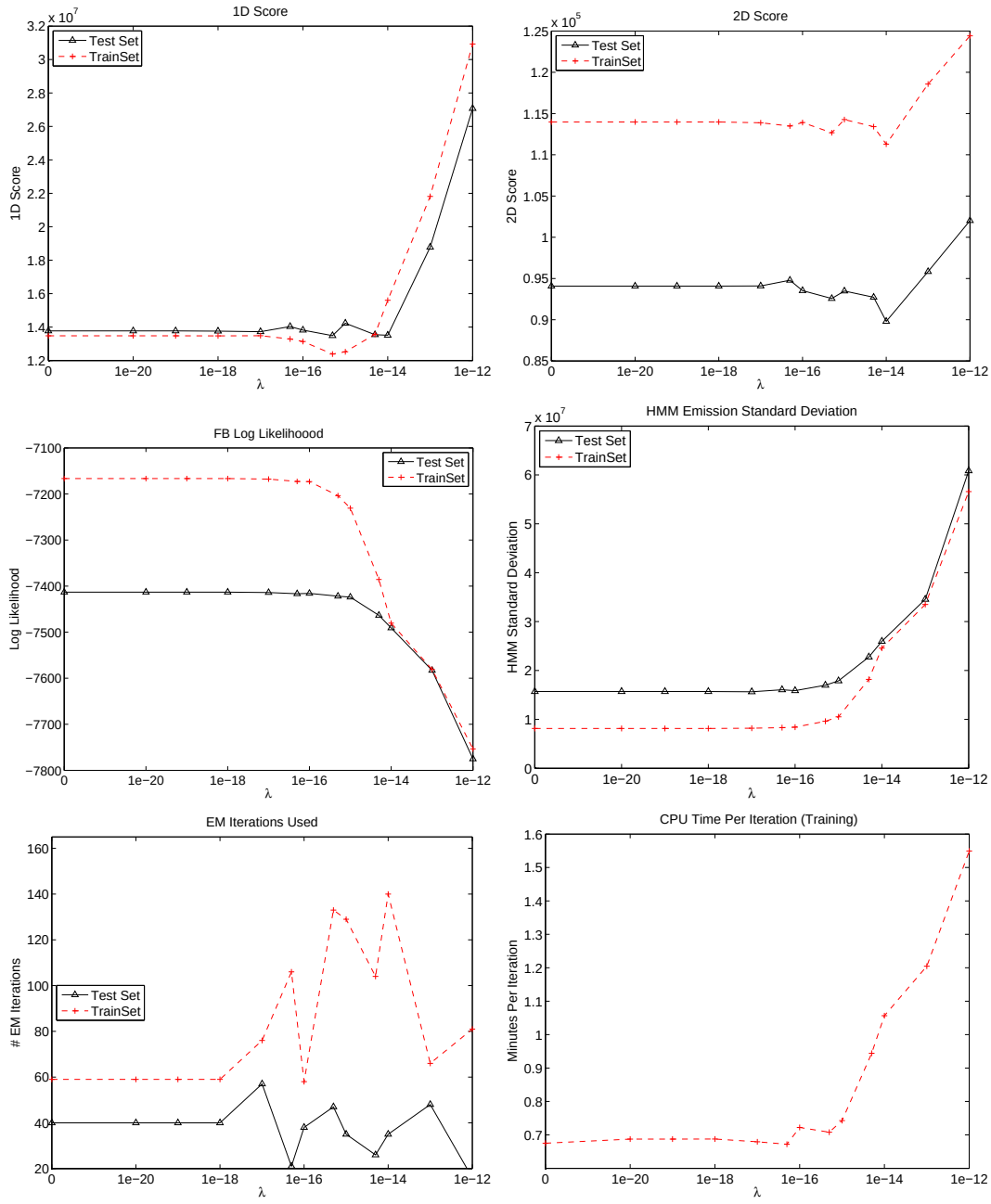


Figure 4.13: Comparison of results for different smoothing penalties used. 'FB Log Likelihood' denotes the portion of the likelihood calculated using the Forward-Backward algorithm, which contains the emission and transition probability portion of the likelihood. From the 1D score, the optimal amount of smoothing appears to be around $\lambda = 5 \times 10^{-16}$. However, looking at the log likelihood, no smoothing seems to be equally good if not better, while the 2D score appears to be best around $\lambda = 10^{-14}$.

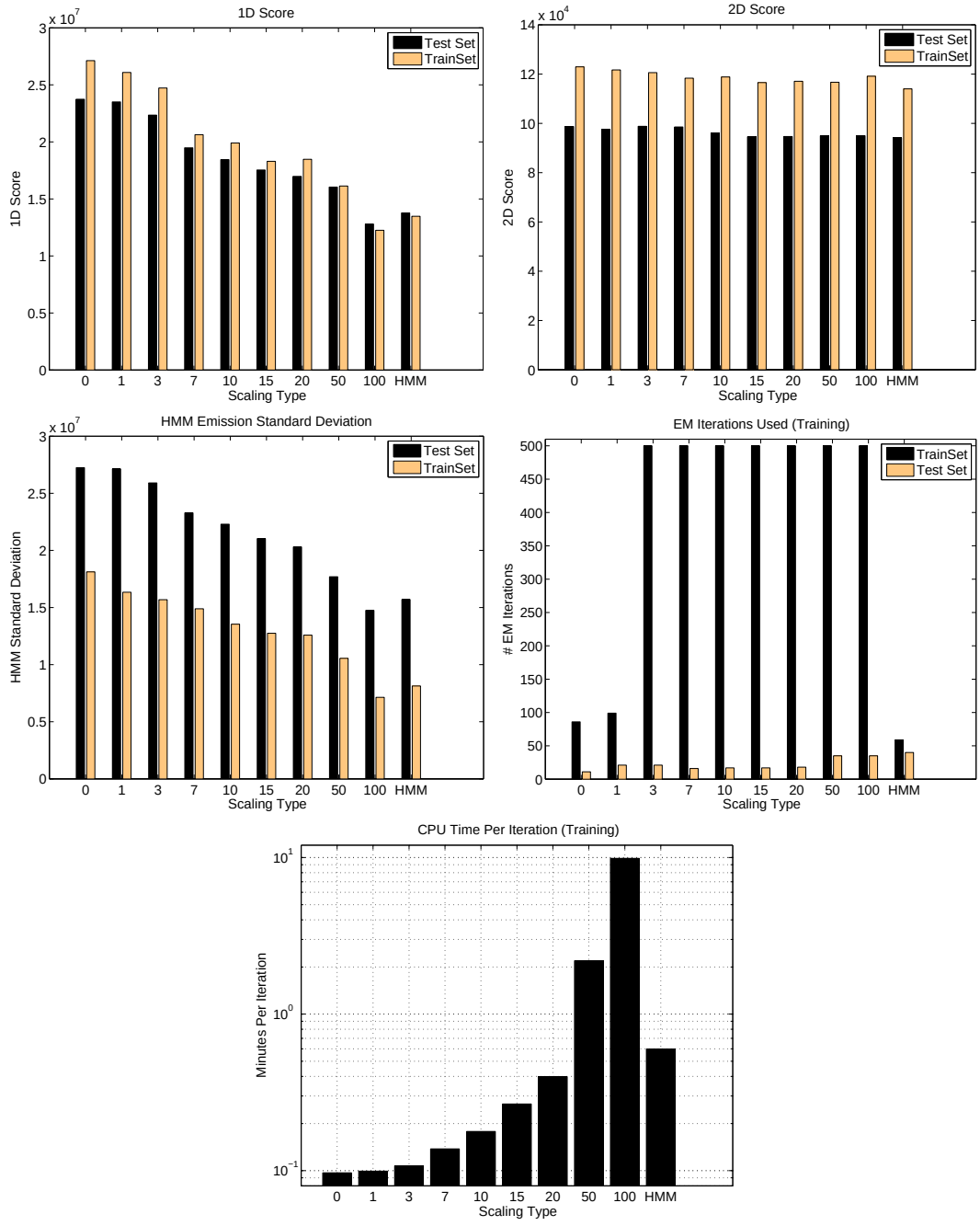


Figure 4.14: Comparison of various type of scaling. The x-axis labels denote, respective from left to right: (0) using no scaling whatsoever (either with HMM states or otherwise), followed by use of various numbers of control points (1, 3, 7, 10, 15, 20, 50, 100), labelled with these numbers on the axis, and finally ('HMM') which uses 7 evenly space scale states in the CPM hidden states along with a single global scaling factor per trace.

4.9.7 Examining the Posterior

Though we have been content to use Viterbi (MAP) alignments, we now briefly explore other parts of the posterior. One can imagine the possibility that the posterior contains a set of similar alignments, each with a posterior probability less than the MAP alignment, but which are actually better alignments than the MAP alignment and together have higher probability mass. In other words, the MAP alignment may have the highest probability precisely because it is unstable (*i.e.*, a small change away from it causes a large drop in likelihood) which would suggest the MAP alignment is actually overfitting the model. Thus a single sample from the posterior may in practice be better than the MAP estimate, or better yet, combining many samples from the posterior, as in a true Bayesian paradigm, although we do not examine this latter approach in this chapter.

Using our experimental run from Section 4.9.2 where we trained the EM-CPM on all 11 TICs of the LC-MS data set, with no latent trace smoothing, we gathered 500 samples from the posterior, as well as a Viterbi alignment, and compared them, as shown in Figure 4.15. We see that TIC 11 (and TIC 1, though less so) have more peaked posteriors than the others, since their posterior sample probabilities are closer to their respective MAP probabilities, and very narrowly distributed. Interestingly, when examining the 1D and 2D scores for the posterior samples versus the MAP state sequence, we see that the MAP 1D score lies in the middle of the posterior sample scores, suggesting that the Viterbi and the posterior samples are comparable to one another in aligning in the TIC space. However, the MAP 2D score lies systematically above all of the posterior sample scores (*i.e.*, is systematically worse), suggesting that the TIC alignment may be overfitting in some sense. Note that if we were going to use samples from the posterior for alignment, it would make more sense to average over them in some way, rather than looking just at individual samples. An exploration of how such an approach compares to Viterbi alignment would also be useful.

4.9.8 Vector Time Series Alignment

Here we explore how using more *m/z* bins with the LC-MS data set (rather than just one bin – the TIC) changes the alignments, and also the running time. Results are shown in Figure 4.16 for use of the TIC (1 bin), as well as 2, 4, 8 16, 20 and 24 bins.

We do not show the emission noise, the 1D score, nor the log likelihood, since do not lend themselves to comparison in this type of experiment. And of course, the 2D score, though it can be compared, will not get worse as more bins are used (aside from small experimental variations) – which is not a reflection of 'better' alignment because of the possibility of overfitting. Convergence does not appear to be systematically affected by use of different number of bins.

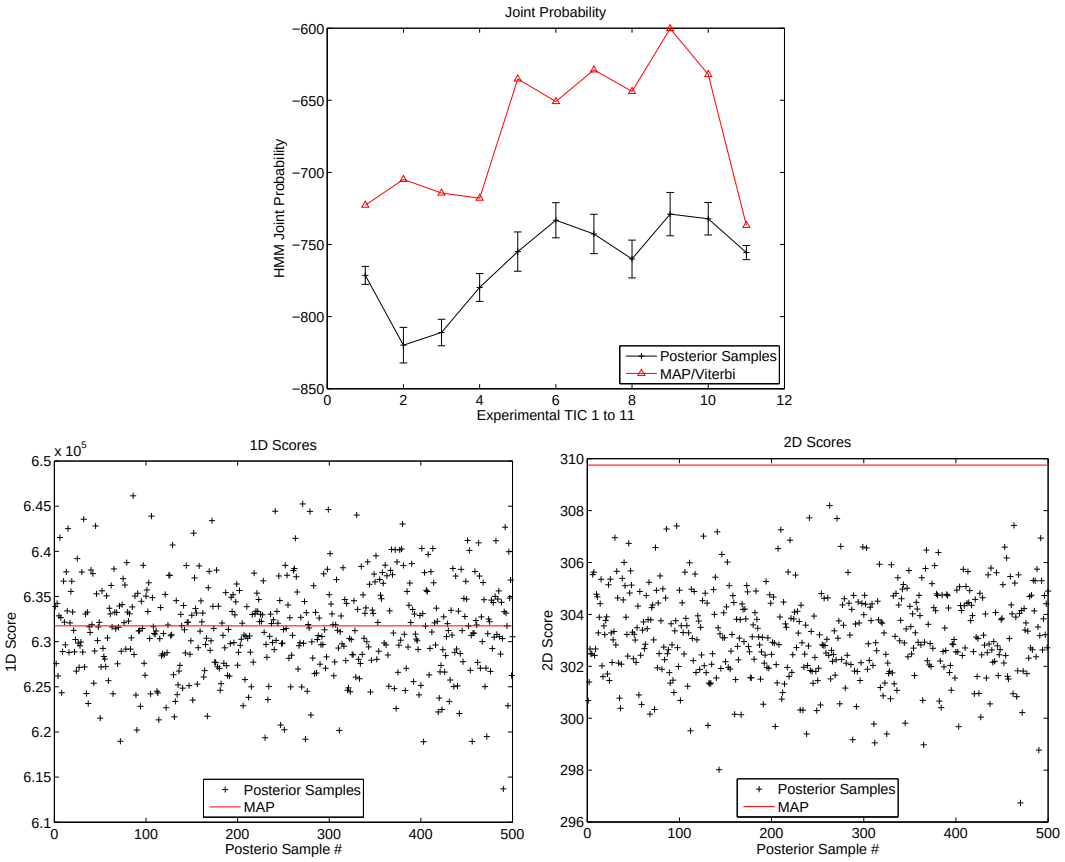


Figure 4.15: Comparison between using a MAP/Viterbi alignment and sampling from the posterior. 500 samples were drawn from the posterior, and the HMM portion of the log likelihood was calculated for each of them, seperately for each of the 11 observed time series. The top plot shows the resulting mean $\pm$ 1 standard deviation of this log likelihood, along with the MAP log likelihoods, while the bottom two plots show the 1D and 2D scores.

Obviously the benefit of using more bins is heavily dependent on the data set used.

Without a gold standard, it is difficult to say how much we gain or lose by using more or fewer bins for alignment, but certainly use of more bins increases the computation time (approximately linearly). Later, in Chapter 6, we will see that for this type of LC-MS data, in the context of a spike-in experiment with known ground truth, that using just one bin (TIC) does not do a good job, and that using more does much better, although beyond 4 bins the returns are diminishing.

4.9.9 Comparison to Dynamic Time Warping-Like Algorithms

Many alignment algorithms, especially those used in the LC community, are heavily based in one way or another on Dynamic Time Warping (DTW – see Chapter 2). Recall that DTW

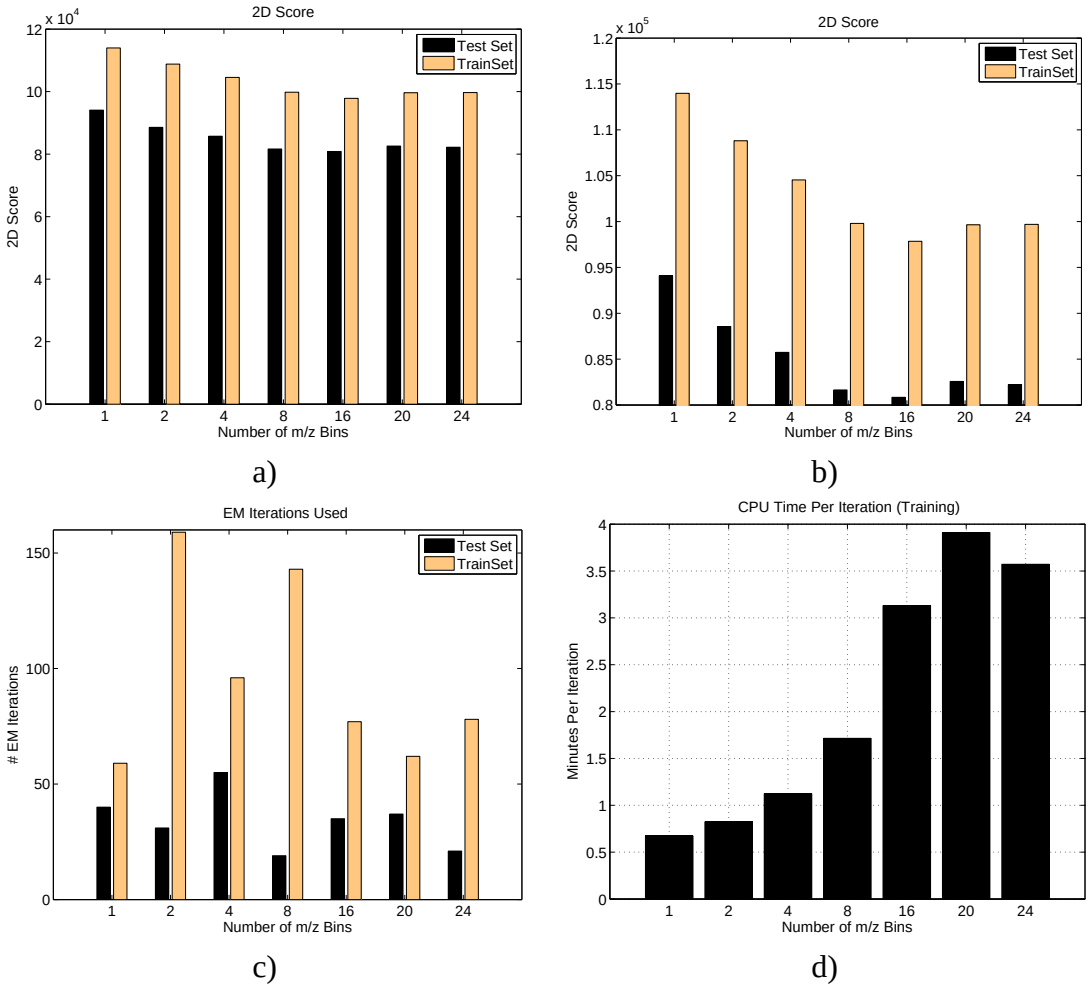


Figure 4.16: Comparison using different number of *m/z* bins during alignment. b) is a y-axis zoom in of a). Note that CPU time per iteration is over all bins.

warps two time series to match each other, using dynamic programming to find the best value of a pre-specified cost function. DTW algorithms vary in terms of what cost function they use (*e.g.*, correlation, Euclidean distance), and in terms of how each is regularized (*e.g.*, constraining the maximal amount of warping by constraining the slope of the path in the DP matrix; allowing only so many 'horizontal' or 'vertical' steps in the path; not allowing the path to go outside of a pre-specified 'bandwidth' around the neutral warp; allowing only certain key nodes to be transformed non-linearly, with all time points in between key being linearly warped (called COW); weighting the length of the path in different ways in the cost function [81]).

Note that for algorithms such as DTW and COW which require a template to be specified *a priori*, that various multi-alignment schemes could be used. One could choose one sample as a reference, and align all the others to that one, as we do here. Alternatively, one could use

DTW to find the distance between every two pairs of time series, and then successivly align and merge in a hierarchical fashion with the closest times series first (something similar to this is done in [74]) although it is not clear that this would provide improved alignment. Perhaps the best strategy would be to use an EM-like iterative refinement (Procrustes fitting criterion [33]) in which one starts off with some template, aligns the data to this template, averages the aligned samples to form a new template, and continues – but which has no guarantee of convergence or improvement. In any case, each of these strategies is *ad hoc* and would need to be explored experimentally.

We do not here perform a formal, quantitative head-to-head comparison of different alignment algorithms, only because it is not clear how to do this without an unbiased ground truth. One might consider trying to devise a ground truth data set by way of some laboratory procedure over which one had control – but this is not within our grasp for this thesis and may be difficult if not impossible to do well. While some researchers advocate the use of synthetic data sets, use of such data is limited, since it is usually impossible to be sure one is generating a data set which is representative of the problems of interest. Furthermore, if one generates a data set from a probabilistic, generative model, and then uses this data to assess various algorithms, that same probabilistic model will have an unfair advantage. Experiments with synthetic data typically reveal little beyond whether or not the model is behaving consistently.

Thus a direct, quantitative, formal evaluation of a solution to the alignment problem is difficult, if not impossible. As a proxy to an alignment ground truth, one could use a desired end goal as an indication of how well alignment is being performed, such as if one had a synthetic spike-in experiment. In such a case, one could try various alignment schemes, followed by identical post-processing to uncover known spiked differences, and see which alignment method worked best. We defer such a comparison to Chapter 6 where we concentrate on analyzing just such an experiment.

One might hope that using the 1D or 2D scores introduced earlier, or some other similar measure, could help us in our quest for formal evaluation, however, these measures do not account for overfitting of the data, and hence do not accurately measure quality of alignment for the purposes of comparing different algorithms. We nevertheless calculate them here simply to emphasize this point. One might also believe that it would be possible to use some sort of hold out set, as we did when exploring the EM-CPM – however, this is also prone to overfitting, since the hold out set must still make use of the alignment algorithm. Alas, all we can really do, without appealing to proxy measures of quality, is to visually look at the results of different algorithms, as in Figure 4.17, where the 11 aligned TIC LC-MS time series are shown, aligned, using Dynamic Time Warping [81] with different regularization settings, COW (Correlated Optimized Warping) [61, 81], and the EM-CPM. DTW was run with a ‘maximum bandwidth’ of

10% and (a) no contraints on the path slope, (b) with minimum/maximum slopes of 0.80/1.25, (c) with minimum/maximum slopes of 0.33/3, while COW was run with (d) 13 segments each allowed any amount of warping. Finally, (e) shows the results of using the EM-CPM. The COW and DTW code used was that of Giorgio Tomasi as described in [81]. We arbitrarily used the first experimental time series as a reference template for DTW and COW. Later, in Chapter 6, we show that in fact DTW is very sensitive to the choice of template, and that the quality of the resulting alignment is equally sensitive.

Looking at the 1D scores reported at the top of each plot in Figure 4.17, we see that the lowest ('best') score is that of the EM-CPM with scaling, followed by uncontrained DTW. Unconstrained DTW, however, has many 'flats', which are surely not representative of the underlying data, and hence this algorithm can be said to be overfitting. Whether the EM-CPM is overfitting or not is difficult to assess. Note that algorithms which simultaneously align all data, such as the CPM, are inherently more robust to overfitting than those which use a greedy approach, such as DTW. With the greedy approach, only two time series at a time are coerced to agree with one another, whereas with simultaneous alignment, all are coerced at once. Constrained DTW and COW visually do much better than unconstrained DTW, and seem comparable to the EM-CPM without any scaling, while visually, the EM-CPM with scaling appears to work quite nicely.

Despite the obvious problems in using a synthetic data set, we nevertheless felt this could serve two purposes: 1) to demonstrate consistency of the model, on what we perceive to be realistic looking data, and 2) to allow us to make explicit the potential power of our approach, especially as compared to simpler DTW-like algorithms which do not allow scaling while aligning. We thus synthesized a data set, and visually compared the same algorithms used in Figure 4.17, against each other, as well as to the known ground truth.

Our synthetic data set contains ten 'observed' time series belonging to a single class, each generated from our EM-CPM, using a latent trace equal to the average of the 11 EM-CPM-aligned LC-MS TICs we have been using throughout this section (using $\lambda = 0$). We set the HMM emission noise to be 5% of the mean latent trace value; the probability of staying in the same scale state was 80%, and the probability of moving 1,2, or 3 latent times states ahead were equal, and all others disallowed. The resulting data are shown in Figure 4.18, while the latent trace can be seen in Figure 4.21. We used the EM-CPM with $\lambda = 0$, not learning the state transition probabilities (these were fixed at a 90% chance of staying in the same scale state, and of 30%, 40% and 30% chance of moving respectively 1,2, and 3 time states ahead. We set all $u^k = 1$ in the data set, but allowed these to be learned during training. Figure **??** shows the aligned data from each of the EM-CPM, DTW, and COW, while Figure 4.20 shows the average of the aligned data from using each method. Figure 4.21 shows the recovered latent

trace and the ground truth latent trace.

There are a number of observations we can make from these figures. Overall, the EM-CPM nicely recovers the alignment, and the latent trace. However, the scaling on the re-covered latent trace is a bit different from the ground truth. For example, the maximum value in the former is just under 6×10^8, while on the latter, it is 7×10^8. Additionally, the recovered latent trace is more jagged, perhaps having overfit the data slightly, owing to limited data and use of $\lambda = 0$. Also notice that the relative timing of the aligned features is slightly different between the ground truth and recovered latent traces. These problems might disappear in the face of much more data, or could be inherent degeneracies in the model, or training may have arrived at a sub-optimal local maximum. More importantly, the alignment problem can be said to be ill-posed – even with a known ground truth, an alignment solution can be said to require only putting the proper points across observed time series into correspondance with each other, but not that relative time within a time series be mapped in any particular way, although some ways are clearly more pleasing than others.

It is clear from Figures 4.19 and 4.20 that unconstrained DTW does not do a good job. Furthermore, without the ability to do scaling, DTW and COW can never recover as detailed and accurate of a summary representation of the data. Whether this is because their alignment suffers because of their inability to do scaling with alignment, or whether the data need only be scaled afterwards is not clear, but in any case, the requirement for scaling certainly seems to be prevalent in LC-MS data, and hence a model to address this issue is needed; the EM-CPM, at least in this synthetic context, and in the former real context, appears to be doing quite well. Note that the learned global scaling parameters (u^k) ranged from 0.96 to 1.08, with a mean of 1. Thus these do not exactly match the ground truth (all 1), but are certainly very close.

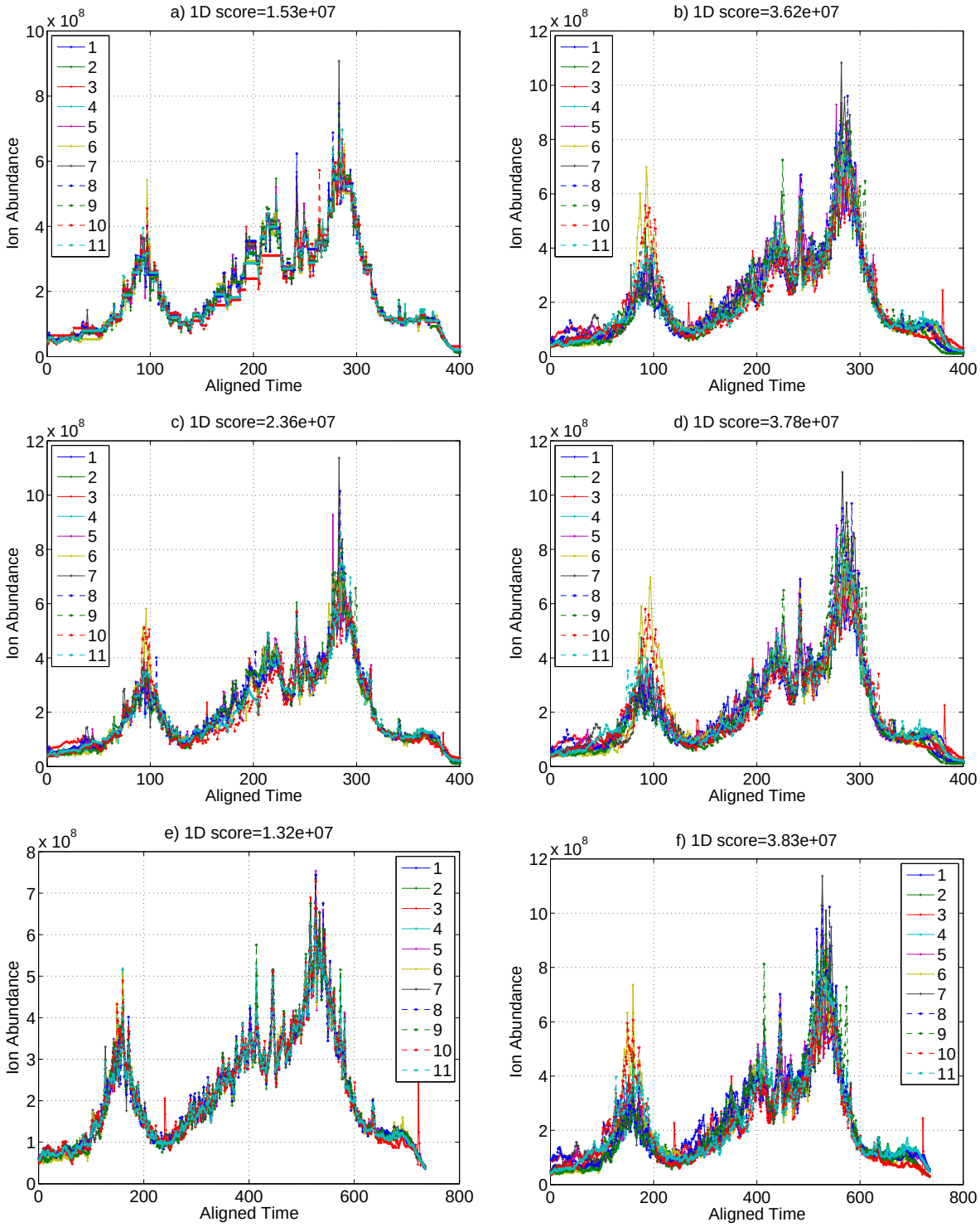


Figure 4.17: Eleven aligned LC-MS TICs, using a) DTW with no slope constraints, b) DTW with min/max slope of 0.80/1.25, c) DTW with min/max slope of 0.33/3, d) COW using 13 segments, e) the EM-CPM showing scaling and alignment, and f) the EM-CPM aligned data, but without showing the scaling correction. Note that the red outlier spikes, most visible in e) and f), around times 250 and 750 result from outlier spikes in the third data set, and are not an artifact of any of the algorithms.

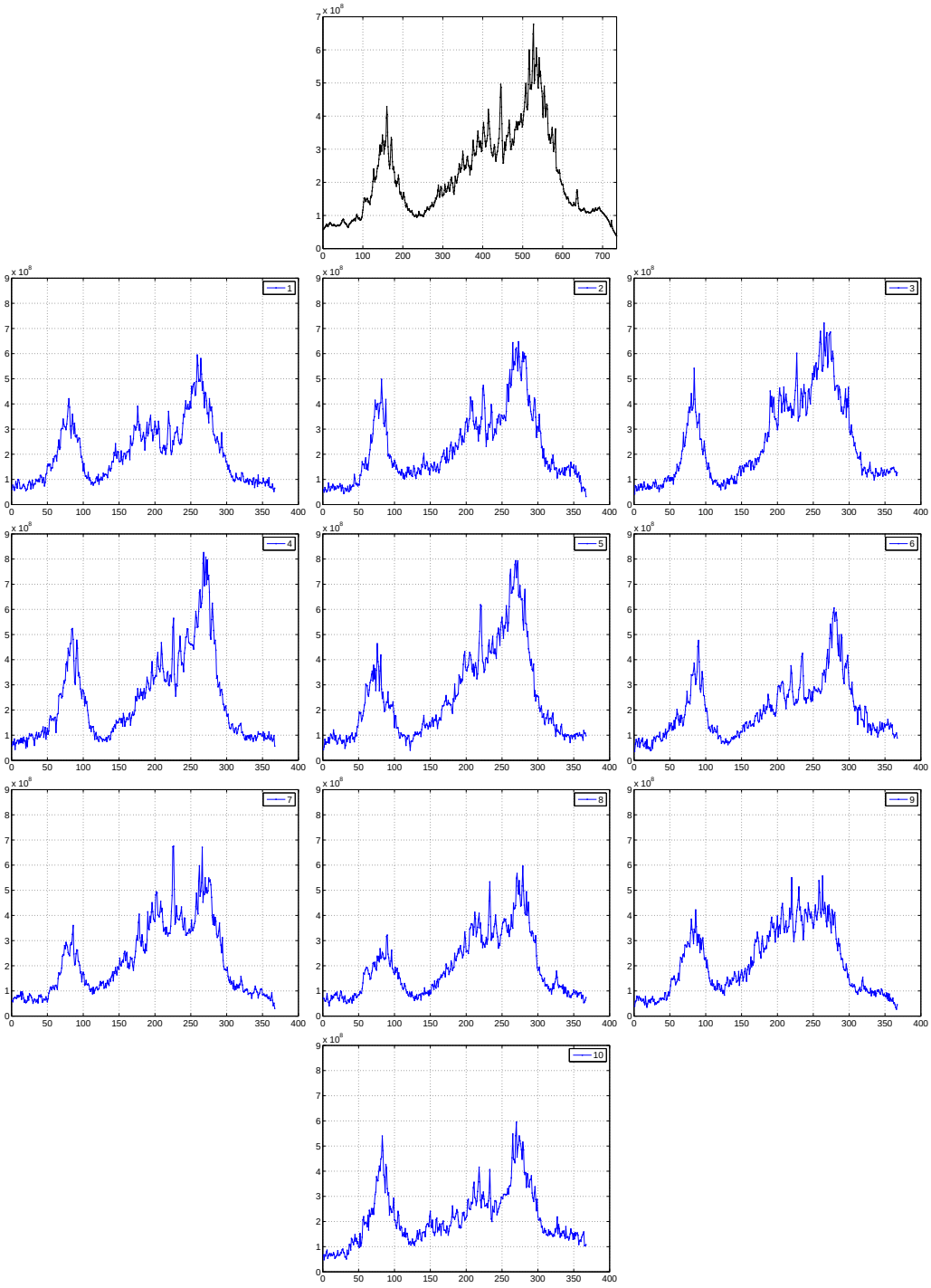


Figure 4.18: Synthetic data set (blue) based on a realistic LC-MS TIC latent trace (black). Note that the vertical scale is different on the latent trace plot.

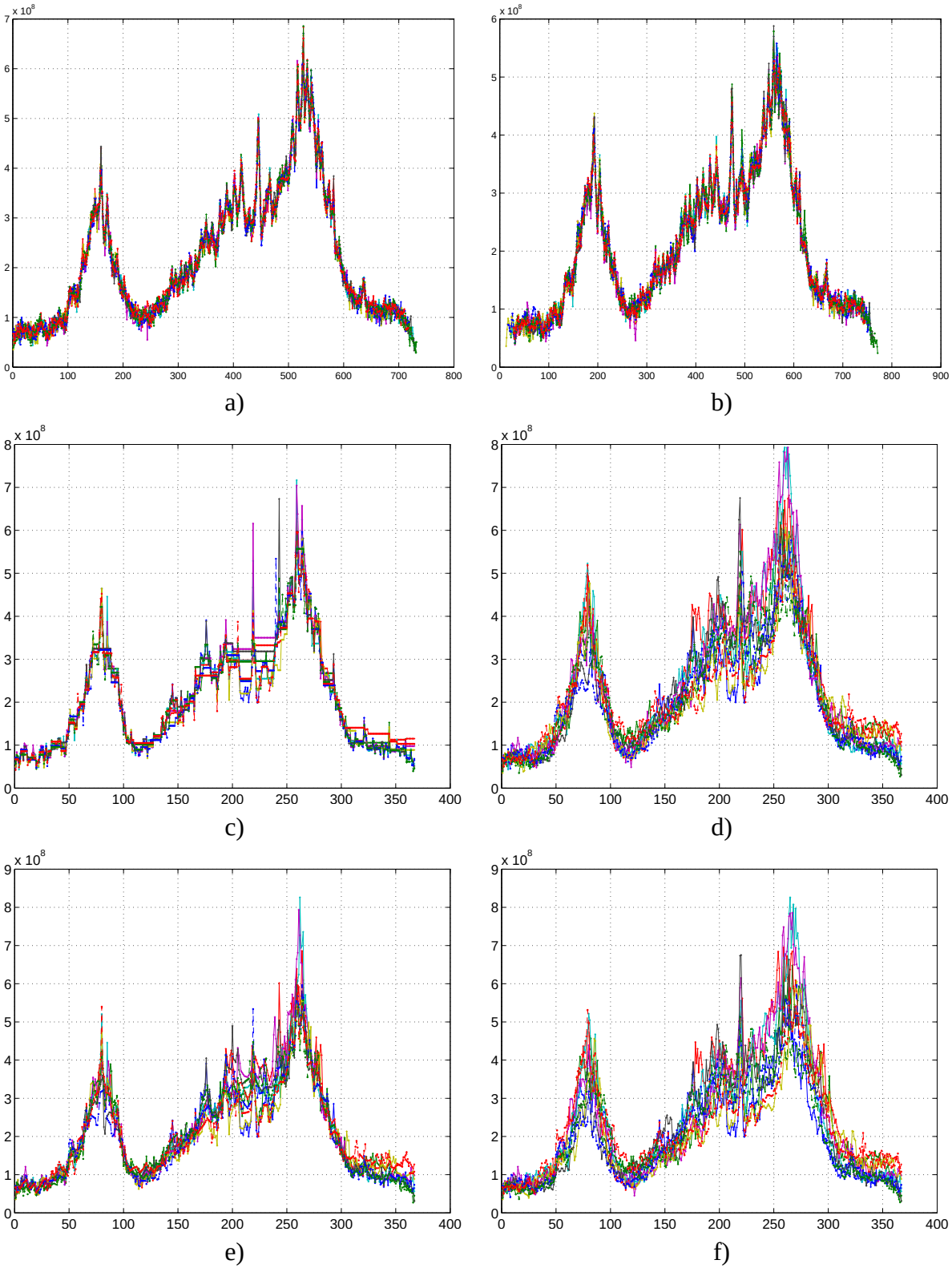


Figure 4.19: The aligned synthetic data from a) the ground truth (with scaling), b) EM-CPM (with scaling), c) unconstrained DTW, d) DTW with min/max slope of 0.80/1.25, e) DTW with min/max slope of 0.33/3, f) COW with 12 segments.

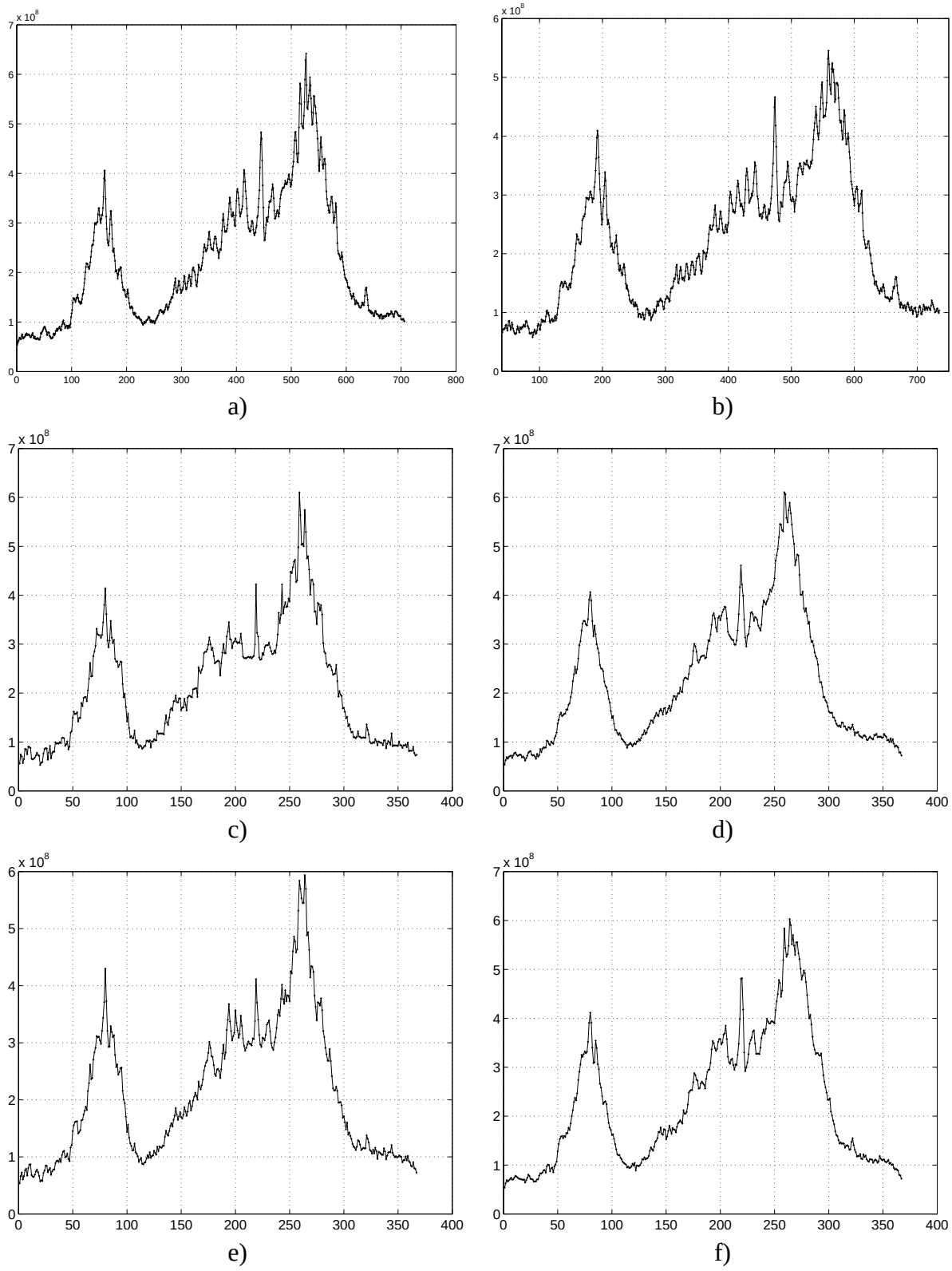


Figure 4.20: The average of the aligned synthetic data from a) the ground truth (with scaling), b) EM-CPM (with scaling), c) unconstrained DTW, d) DTW with min/max slope of 0.80/1.25, e) DTW with min/max slope of 0.33/3, f) COW with 12 segments.

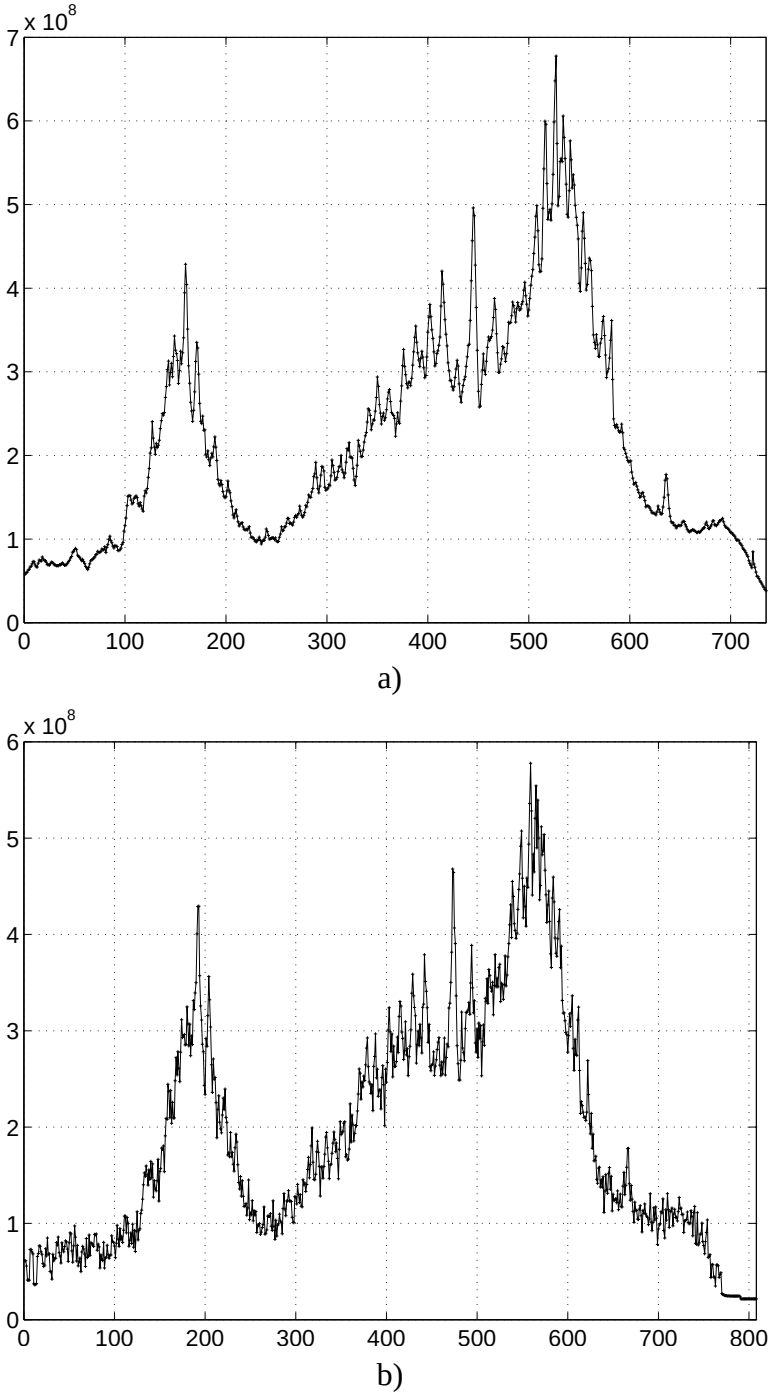


Figure 4.21: a) The latent trace used to generate the synthetic LC-MS TICs, b) the recovered latent trace using the EM-CPM with $\lambda = 0$. Note that time axes are not really comparable (both are in latent time).

Relationship between DTW-like algorithms and the EM-CPM

DTW can be thought of as a highly restricted version of the EM-CPM. If using DTW to align several observed time series using one as a reference, one could achieve an almost identical algorithm using a very simplified EM-CPM which requires no learning. One would set the latent trace to be one of the observed traces (*i.e.*, a reference template) without upsampling it first so that it was no more dense than the original time series. One would then hard-code the HMM emission variance, and all other parameters (this is akin to hard-coding the similarity measure in DTW). Then if one used no hidden scale states (or scaling spline), and used CPM Viterbi alignments, the result would be very close to that of DTW. If the same noise model was used, then in fact the results would be identical. Viewing the EM-CPM in this context, some strenghts of the model become apparent:

1. No template need be specified *a priori* with the EM-CPM – rather an appropriate template is learned from the data (and from all of the data at one time) in a principled, well-developed manner. Even if one considered the more sophisticated schemes of learning/choosing a DTW reference template, there is not one obvious or principled method.

2. Individual time points across different observed time series need not always be mapped to each other with the EM-CPM because of the high density latent trace, whereas DTW always maps a time point in one time series to that in another (COW, however, can map in between points, but at the cost of losing warping flexiblity). One could approximate this EM-CPM mapping flexibility using so-called 'super-alignment' DTW in which the observed time series are first upsampled by means of an interpolation scheme, and then the upsampled versions are aligned by way of DTW. However, this solution is quite *ad hoc*, though it might work well in some settings.

3. The distance function in in the EM-CPM need not be fully specified *a priori* – rather, we can learn suitable error function parameters from the data, whereas with DTW, one must specifiy all parameters in the cost function ahead of time.

4. Scaling of the data can be naturally incorporated in to the EM-CPM model, simultaneously with alignment. Additionally, other extensions, such as multi-class alignment can be naturally introduced in a principled manner, such as in Section 4.3, and more importantly, as in Chapter 5. These benefits clearly seperate the EM-CPM from DTW-like algorithms, and reflect the power and elegance inherent in probabilistic, generative models.

Additionally, in using the EM-CPM, information across an entire data set is leveraged, simultaneously, when performing alignment, rather than the standard incremental approach used with

DTW and COW which is inherently prone to overfitting. This means that we need not heuristically select a data order from which to progressively align, and thus need not incur any penalty for picking a sub-optimal order. In fact, even the optimal order of progressive alignment may be inferior to the EM-CPMs simultaneous alignment since all information is available at once with the EM-CPM.

Where HMMs have become the more sophisticated, higher quality, and principled approach to speech recognition problems, over the older, *ad hoc* DTW algorithms, similarly CPMs now provide a principled and higher quality approach to the problem of aligning sets of time series data (of course relying heavily on HMMs). That being said, DTW-like approaches are at least an order of magnitude faster than the EM-CPM (provided no iterative template learning is used with the DTW), and may very well, for certain problems requiring a faster solution, be the method of choice. DTW could also provide an excellent initialization for the EM-CPM.

4.9.10 Demonstration on Speech Data

Our initial motivation for the CPM was to align chromatography data, but here we use our model to align 10 speech signals, to provide a demonstration of the potential breadth of application of CPMs. However, we do not investigate use of the CPM with speech data in any great detail.

The 10 speech signals used are each an utterance of the same sentence spoken by a different speaker. The short-time energy (using a 30ms Hanning window) was computed every 8ms for each utterance and the resulting vectors were used as the input to CPM for alignment. The smoothing parameter λ was set to zero and we used a full set of scale states rather than a scaling spline. For comparison, we also performed a linear warping of time with an offset. (i.e. each signal was translated so as to start at the same time, and the length of each signal was stretched or compressed so as to each occupy the same amount of time). Figure 4.22 shows the successful alignment of the speech signals by CPM and also the (unsuccessful) linear warp. Audio for this example can be heard at `http://www.cs.toronto.edu/~jenn/phdThesis`.

4.10 Multi-Class Experiments

In this section we briefly explore use of the multi-class EM-CPM, although we do not do so extensively since we feel that this model is superceded by that in Chapter 5 which presents a fully Bayesian multi-class CPM model.

We demonstrate use of the multi-class EM-CPM on part of a NASA shuttle valve data [26] which measures valve solenoid current against time at a rate of 1ms per sample, with 1000

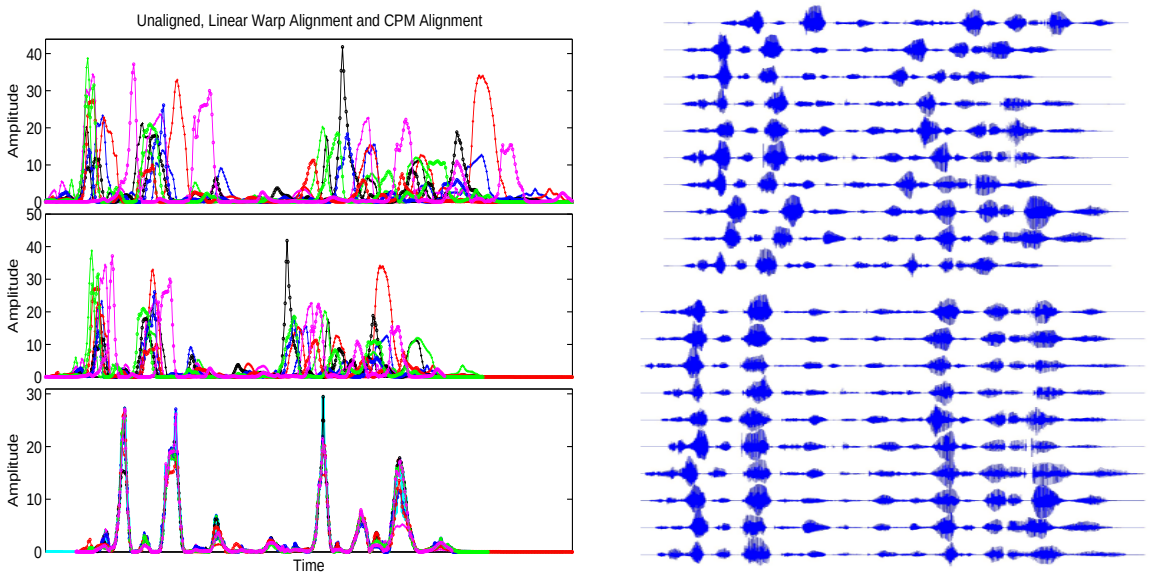


Figure 4.22: a) Top: ten replicate speech energy signals, Middle: same signals, aligned using a linear warp with an offset, Bottom: aligned with CPM (the learned latent trace is also shown in cyan). b) Speech waveforms corresponding to energy signals in a), Top: unaligned originals, Bottom: aligned using CPM.

samples per time series, for some 'normal' runs and some 'abnormal' runs. We subsampled the data by a factor of 7 in time since it was extremely dense and redundant. We separated the data in to two classes, using two and three members from each class to train, and then another two from each class to hold out, as shown in Figure 4.23. Although this data is not LC data, and not necessarily representative of the type of data we might have in mind, it is a nice data set to use for illustrative purposes.

Figure 4.24 shows the results of using the set to evaluate the hold out log likelihood (where we have removed the portion of the likelihood containing λ and ν for this purpose), and also the latent traces resulting from each parameter setting. Note that in general, a non-zero λ provides notably better results, as compared to the LC-MS TICs in the previous section. This is very likely related to the fact that here we have used much fewer data per latent trace, and hence are more in need of regularization. Also note that the λ and ν terms interact with each other, in that as one increases with the other held constant, the effective magnitude of the constant one is reduced. For $\nu = \infty$ (denoted 'Inf' in figures), we actually ran the single-class version of the model which removes that term entirely. This may explain why the hold out likelihoods (and the latent traces) for $\nu = Inf$ are slightly better than for the next highest values, $\nu = 10^4$, since using the ν term changes the latent trace solver to be iterative from analytic, and also

causes the λ term to interact with the ν term. Recall that higher values for λ make the latent traces smoother, while higher values for ν make the latent traces across classes more similar to one another.

The parameter setting with the highest hold out likelihood is for ($\lambda = 10^2, \nu = 10^{-1}$) which has a hold out log likelihood of 408. The next best settings are for the same value of λ, but with $\nu = 10^{-2}$ and $\nu = 10^{-3}$ which had a hold out likelihood of 397 and 398 respesctively. Thus we see that leaving the two class traces to be independent, while not drastically bad for this data set, might be improved upon by coercing them slightly to be the same, if we believe that the hold out log likelihood is a good hold out measure. Using a single-class solution (*i.e.*, $\nu = Inf$) clearly gives much worse hold out likelihood.

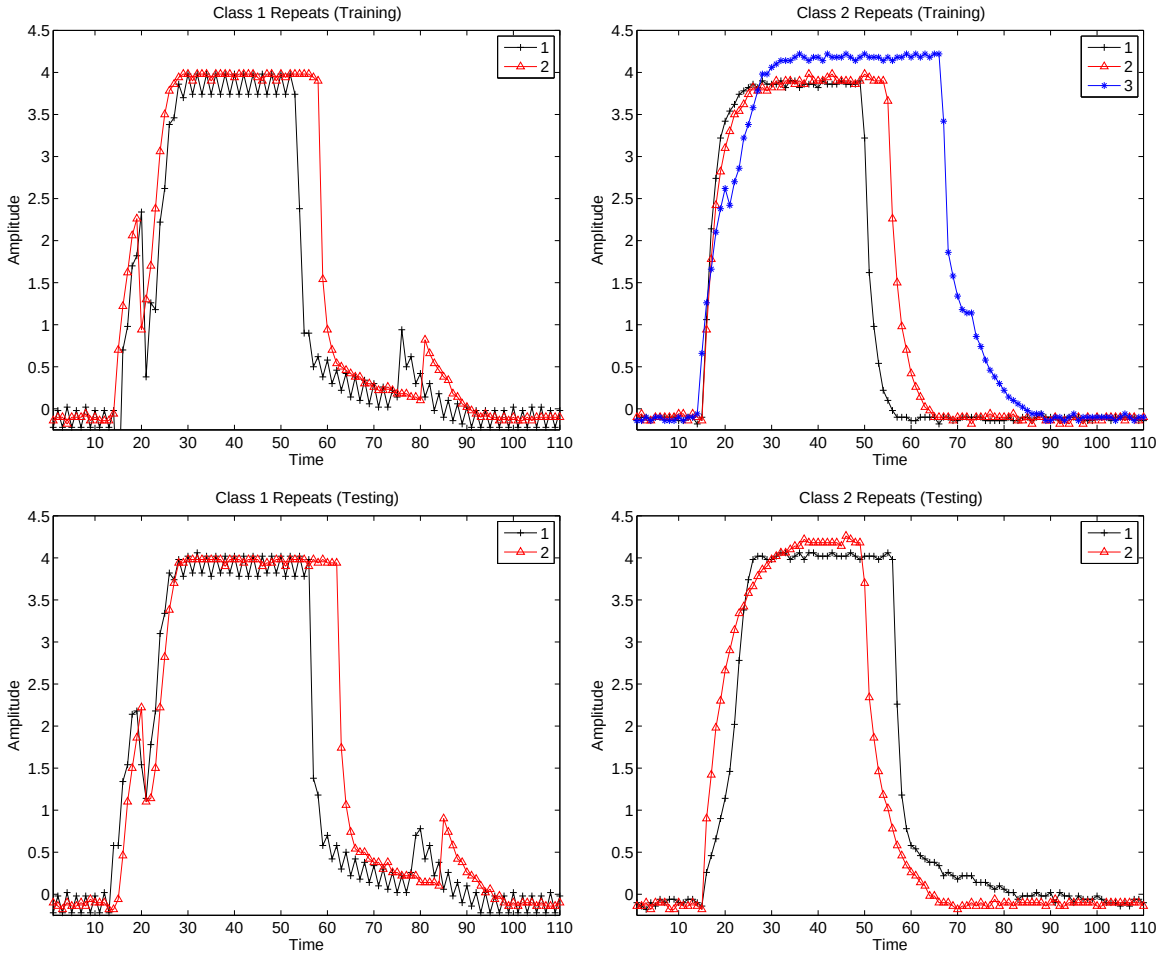


Figure 4.23: Raw NASA data used to illustrate the multi-class EM-CPM.

4.11 Discussion

While the single-class EM-CPM seems to provide a nice solution to the alignment problem, the multi-class EM-CPM has some features which make it less desirable. The fact that we have to use a hold out set to see the effect of two interacting parameters (λ and ν), over a grid of values is inelegant, not to mention impractical. It requires us to separate the data set, which may already have very few items in it (such as the NASA data set), thereby causing us to miss out on much valuable information. Whatever data we choose to include in the training set rather than the hold out set causes our hold out set-based assessment to be that much weaker, whereas whatever data we choose to include into the hold out set rather than the training set weakens our trained model. There is no possible way to make optimal use of all data in a small data set. With the single-class EM-CPM, this same problem exists, however it seemed that the solution was not overly sensitive to the smoothing parameter, whereas with the multi-class data set, the solution is critically dependent upon the value of ν, and it would be impossible not to set this parameter.

One can imagine other possible ways to use the EM-CPM for a multi-class problem. For example, one could imagine using a two-stage process in which one first used the single-class CPM within each class, independently of each another, and then use the multi-class EM-CPM on to align their resulting latent traces (or even just to initialize them). However, this approach seems unlikely to provide any benefit over using the multi-class EM-CPM directly, as we would be limited to using data from only one class at the first stage, rather than leveraging all available data. Furthermore, this two-stage approach would be complicated by the fact that in the second stage, we would be required to align latent traces which are twice the density of the original data. One might consider reducing these representations back down to the original time series density in some *ad hoc* manner, but even then, the second stage must make do with only two time series (the latent traces from the first stage) to do an alignment, and hence would be missing out on information which had been been lost in the compression from observed time series to latent traces in the first stage.

Although the multi-class EM-CPM seems less than ideal, it provides a good spring board and motivation for the model developed in the next chapter, where we move into a fully Bayesian paradigm. In this paradigm, the problems just discussed are eliminated.

Lastly, we now mention some tricks which could be used to reduce the space or time requirements for training of the EM-CPM. The space and time requirements in the Forward-Backward algorithm are linear in N, the length of the time series. For long time series, this could become problematic. One possible solution would be to use the 'checkpointing' idea in [12] which transforms the space requirements from linear to logarithmic, at the cost of an

additional time complexity factor of $\log N$. A related idea of doing approximate just-in-time Viterbi is presented in [56]. On a related note, [43] shows how to modify the Viterbi algorithm to make it a more consistent algorithm, while retaining the advantage that Viterbi training has over full EM with respect to computational speed. Other tricks to speeding up EM are presented in [71, 10, 72, 55]. Also, one might consider using vector quantization (clustering) of the observed time data points into a fixed number of clusters, and then treating the observations as discrete, rather than continuous, Gaussian quantities.

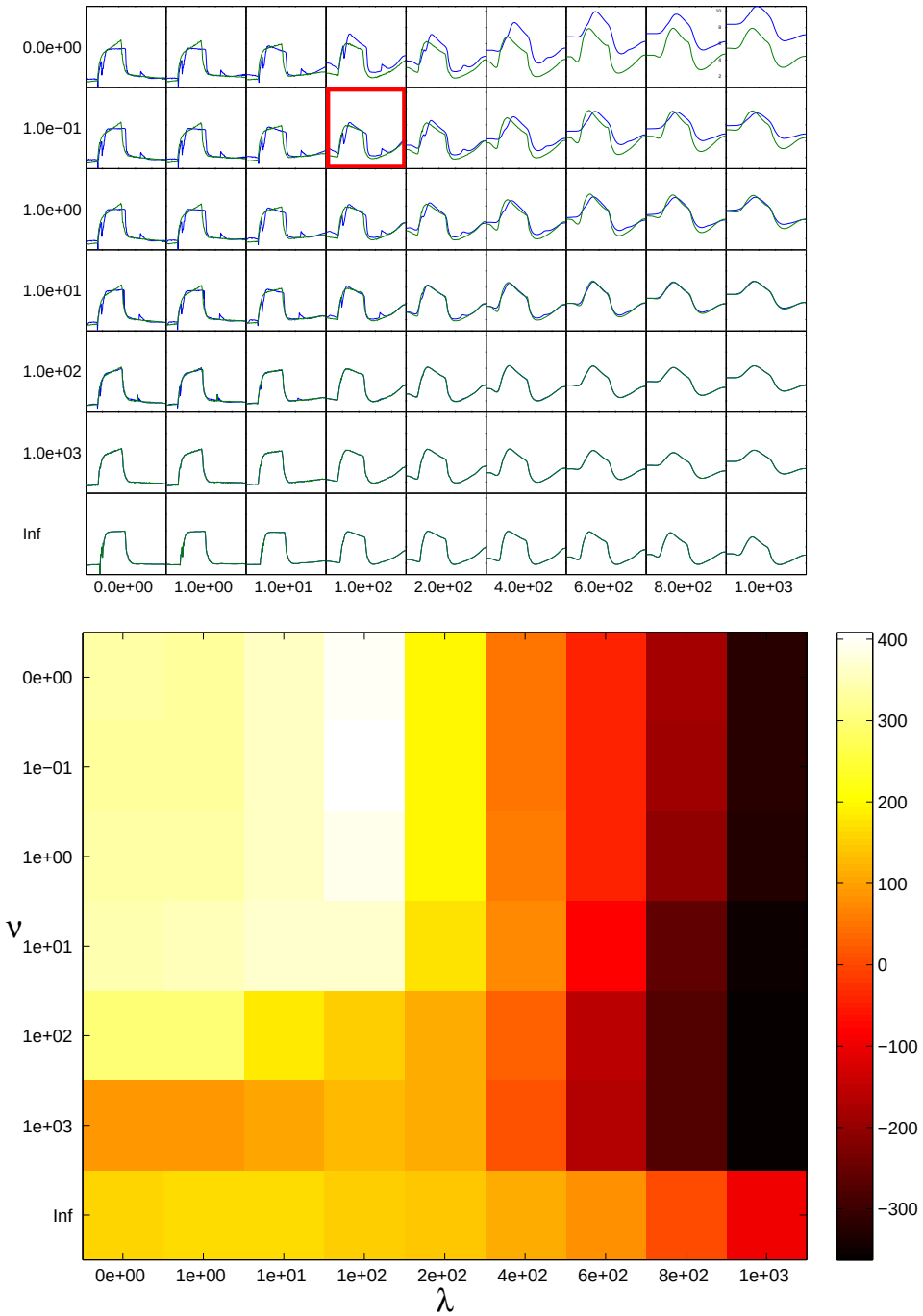


Figure 4.24: Results from using seven different values of ν and nine different values of λ, in conjunction with a hold out set. The top figure shows the class-specific latent traces resulting from the training set, while the bottom figure shows the corresponding log likelihood of the hold out data (removing the λ and ν terms) The best latent traces, as judged by the hold out likelihood, are outlined in bold red lines. Note that $\nu = Inf$ was really just the single-class version of the model.

Chapter 5

A Hierarchical Bayesian Continuous Profile Model

Synopsis: In this chapter we introduce a fully Bayesian instance of the class of CPM models, the *Bayesian Hierarchical Continuous Profile Model* (BH-CPM), as well as an associated (MCMC) algorithm for inference. We then explore use of this model on two data sets – a three-class liquid-chromatography-ultraviolet-diode array data from a study of the plant *Arabidopsis thaliana* and a two-class solenoid valve current data set.

5.1 Motivation

In the previous chapter we introduced a general class of probabilistic, generative models called Continuous Profile Models, for alignment and normalization of sets of continuous-valued time series data. We then delved into a particular instance of CPMs – the EM-CPM, a model in which we used EM to optimize the penalized maximum-likelihood during training. The EM-CPM provided appealing results for the single-class alignment/normalization problem – that is, when all of the time series being aligned were from the same class. However, the multi-class extension of the EM-CPM left much to be desired because training in this model was critically dependent upon accurate cross-validation. With a large number of time series available from each class, such cross-validation would not theoretically be too problematic, although running the cross-validation could be onerous. In the more realistic setting of a very limited number of experimental samples, however, cross-validation can never use all of of the data at once to find the best possible model as it inherently requires partitions of the data. Furthermore, for both the single and multi-class instances of the model, the EM-CPM used parameter point-estimates (only the HMM hidden states were integrated over), which is probably sufficient for parameters

whose posteriors are likely to be unimodal and narrowly distributed, such as the HMM state transition parameters and emission variances, but is likely not ideal for other parameters, such as latent traces and inter/intra-class penalty parameters. A point estimate may even be sufficient for these latter parameters, but this is difficult to assess without examining more than just the MAP estimate which gives us no indication of uncertainty in the trained model.

We are thus motivated to develop a fully Bayesian[1] CPM, which will allow us to:

1. Avoid sacrificing scarce data to cross-validation by instead using MCMC (Markov Chain Monte Carlo) to sample the posterior, which incorporates all available data in a principled way.

2. Obtain error bars on parameter settings, such as the latent traces, so that we can better judge how reliable our alignments are.

3. Develop a hierarchical, multi-class CPM, suitable for problems such as biomarker discovery in which related, but different, sets of time series need to be aligned, normalized and compared to one another.

5.2 Hierarchical Structure: Parent Traces to Child Traces

The Bayesian Hierarchical Continuous Profile Model (HB-CPM), introduced by us in [49], is closely linked in spirit to the EM-CPM. The core of the model is again an HMM which maps each observed time point to an underlying latent time, and this mapping forms the basis of the multiple alignment. However, in contrast to the EM-CPM, we will set the model up in a hierarchical manner that seems particularly appropriate to our multi-class problem: we will use a high-level *parent latent trace* which acts as a prior for class-specific *child latent traces*, which inherit the shape of the parent trace, except for a sparse, class-specific *difference vector* which details how each class differs from the global parent. That is, the joint prior distribution for the child traces in the HB-CPM model can be realized by first sampling a parent trace, and then, for each class, sampling a 'difference vector' which dictates how and where each child trace should differ from the common parent, as depicted in Figure 5.1.

We will estimate the full posterior over all parameters in the model (other than top-level hyper-parameters), and specify proper priors over over all of these parameters. In this sense, the HB-CPM it is a fully Bayesian model.

[1] A brief introduction to the Bayesian modeling paradigm is provided in Appendix E.

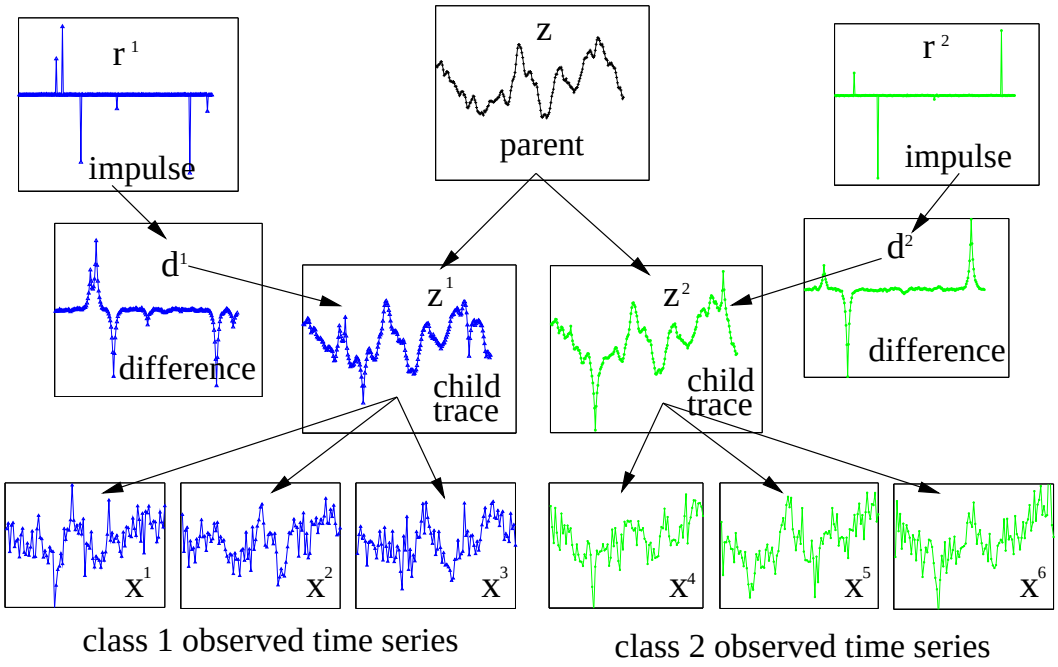


Figure 5.1: Core elements of the HB-CPM model, illustrated with two-class data (hidden and observed) drawn from the model's prior. Two child latent traces are generated from the shape of the parent latent trace, except with differences specified by a sparse 'impulse'/difference vector. Then observed time series are generated from each class-specific child latent trace.

5.3 General Set-Up and Notation Reminder

We will introduce the HB-CPM with scalar time series for simplicity of presentation. Extension to vector time series can be done simply, in a manner similar to the extension of the EM-CPM in Section 4.4.

As in the previous chapter, we let the observed time series for experiment k be $\boldsymbol{x}^k = (x_1^k, x_2^k, ..., x_N^k)$, where $k = 1..K$ and K is the number of observed time series. We use a hierarchical set of *latent traces*. Let $\boldsymbol{z} = (z_1, z_2, ..., z_M)$ be the single parent latent trace, and let $\boldsymbol{z}^c = (z_1^c, z_2^c, ..., z_M^c)$ be each of the class-specific child latent traces, for $c = 1..C$. Each $\boldsymbol{z}^c$ inherits its shape from the shared parent $\boldsymbol{z}$, except for some class-specific differences whose details are provided by a sparse difference vector, the mechanism of which we explain shortly.

Recall that $M > N$, that is, the latent traces lie in an upsampled latent time space to account for possible shifting, stretching and compression of the observed traces relative to one another.

5.4 Formal Specification of the HB-CPM

We now introduce each component of the HB-CPM in turn. For convenience, a summary of all the equations is provided on page 99.

HMM portion of the model

As in the EM-CPM, each observed $\boldsymbol{x}^k$ is modeled as being generated by an HMM conditioned on a latent trace. Recall that each HMM state, π_i, in the EM-CPM mapped to a time state and a scale state, respectively, $\pi_i \rightarrow \{\tau_i, \phi_i\}$. So for the HB-CPM, as with the EM-CPM,

$$\log p(\boldsymbol{x}^k, \boldsymbol{\pi}^k | \boldsymbol{z}^{w^k}) = \log p(\pi_1^k) + \sum_{i=1}^{N} \log \mathcal{N}(x_i^k | z_{\tau_i^k}^{w^k} \phi_i^k u_{\tau_i^k}^k, \xi^k) + \sum_{i=2}^{N} \log T^k_{\pi_{i-1}^k, \pi_i^k},$$

where ξ^k is the emission variance for observed trace k and $u^k_{\tau_i^k}$ is a scaling factor, which can be either global ($u_i^k = u^k$), or local (u_i^k are modeled by a spline function as explained in Section 4.7), and $w^k \in [1..C]$ is the observed class label for the k^{th} observed time series. $T^k_{\pi_{i-1}^k, \pi_i^k}$ are the HMM state transition probabilities, and are modeled as in the EM-CPM, by factoring into scale and time state transitions, $T^k_{\pi_{i-1}, \pi_i} \equiv p(\pi_i | \pi_{i-1}) = p(\phi_i | \phi_{i-1}) p^k(\tau_i | \tau_{i-1})$, each of which is modeled by a multinomial:

$$p^k(\tau_i = a | \tau_{i-1} = b) = \begin{cases} \kappa_1^k, & \text{if } a - b = 1 \\ \kappa_2^k, & \text{if } a - b = 2 \\ \vdots & \\ \kappa_J^k, & \text{if } a - b = J \\ 0, & \text{otherwise} \end{cases}$$

subject to the contraint that $\sum_{i=1}^{J} \kappa_i^k = 1$ and where J is the maximum allowable number of consecutive time states that can be jumped ahead in a single state transition. Similarly the scale state transitions are constrained to only jump at most one scale state at a time:

$$p(\phi_i = a | \phi_{i-1} = b) = \begin{cases} s_0, & \text{if } D(a, b) = 0 \\ s_1, & \text{if } D(a, b) = 1 \\ s_1, & \text{if } D(a, b) = -1 \\ 0, & \text{otherwise} \end{cases}$$

where $D(a,b) = 1$ means that a is one scale state larger than b, and $D(a,b) = -1$ means that a is one scale state smaller than b, and $D(a,b) = 0$ means that $a = b$. This distribution is constrained by $2s_1 + s_0 = 1$. Recall also that in the CPM we put Dirichlet priors with parameters η_v and η'_v on these multinomials, which we again do with the HB-CPM,

$$p(\boldsymbol{\kappa^k}) = \mathcal{D}(\boldsymbol{\kappa^k}|\boldsymbol{\eta}) \propto \prod_{v=1}^{J} (\kappa_v^k)^{\eta_v - 1}, \qquad p(\boldsymbol{s}) = \mathcal{D}(\boldsymbol{s}|\boldsymbol{\eta'}) \propto \prod_{v=1}^{2} (s_v)^{\eta'_v - 1},$$

where v indexes the dimensions of each multinomial. The HMM emission variances are specified by

$$p(\xi^k|\alpha_h, \beta_h) = \text{Inv-Gamma}(\xi^k|\alpha_h, \beta_h).$$

(Note that ξ^k is the variance, not the standard deviation.) Lastly, we put log-normal priors on the HMM emission scaling (spline) parameter(s),

$$p(\log(\mu_d^k)|\xi_u) = \mathcal{N}(\log(\mu_d^k)|0, \xi_u) \text{ (for each control point } j \text{ in the } k^{\text{th}} \text{ time series)}$$

or

$$p(\log(u^k)|\xi_u) = \mathcal{N}(\log(u^k)|0, \xi_u) \text{ (for each global scaling factor in the } k^{\text{th}} \text{ time series).}$$

Hierarchical Latent Traces from Energy Impulse Chains

As mentioned previously, the child traces in the HB-CPM inherit most of their structure from a common parent trace. The differences between child and parent are encoded in a difference vector for each class $\boldsymbol{d}^c = (d_1^c, d_2^c, ..., d_M^c)$. Child traces are obtained by adding this difference vector to the parent trace: $\boldsymbol{z}^c = \boldsymbol{z} + \boldsymbol{d}^c$.

We model both the parent trace and class-specific difference vectors with what we call an *energy impulse chain*, which is an undirected Markov chain in which neighbouring nodes are encouraged to be similar (*i.e.*, smooth), and where this smoothness is perturbed by a set of energy impulse nodes, with one energy impulse node attached to each node in the chain. For the difference vector of the c^{th} class, the corresponding energy impulses are denoted $\boldsymbol{r}^c = (r_1^c, r_2^c, ..., r_M^c)$, and for the parent trace the energy impulses are denoted $\boldsymbol{r} = (r_1, r_2, ..., r_M)$. Conditioned on the energy impulses, the probability of a difference vector is

$$p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho') = \frac{1}{Z_{\boldsymbol{r}^c}} \left[\exp\left(-\frac{1}{2} \left(\sum_{j=1}^{M-1} \frac{(d_j^c - d_{j+1}^c)^2}{\alpha'} + \sum_{j=1}^{M} \frac{(d_j^c - r_j^c)^2}{\rho'} \right) \right) \right], \qquad (5.1)$$

where $Z_{\boldsymbol{r}^c}$ is the normalizing constant and α', ρ' control the relative trade-off between smoothness on the chain, and the influence of the energy impulses, as well as the overall tightness of the distribution. Similarly, the conditional probability of the parent trace is

$$p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho) = \frac{1}{Z_{\boldsymbol{r}}}\left[\exp\left(-\frac{1}{2}\left(\sum_{j=1}^{M-1}\frac{(z_j - z_{j+1})^2}{\alpha} + \sum_{j=1}^{M}\frac{(z_j - r_j)^2}{\rho}\right)\right)\right]. \tag{5.2}$$

These energy impulse chain densities, $p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho)$ and $p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho')$, are each multivariate Gaussian with tridiagonal inverse covariance matrices (corresponding to the Markov nature of the interactions).

The priors for α', ρ', α, ρ are each log-normal. We use separate energy impulse chain parameters for the parent and the child models.

$$p(\log\rho|t,T) = \mathcal{N}(\log\rho|t,T) \tag{5.3}$$
$$p(\log\alpha|h,H) = \mathcal{N}(\log\alpha|h,H) \tag{5.4}$$
$$p(\log\rho'|t',T') = \mathcal{N}(\log\rho'|t',T') \tag{5.5}$$
$$p(\log\alpha'|h',H') = \mathcal{N}(\log\alpha'|h',H'). \tag{5.6}$$

These priors are not conjugate. Even if we were to use the standard choice of an inverse-gamma, which is conjugate for Gaussian variance, such a prior would in fact not be conjugate in our model, since the variance parameters, $\rho, \rho', \alpha, \alpha'$ come repeated in a tri-diagonal precision matrix, not through independent, scalar Gaussians. Also, one could use separate energy impulse chain parameters for each child class, but this would remove some nice conjugacy in in the posterior of the parent traces, which will become apparent in the next section. Furthermore, since we expect sibling classes to be very related, it is not unreasonable to use the same parameters across classes.

Each component of each energy impulse for the parent, r_j, is drawn independently from a single univariate Gaussian. The mean and variance of this Gaussian are drawn from a Gaussian and an inverse-gamma, respectively:

$$p(r_j|\mu_{\text{par}},s_{\text{par}}) = \mathcal{N}(r_j|\mu_{\text{par}},s_{\text{par}}), \tag{5.7}$$

with priors on $\mu_{\text{par}}, s_{\text{par}}$ given by

$$p(\mu_{\text{par}}) = \mathcal{N}(\mu_{\text{par}}|m_{\text{par}},s'_{\text{par}}) \tag{5.8}$$
$$p(s_{\text{par}}) = \text{Inv-Gamma}(s_{\text{par}}|\alpha_{\text{par}},\beta_{\text{par}}). \tag{5.9}$$

The class-specific difference vector impulses are drawn from a mixture of two zero-mean Gaussians – one 'no difference' (inlier) Gaussian, and one 'class-difference' (outlier) Gaussian. The means are zero so as to encourage difference vectors to be near zero (and thus child traces to be similar to the parent trace). Each Gaussian mixture variance has an Inverse-Gamma prior, which for the 'no difference' variance, s_{in}, is set to have very low mean (and not overly dispersed) so that 'no difference' regions truly have little difference from the parent class, while for the 'class-difference' variance, s_{out}, this prior is set to have a larger mean, so as to model our belief that substantial class-specific differences do occasionally exist. These variances have inverse-gamma priors:

$$p(s_{\text{in}}) = \text{Inv-Gamma}(s_{\text{in}}|\alpha_{\text{in}}, \beta_{\text{in}})$$
$$p(s_{\text{out}}) = \text{Inv-Gamma}(s_{\text{out}}|\alpha_{\text{out}}, \beta_{\text{out}})$$

The mixing proportions, $m^c_{\text{in}}, m^c_{\text{out}}$, have a Dirichlet prior, which typically encodes our belief that the proportion that are 'class differences' is likely to be small. Letting δ^c_j denote the binary latent mixture component indicator variables for each r^c_j,

$$p(\delta^c_j) = \text{Multinomial}(\delta^c_j|m^c_{\text{in}}, m^c_{\text{out}}) = (m^c_{\text{in}})^{\delta^c_j}(m^c_{\text{out}})^{1-\delta^c_j} \tag{5.10}$$

$$p(r^c_j|\delta^c_j) = \begin{cases} \mathcal{N}(r^c_j|0, s_{\text{in}}), & \text{if } \delta^c_j = 1 \\ \mathcal{N}(r^c_j|0, s_{\text{out}}), & \text{if } \delta^c_j = 0 \end{cases}. \tag{5.11}$$

Lastly, the mixture model and gaussian hyper-parameters are are distributed as follows:

$$p(m^c_{\text{out}}, m^c_{\text{in}}) = \mathcal{D}(m^c_{\text{out}}, m^c_{\text{out}}|m'_{\text{out}}, m'_{\text{in}}) \tag{5.12}$$

We have now fully specified the HB-CPM.

Note that the prior on the energy impulse nodes, $p(r^c_j|\delta^c_j)$ could also be any mixture of Gaussian distribution while still maintaining tractability in the Gibbs sampling step for $\boldsymbol{d}^c$. This is made clearer when we derive this Gibbs step in the next section. One might even use a t-distribution, since a t-distribution can be viewed as an infinite mixture of gaussians, where the variance of each mixture component is drawn from a particular inverse-gamma distribution.

Summary of the Hierarchical Bayesian Continuous Profile Model

Parent latent trace:

$$p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho) = \frac{1}{Z_{\boldsymbol{r}}}\left[\exp\left(-\frac{1}{2}\left(\sum_{j=1}^{M-1}\frac{(z_j - z_{j+1})^2}{\alpha} + \sum_{j=1}^{M}\frac{(z_j - r_j)^2}{\rho}\right)\right)\right]$$

$$p(r_j|\mu_{\text{par}}, s_{\text{par}}) = \mathcal{N}(r_j|\mu_{\text{par}}, s_{\text{par}}) \quad \textit{Energy Impulses}$$

$$p(\mu_{\text{par}}) = \mathcal{N}(\mu_{\text{par}}|m_{\text{par}}, s'_{\text{par}})$$

$$p(s_{\text{par}}) = \text{Inv-Gamma}(s_{\text{par}}|\alpha_{\text{par}}, \beta_{\text{par}})$$

$$p(\rho|t,T) = \frac{1}{\rho}\mathcal{N}(\log\rho|t,T), \qquad p(\alpha|h,H) = \frac{1}{\alpha}\mathcal{N}(\log\alpha|h,H)$$

Child latent traces:

$$\boldsymbol{z}^c = \boldsymbol{z} + \boldsymbol{d}^c$$

$$p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') = \frac{1}{Z_{\boldsymbol{r}^c}}\left[\exp\left(-\frac{1}{2}\left(\sum_{j=1}^{M-1}\frac{(d_j^c - d_{j+1}^c)^2}{\alpha'} + \sum_{j=1}^{M}\frac{(d_j^c - r_j^c)^2}{\rho'}\right)\right)\right]$$

$$p(r_j^c|\delta_j^c) = \begin{cases}\mathcal{N}(r_j^c|0, s_{\text{in}}), & \text{if } \delta_j^c = 1\\ \mathcal{N}(r_j^c|0, s_{\text{out}}), & \text{if } \delta_j^c = 0\end{cases} \quad \textit{Energy Impulses}$$

$$p(\delta_j^c) = \text{Multinomial}(\delta_j^c|m_{\text{in}}^c, m_{\text{out}}^c) \quad \textit{Mixture Component Indicators}$$

$$p(m_{\text{out}}^c, m_{\text{in}}^c) = \mathcal{D}(m_{\text{out}}^c, m_{\text{out}}^c|m'_{\text{out}}, m'_{\text{in}}) \quad \textit{Mixing Proportions}$$

$$p(s_{\text{in}}) = \text{Inv-Gamma}(s_{\text{in}}|\alpha_{\text{in}}, \beta_{\text{in}})$$

$$p(s_{\text{out}}) = \text{Inv-Gamma}(s_{\text{out}}|\alpha_{\text{out}}, \beta_{\text{out}})$$

$$p(\rho'|t',T') = \frac{1}{\rho'}\mathcal{N}(\log\rho'|t',T'), \qquad p(\alpha'|h',H') = \frac{1}{\alpha'}\mathcal{N}(\log\alpha'|h',H')$$

HMM portion of the model:

$$\log p(\boldsymbol{x}^{\boldsymbol{k}}) = \log p(\pi_1^k) + \sum_{i=1}^{N}\log\mathcal{N}(x_i^k|z_{\tau_i^k}^{w^k}\phi_i^k u_{\tau_i^k}^k, \xi^k) + \sum_{i=2}^{N}\log T^k_{\pi_{i-1}^k,\pi_i^k}$$

$$p(\xi^k) = \text{Inv-Gamma}(\xi^k|\alpha_h, \beta_h)$$

$$T^k_{\pi_j,\pi_i} \equiv p(\pi_i|\pi_j) = p(\phi_i|\phi_j)p^k(\tau_i|\tau_j) \quad \textit{HMM hidden state factorization}$$

$$p^k(\tau_i|\tau_{i-1}) = \text{Multinomial}(\kappa_1^k, \kappa_2^k, \ldots, \kappa_J^k) \quad \textit{HMM time transition distribution}$$

$$p(\phi_i = a|\phi_{i-1} = b) = \text{Multinomial}(s_1, s_0, s_1) \quad \textit{HMM scale transition distribution}$$

$$p(\boldsymbol{\kappa}^{\boldsymbol{k}}) = \mathcal{D}(\boldsymbol{\kappa}^{\boldsymbol{k}}|\boldsymbol{\eta}) \propto \prod_{v=1}^{3}(\kappa_v^k)^{\eta_v - 1} \quad \textit{HMM time transition parameters}$$

$$p(\boldsymbol{s}) = \mathcal{D}(\boldsymbol{s}|\boldsymbol{\eta'}) \propto \prod_{v=1}^{2}(s_v)^{\eta'_v - 1} \quad \textit{HMM scale transition parameters}$$

$$p(u^k) = \frac{1}{u^k}\mathcal{N}(\log(u^k)|0, \xi_u) \quad \textit{HMM scaling parameters}$$

5.5 Training the HB-CPM by MCMC

Given a set of observed time series (and their associated class labels), the main computational operation to be performed in the HB-CPM is inference of the latent traces, hidden state paths and other model parameters. Exact inference is analytically intractable, but we are able to use Markov Chain Monte Carlo (MCMC) methods to create an iterative algorithm which, if run for sufficiently long, produces samples from the correct posterior distribution. This posterior provides simultaneous alignments of all observed time series in all classes, and also, crucially, aligned canonical representations of each class, along with error bars on these representations. We have provided a brief, intuitive tutorial on sampling along with the introduction to the Bayesian modeling paradigm in Appendix E.

We wish to obtain the posterior estimate of some of our parameters conditioned on the data, and marginalized over the other parameters. In particular, we may be interested in obtaining the posterior over hidden time state vectors for each trace, $\boldsymbol{\pi}^k$, which together provide a simultaneous, multi-class alignment of our data. We may, in addition, or, alternatively, be interested in the posterior of the child traces, $\boldsymbol{z}^c$, which together characterize how the classes agree and disagree. The former may be more of interest for visualizing aligned observed time series, or in expanding out aligned scalar time series to a related vector time series, while the latter would be more of interest when looking to characterize differences in multi-class scalar time series data.

We group our parameters into blocks, and sample these blocks conditioned on the values of the other parameters (as in Gibbs sampling) – however, when certain conditional distributions are not amenable to direct sampling, we use slice sampling [60]. The scalar conditional distributions for each of μ_{par}, s_{par}, m^c_{in}, m^c_{out}, δ^c_j, κ^k_i are known distributions, amenable to exact sampling. The conditional distributions for the scalars α^c, ρ^c, α, ρ and u^k are not tractable, and for each of these we use slice sampling (with doubling out and shrinkage).

We now go through one-by-one each of the model parameters/variables (sometimes in blocks), explaining how each can be sampled conditioned on the others.

Sampling Class-Specific Latent Traces

First we concentrate on sampling from $p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho')$, and then later adjust this so that we can sample from $p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho')p(\boldsymbol{x}^k|\boldsymbol{z}, \boldsymbol{d}^c)$ which we need in our Gibbs sampling step. We drop the superscripts c in this section for clarity of notation.

The difference vectors are best sampled each as an entire vector since entries within a vector are correlated. This can be accomplished either by unrolling the energy impulse chain model into an explicit, large, multi-variate Gaussian of dimension M, or by using belief propagation.

The former method of sampling has time complexity in $\Theta(M^3)$ due to the Cholesky decomposition which is required to sample from a multi-variate Gaussian, while the belief propagation has time complexity in $\Theta(M)$.[2] It is instructive to examine both approaches because this will later help us when trying to do the Gibbs step for the parent trace, $\boldsymbol{z}$. It also provides us with a way to double check our code and derivation (for example, by comparing the normalization constant from each method).

Sampling the multivariate Gaussian directly

The energy impulse chain model specifies,

$$p(\boldsymbol{d}|\boldsymbol{r},\alpha',\rho') = \frac{1}{Z_{\boldsymbol{r}}}\left[\exp\left(-\frac{1}{2}\left(\sum_{i=1}^{M-1}\frac{(d_i-d_{i+1})^2}{\alpha'}+\sum_{i=1}^{M}\frac{(d_i-r_i)^2}{\rho'}\right)\right)\right] \tag{5.13}$$

$$\text{and}\quad Z_{\boldsymbol{r}} = \int_{\boldsymbol{d}}\left[\exp\left(-\frac{1}{2}\left(\sum_{i=1}^{M-1}\frac{(d_i-d_{i+1})^2}{\alpha'}+\sum_{i=1}^{M}\frac{(d_i-r_i)^2}{\rho'}\right)\right)\right]d\boldsymbol{d} \tag{5.14}$$

To sample directly from this multi-variate gaussian, we need to figure out what the multi-variate mean and covariance are. That is, we need to try to put it in its standard form of

$$p(\boldsymbol{d}|\boldsymbol{r},\alpha',\rho') = \mathcal{N}(\boldsymbol{d}|\boldsymbol{u},\boldsymbol{Q}) = \frac{1}{(2\pi)^{\frac{M}{2}}|\boldsymbol{Q}|^{\frac{1}{2}}}\exp\left(-\frac{1}{2}(\boldsymbol{d}-\boldsymbol{u})^T\boldsymbol{Q}^{-1}(\boldsymbol{d}-\boldsymbol{u})\right)$$

One way to do this[3] is to start by re-writing the terms in the exponent as shown below in Equation 5.15, which can be seen by inspection. We here demonstrate visually with the case of $M=3$, where M is the length of $\boldsymbol{d}$, but generalizing from this is trivial.

$$p(\boldsymbol{d}|\boldsymbol{r},\alpha',\rho') = \frac{1}{Z_{\boldsymbol{r}}}\left(\prod_{i=1}^{M}\exp\left(-\frac{1}{2}(\boldsymbol{d}-\boldsymbol{v_i})^T\boldsymbol{V_i}(\boldsymbol{d}-\boldsymbol{v_i})\right)\right)\left(\prod_{i=1}^{M}\exp\left(-\frac{1}{2}\boldsymbol{d}^T\boldsymbol{W_i}\boldsymbol{d}\right)\right), \tag{5.15}$$

[2] We use the notation '$f(n)$ is $\Theta(g(n))$' to mean that the function $f(n)$ is tightly bound by $g(n)$ (*i.e.*, bound from below and above).

[3] Thanks to David Ross for first pointing this out to me.

where $\boldsymbol{v_i}$, $\boldsymbol{V_i}$ and $\boldsymbol{W_i}$ are defined as follows:

$$\boldsymbol{v}_1 = \begin{pmatrix} r_1 \\ 0 \\ 0 \end{pmatrix} \qquad \boldsymbol{v}_2 = \begin{pmatrix} 0 \\ r_2 \\ 0 \end{pmatrix} \qquad \boldsymbol{v}_3 = \begin{pmatrix} 0 \\ 0 \\ r_3 \end{pmatrix}$$

$$\boldsymbol{V}_1 = \begin{pmatrix} \frac{1}{\rho'} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{V}_2 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & \frac{1}{\rho'} & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{V}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & \frac{1}{\rho'} \end{pmatrix}$$

and

$$\boldsymbol{W}_1 = \begin{pmatrix} \frac{1}{\alpha'} & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{W}_2 = \begin{pmatrix} 0 & \frac{-1}{\alpha'} & 0 \\ \frac{-1}{\alpha'} & \frac{2}{\alpha'} & 0 \\ 0 & 0 & 0 \end{pmatrix} \qquad \boldsymbol{W}_3 = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & \frac{-1}{\alpha'} \\ 0 & \frac{-1}{\alpha'} & \frac{1}{\alpha'} \end{pmatrix}.$$

Next, we complete the square for Equation 5.15, as shown in Identity F.6, to obtain a Gaussian probability density function,

$$p(\boldsymbol{d}|\boldsymbol{r}, \alpha', \rho') = \frac{1}{\boldsymbol{Z_r}} C \exp\left(-\frac{1}{2}\left(\boldsymbol{d}-\boldsymbol{\mu}\right)^T \boldsymbol{S}^{-1}(\boldsymbol{d}-\boldsymbol{\mu})\right) = \mathcal{N}(\boldsymbol{d}|\boldsymbol{\mu}, \boldsymbol{S}), \tag{5.16}$$

where

$$\boldsymbol{S}^{-1} = \sum_i \boldsymbol{W_i} + \sum_i \boldsymbol{V_i} = \begin{pmatrix} \frac{1}{\alpha'} + \frac{1}{\rho'} & \frac{-1}{\alpha'} & 0 \\ \frac{-1}{\alpha'} & \frac{2}{\alpha'} + \frac{1}{\rho'} & \frac{-1}{\alpha'} \\ 0 & \frac{-1}{\alpha'} & \frac{1}{\alpha'} + \frac{1}{\rho'} \end{pmatrix} \tag{5.17}$$

$$\boldsymbol{\mu} = \boldsymbol{S}\left(\sum_i \boldsymbol{V_i v_i}\right) = \boldsymbol{S}\begin{pmatrix} \frac{r_1}{\rho'} \\ \frac{r_2}{\rho'} \\ \frac{r_3}{\rho'} \end{pmatrix} = \frac{\boldsymbol{S}}{\rho'}\boldsymbol{r} \tag{5.18}$$

$$C = \exp\left(-\frac{1}{2}\left[\left(\sum_i \boldsymbol{v_i}^T \boldsymbol{V_i v_i}\right) - \boldsymbol{\mu}^T \boldsymbol{S}^{-1}\boldsymbol{\mu}\right]\right) = \exp\left(-\frac{1}{2}\left[\left(\frac{1}{\rho'}\sum_i r_i^2\right) - \frac{\boldsymbol{r}^T \boldsymbol{S}\boldsymbol{r}}{\rho'}\right]\right). \tag{5.19}$$

Thus the normalization factor, $\boldsymbol{Z_r}$, is easily seen to be

$$Z_{\boldsymbol{r}} = C(2\pi)^{\frac{M}{2}}|\boldsymbol{S}|^{\frac{1}{2}}. \tag{5.20}$$

In this manner then, one can sample $\boldsymbol{d}$ from this Gaussian using the standard sampling technique, which uses the Cholesky decomposition of the covariance matrix. Next we show how one could instead use belief propagation to do this same task in a more efficient manner.

Sampling by Belief Propagation

Recall that belief propagation is nothing more than dynamic programming tricks which make use of the structure of our model to re-use computations rather than recomputing them as required in the naive approach. We will use such tricks to more efficiently sample $\boldsymbol{d}$.

Let $\phi(d_i, d_{i+1}) = \exp(-\frac{1}{2}\frac{(d_i - d_{i+1})^2}{\alpha'})$ and $\beta(d_i) = \exp(-\frac{1}{2}\frac{(d_i - r_i)^2}{\rho'})$, then

$$p(\boldsymbol{d}|\boldsymbol{r}, \alpha', \rho') = \frac{1}{Z_{\boldsymbol{r}}}\left[\exp\left(-\frac{1}{2}\left(\sum_{i=1}^{M-1}\frac{(d_i - d_{i+1})^2}{\alpha'} + \sum_{i=1}^{M}\frac{(d_i - r_i)^2}{\rho'}\right)\right)\right] \tag{5.21}$$

$$= \frac{1}{Z_{\boldsymbol{r}}}\left(\prod_{i=1}^{M-1}\phi(d_i, d_{i+1})\right)\left(\prod_{i=1}^{M}\beta(d_i)\right) \tag{5.22}$$

$$\text{and} \quad Z_{\boldsymbol{r}} = \int_{\boldsymbol{d}}\left(\prod_{i=1}^{M-1}\phi(d_i, d_{i+1})\right)\left(\prod_{i=1}^{M}\beta(d_i)\right)d\boldsymbol{d} \tag{5.23}$$

To efficiently sample $\boldsymbol{d} \sim p(\boldsymbol{d}|\boldsymbol{r}, \alpha', \rho')$, we can make use of the chain rule in probability as well as the fact that our graphical model forms a chain. With the chain rule in probability, we can trivially write

$$p(\boldsymbol{d}|\boldsymbol{r}) = p(d_1|d_2 \ldots d_M, \boldsymbol{r})p(d_2|d_3 \ldots d_M, \boldsymbol{r}) \ldots p(d_i|d_{i+1} \ldots d_M, \boldsymbol{r}) \ldots\ldots p(d_M|\boldsymbol{r})$$

which because of our chain-structured graphical model, is equivalent to writing

$$p(\boldsymbol{d}|\boldsymbol{r}) = p(d_1|d_2, \boldsymbol{r})p(d_2|d_3, \boldsymbol{r}) \ldots p(d_i|d_{i+1}, \boldsymbol{r}) \ldots\ldots p(d_M|\boldsymbol{r}).$$

We can think of sampling from the joint by first sampling from the marginal $p(d_M|\boldsymbol{r})$, and then progressively working backward by conditioning on the last variable sampled. Since $p(d_i|d_{i+1}, \boldsymbol{r}) = \frac{p(d_i, d_{i+1}|\boldsymbol{r})}{p(d_{i+1}|\boldsymbol{r})}$, we need only be able to calculate pairwise marginals of the form $p(d_i, d_{i+1}|\boldsymbol{r})$ and then we are done (except for applying an identity to convert these pairwise marginals to the desired conditionals). These terms can be efficiently computed because the underlying graphical model is a chain with exponential-form clique potentials. We now show how to compute these terms, using recursive forward and backward message-passing algorithms similar to those in a Kalman Filter [35]. Similarly to Kalman Filtering, we will find it convenient to use a two-step update for each recursive step. Thus we will have a 'predictor'

step which produces intermediate forward and backward messages, respectively, f_j^* and b_j^*, and then a corrector step which produces the final messages, f_j and b_j, from their intermediate counterparts. These messages are defined as follows, and a derivation of how to calculate them appears later.

$$
\begin{aligned}
f_1 &\equiv \beta(d_1) \\
f_j^* &\equiv \int_{d_1 \ldots d_{j-1}} \prod_{i=1}^{j-1} \phi(d_i, d_{i+1}) \prod_{i=1}^{j-1} \beta(d_i)\, dd_1 \ldots dd_{j-1} \\
f_j &\equiv f_j^* \beta(d_j) \\
b_M &\equiv \beta(d_M) \\
b_j^* &\equiv \int_{d_{j+1} \ldots d_M} \prod_{i=j}^{M-1} \phi(d_i, d_{i+1}) \prod_{i=j+1}^{M} \beta(d_i)\, dd_{j+1} \ldots dd_M \\
b_j &\equiv b_j^* \beta(d_j)
\end{aligned}
$$

From these definitions, it follows that

$$Z_{\boldsymbol{r}} = \int_{d_j} f_j b_j^*\, dd_j = \int_{d_M} f_M\, dd_M = \int_{d_1} b_1\, dd_1 \tag{5.24}$$

$$p(d_j|\boldsymbol{r}) = \frac{1}{Z_{\boldsymbol{r}}} f_j b_j^* \tag{5.25}$$

$$p(d_{j-1}, d_j|\boldsymbol{r}) = \frac{1}{Z_{\boldsymbol{r}}} f_j \phi(d_j, d_{j+1}) b_{j+1}, \tag{5.26}$$

Note that since all clique potentials are unnormalized Gaussians, it follows that all messages will be unnormalized Gaussians. Thus every message, can be expressed as a Gaussian times a constant. In particular, we will use the following notation:

$$
\begin{aligned}
f_j^* &\propto \mathcal{N}(d_j|v_j, V_j) \\
f_j &\propto \mathcal{N}(d_j|u_j, U_j) \\
b_j^* &\propto \mathcal{N}(d_j|q_j, Q_j) \\
b_j &\propto \mathcal{N}(d_j|s_j, S_j).
\end{aligned}
$$

If we want to compute the normalization constant, $Z_{\boldsymbol{r}}$, then we can keep track of the forward and backward, individual proportionality constants, respectively c_j and o_j, corresponding to $f_j = c_j \mathcal{N}(d_j|u_j, U_j)$ and $b_j = o_j \mathcal{N}(d_j|s_j, S_j)$. We do not require computation of these in order to sample from the joint, and so can instead only keep the the means and covariances of the four types of messages. However, we will include these proportionality constants in our

derivations so that we can calculate Z_r multiple ways (*e.g.*, using Equation 5.24 for different values of j, or by using the Gaussian formulation provided in 5.20). In actual implementation, we will store the log of these proportionality constants to avoid floating point errors as they become very small.

We can compute the forward messages as follows, using Identity F.2 throughout, and starting with the predictor step:

$$\begin{aligned}
f_1 &= c_1 \mathcal{N}(d_1 | r_1, \rho'), \quad \text{where } c_1 = (\sqrt{2\pi\rho'}) \\
f_j^* &= \int_{d_1 \ldots d_{j-1}} \prod_{i=1}^{j-1} \phi(d_i, d_{i+1}) \prod_{i=1}^{j-1} \beta(d_i)\, dd_1 \ldots dd_{j-1} \\
&= \int_{d_{j-1}} \phi(d_{j-1}, d_j) f_{j-1}\, dd_{j-1} \\
&= \int_{d_{j-1}} \phi(d_{j-1}, d_j) c_{j-1} \mathcal{N}(d_{j-1} | u_{j-1}, U_{j-1})\, dd_{j-1} \\
&= c_{j-1} \int_{d_{j-1}} (\sqrt{2\pi\alpha'}) \mathcal{N}(d_{j-1} | d_j, \alpha') \mathcal{N}(d_{j-1} | u_{j-1}, U_{j-1})\, dd_{j-1} \\
&= (\sqrt{2\pi\alpha'}) c_{j-1} \mathcal{N}(d_j | v_j, V_j) \int_{d_{j-1}} \mathcal{N}(d_{j-1} | g_j, G_j)\, dd_{j-1} \\
&= (\sqrt{2\pi\alpha'}) c_{j-1} \mathcal{N}(d_j | v_j, V_j),
\end{aligned}$$

where

$$V_j = U_{j-1} + \alpha', \qquad v_j = u_{j-1}.$$

The corrector step then is given by

$$\begin{aligned}
f_j &= f_j^* \beta(d_j) \\
&= \left((\sqrt{2\pi\alpha'}) c_{j-1} \mathcal{N}(d_j | v_j, V_j) \right) \left((\sqrt{2\pi\rho'}) \mathcal{N}(d_j | r_j, \rho') \right) \\
&= c_{j-1} (2\pi) (\sqrt{\alpha'\rho'}) \mathcal{N}(d_j | v_j, V_j) \mathcal{N}(d_j | r_j, \rho') \\
&= c_{j-1} (2\pi) (\sqrt{\alpha'\rho'}) \mathcal{N}(v_j | r_j, \rho' + V_j) \mathcal{N}(d_j | u_j, U_j) \\
&= c_j \mathcal{N}(d_j | u_j, U_j),
\end{aligned}$$

where

$$U_j = (\frac{1}{\rho'} + \frac{1}{V_j})^{-1}, \quad u_j = U_j (\frac{v_j}{V_j} + \frac{r_j}{\rho'}), \quad c_j = c_{j-1} (2\pi) (\sqrt{\alpha'\rho'}) \mathcal{N}(v_j | r_j, \rho' + V_j).,$$

Also, $Z_{\boldsymbol{r}} = c_M\sqrt{2\pi U_M}$.

The backward algorithm is identical in structure to the forward algorithm – only the indexes change (and only on the predictor step) because we are working backward:

$$\begin{aligned}
b_M &= o_M\mathcal{N}(d_M|r_M,\rho'), \quad \text{where } o_M = (\sqrt{2\pi\rho'}) \\
b_j^* &= \int_{d_{j+1}\ldots d_M} \prod_{i=j}^{M}\phi(d_i,d_{i+1}) \prod_{i=j+1}^{M}\beta(d_i)\,dd_{j+1}\ldots dd_M \\
&= \int_{d_{j+1}} \phi(d_j,d_{j+1})f_{j+1}\,dd_{j+1} \\
&= \int_{d_{j+1}} \phi(d_j,d_{j+1})o_{j+1}\mathcal{N}(d_{j+1}|s_{j+1},S_{j+1})\,dd_{j+1} \\
&= o_{j+1}\int_{d_{j+1}} (\sqrt{2\pi\alpha'})\mathcal{N}(d_j|d_{j+1},\alpha')\mathcal{N}(d_{j+1}|s_{j+1},S_{j+1})\,dd_{j+1} \\
&= o_{j+1}(\sqrt{2\pi\alpha'})\mathcal{N}(d_j|q_j,Q_j)\int_{d_{j+1}} \mathcal{N}(d_{j+1}|g_j,G_j)\,dd_{j+1} \\
&= o_{j+1}(\sqrt{2\pi\alpha'})\mathcal{N}(d_j|q_j,Q_j),
\end{aligned}$$

where

$$Q_j = S_{j-1} + \alpha, \quad q_j = s_{j-1},$$

followed by

$$\begin{aligned}
b_j &= b_j^*\beta(d_j) \\
&= \left(o_{j+1}(\sqrt{2\pi\alpha'})\mathcal{N}(d_j|q_j,Q_j)\right)\left((\sqrt{2\pi\rho'})\mathcal{N}(d_j|r_j,\rho')\right) \\
&= o_{j+1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(d_j|q_j,Q_j)\mathcal{N}(d_j|r_j,\rho') \\
&= o_{j+1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(q_j|r_j,Q_j+\rho')\mathcal{N}(d_j|s_j,S_j), \\
&= o_j\mathcal{N}(d_j|s_j,S_j),
\end{aligned}$$

where

$$S_j = (\frac{1}{\rho'} + \frac{1}{Q_j})^{-1}, \quad s_j = S_j(\frac{q_j}{Q_j} + \frac{r_j}{\rho'}), \quad o_j = o_{j+1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(q_j|r_j,Q_j+\rho').$$

After these for forward and backward messages have been computed, we can then compute

the pairwise marginals, $p(d_j, d_{j+1}|\boldsymbol{r})$, as follows

$$\begin{aligned}
p(d_j, d_{j+1}|\boldsymbol{r}) &\propto f_j b_{j+1} \phi(d_j, d_{j+1}) \\
&\propto \mathcal{N}(d_j|u_j, U_j)\mathcal{N}(d_{j+1}|s_{j+1}, S_{j+1})\mathcal{N}(d_j|d_{j+1}, \alpha') \\
&\propto \mathcal{N}\left(\begin{bmatrix} d_j \\ d_{j+1} \end{bmatrix} | \begin{bmatrix} u_j \\ s_{j+1} \end{bmatrix}, \begin{bmatrix} U_j & 0 \\ 0 & S_{j+1} \end{bmatrix}\right) \mathcal{N}(d_j|d_{j+1}, \alpha') \\
&\propto \mathcal{N}\left(\begin{bmatrix} d_j \\ d_{j+1} \end{bmatrix} | \begin{bmatrix} u_j \\ s_{j+1} \end{bmatrix}, \begin{bmatrix} U_j & 0 \\ 0 & S_{j+1} \end{bmatrix}\right) \mathcal{N}\left(\begin{bmatrix} d_j \\ d_{j+1} \end{bmatrix} | \begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \frac{1}{\alpha'} & -\frac{1}{\alpha'} \\ -\frac{1}{\alpha'} & \frac{1}{\alpha'} \end{bmatrix}^{-1}\right) \\
&\equiv \mathcal{N}(\boldsymbol{d_{j,j+1}}|\boldsymbol{\psi_j}, \boldsymbol{\Psi_j})\mathcal{N}(\boldsymbol{d_{j,j+1}}|\boldsymbol{\lambda_j}, \boldsymbol{\Lambda_j}) \\
&\propto \mathcal{N}(\boldsymbol{d_{j,j+1}}|\boldsymbol{m_j}, \boldsymbol{M_j}),
\end{aligned}$$

where, again appealing to Identity F.2,

$$\begin{aligned}
\boldsymbol{M_j} &= (\boldsymbol{\Psi_j}^{-1} + \boldsymbol{\Lambda_j}^{-1})^{-1} \\
\boldsymbol{m_j} &= \boldsymbol{M_j}(\boldsymbol{\Psi_j}^{-1}\boldsymbol{\psi_j} + \boldsymbol{\Lambda_j}^{-1}\boldsymbol{\lambda_j}) \\
&= \boldsymbol{M_j}\boldsymbol{\Psi_j}^{-1}\boldsymbol{\psi_j}
\end{aligned}$$

Lastly, we use Identity F.4 to convert the pairwise marginals to the pairwise conditional marginals required. To start our sampling, we will need $p(d_M|\boldsymbol{r})$ which is given by $p(d_M|\boldsymbol{r}) \propto f_M \propto \mathcal{N}(d_M|u_M, U_M)$.

Adding the HMM Evidence

So far we have shown how to sample from the prior, $\boldsymbol{d}^c \sim p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho')$, but for Gibbs sampling, we need to sample from $\boldsymbol{d}^c \sim \frac{1}{Z}p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho') \prod_{\{k|w^k=c\}} p(\boldsymbol{x^k}|\boldsymbol{z}, \boldsymbol{d}^c)$, which, can

be written as

$$\propto p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') \prod_{\{k|w^k=c\}} \prod_{i=1}^{N} p(\boldsymbol{x^k}|\boldsymbol{z},\boldsymbol{d}^c)$$

$$= p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') \prod_{\{k|w^k=c\}} \prod_{i=1}^{N} \mathcal{N}(x_i^k|(z_{\tau_i^k}+d^c_{\tau_i^k})\phi_i^k u^k_{\tau_i^k},\xi^k)$$

$$= p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') \prod_{\{k|w^k=c\}} \prod_{i=1}^{N} \left(\frac{\sqrt{2\pi\xi^k}}{\phi_i^k u^k_{\tau_i^k}}\right) \mathcal{N}\left(d^c_{\tau_i^k}|\frac{(x_i^k-\phi_i^k u^k_{\tau_i^k} z_{\tau_i^k})}{(\phi_i^k u^k_{\tau_i^k})},\frac{\xi^k}{(\phi_i^k u^k_{\tau_i^k})^2}\right) \mathcal{N}\left(x_i^k|(z_{\tau_i^k}\phi_i^k u^k_{\tau_i^k}),\frac{\xi^k}{2}\right) \tag{5.27}$$

$$\propto p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') \prod_{\{k|w^k=c\}} \prod_{i=1}^{N} \mathcal{N}\left(d^c_{\tau_i^k}|\frac{(x_i^k-\phi_i^k u^k_{\tau_i^k} z_{\tau_i^k})}{(\phi_i^k u^k_{\tau_i^k})},\frac{\xi^k}{(\phi_i^k u^k_{\tau_i^k})^2}\right) \tag{5.28}$$

$$\propto p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho') \prod_{j=1}^{M} \mathcal{N}(d_j|a_j^c,A_j^c) \tag{5.29}$$

where Equation 5.27 is obtained with the help of Identity F.3, and Equation 5.29 is obtained by recursive use of Identity F.2. Each $\mathcal{N}(d_j|a_j^c,A_j^c)$ represents the combined evidence from all observed time series in class c, from the real time point which maps to latent time j (if it exists). Letting $\delta_i^k = \frac{(x_i^k-\phi_i^k u^k_{\tau_i^k} z_{\tau_i^k})}{(\phi_i^k u^k_{\tau_i^k})}$ and $\Delta_i^k = \frac{\xi^k}{(\phi_i^k u^k_{\tau_i^k})^2}$, then

$$A_j^c = \left(\sum_{\{k|w^k=c\}} \sum_{\{i|\tau_i^k=j\}}^{N} \frac{1}{\Delta_i^k}\right)^{-1}, \qquad a_j^c = A_j^c \left(\sum_{\{k|w^k=c\}} \sum_{\{i|\tau_i^k=j\}}^{N} \frac{\delta_i^k}{\Delta_i^k}\right).$$

The end effect of this HMM evidence is to add in a single extra 'observation' into the belief propagation algorithm that we derived for $p(\boldsymbol{d}^c|\boldsymbol{r}^c,\alpha',\rho')$. Letting $f'_j = c'_j\mathcal{N}(d_j|u'_j,U'_j)$ be the new forward message which incorporates this evidence, we do so as follows:

$$f'_j = f^*_j\beta(d_j)\mathcal{N}(d_j|a_j,A_j) \tag{5.30}$$

$$= c'_{j-1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(v_j|r_j,\rho'+V_j)\mathcal{N}(d_j|u_j,U_j)\mathcal{N}(d_j|a_j,A_j) \tag{5.31}$$

$$= c'_{j-1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(v_j|r_j,\rho'+V_j)\mathcal{N}(u_j|a_j,U_j+A_j)\mathcal{N}(d_j|u'_j,U'_j) \tag{5.32}$$

$$= c'_j\mathcal{N}(d_j|u'_j,U'_j), \tag{5.33}$$

where,

$$U'_j = (\frac{1}{A_j} + \frac{1}{U_j})^{-1}, \quad u'_j = U'_j(\frac{a_j}{A_j} + \frac{u_j}{U_u}) \tag{5.34}$$

$$c'_j = c'_{j-1}(2\pi)(\sqrt{\alpha'\rho'})\mathcal{N}(v_j|r_j, \rho' + V_j)\mathcal{N}(u_j|a_j, U_j + A_j) \tag{5.35}$$

and where U_j, u_j are defined exactly as before. Note that we also need to make sure that c'_1 and f'_1 account for all observations at the first time point. Computation of the backward messages is again directly analogous.

Now we can sample for a 'difference' vector, $\boldsymbol{d}^c$ for each class using belief propagation, and then deterministically obtain the class-specific latent-traces from the parent traces by $\boldsymbol{z}^c = \boldsymbol{z} + \boldsymbol{d}^c$.

Sampling the Parent Trace

Sampling the parent trace is very similar to sampling the class specific traces. Instead of sampling for a difference vector, $\boldsymbol{d}^c$, from a multivariate gaussian, as we did for the child traces, we instead sample for the parent trace, $\boldsymbol{z}$, directly from a multivariate gaussian. And instead of incorporating evidence from the observation in the HMM, we need to incorporate information from from the class-specific traces. That is, we need to sample

$$\boldsymbol{z} \sim \frac{1}{Z}p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho)\left(\prod_c p(\boldsymbol{d}^c|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho')\right) = \frac{1}{Z}p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho)\left(\prod_c p(\boldsymbol{z}^c - \boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho')\right). \tag{5.36}$$

The first term, $p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho)$, being an energy impulse chain model, while the second term, $\prod_c p(\boldsymbol{z}^c - \boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho')$, is what couples the class specific traces to the parent trace and is analogous to the HMM evidence when sampling the class-specific traces). Equation 5.16 tells us how to re-write the each factor in the second term as a multivariate gaussian, and from this, we

can manipulate to obtain gaussians in $\boldsymbol{z}$:

$$
\begin{aligned}
&p(\boldsymbol{z}^c - \boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho') \\
&= \mathcal{N}(\boldsymbol{z}^c - \boldsymbol{z}|\boldsymbol{\mu}^c, \boldsymbol{S}) \\
&= (2\pi)^{\frac{D}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\boldsymbol{z}^c - \boldsymbol{z} - \boldsymbol{\mu}^c)^T \boldsymbol{S}^{-1}(\boldsymbol{z}^c - \boldsymbol{z} - \boldsymbol{\mu}^c)\right) \\
&\propto \exp\left(-\frac{1}{2}(\boldsymbol{z}^c - \boldsymbol{z} - \boldsymbol{\mu}^c)^T \boldsymbol{S}^{-1}(\boldsymbol{z}^c - \boldsymbol{z} - \boldsymbol{\mu}^c)\right) \\
&= \exp\left(-\frac{1}{2}(-\boldsymbol{z}^c + \boldsymbol{z} + \boldsymbol{\mu}^c)^T \boldsymbol{S}^{-1}(-\boldsymbol{z}^c + \boldsymbol{z} + \boldsymbol{\mu}^c)\right) \\
&\propto \exp\left(-\frac{1}{2}\left[\boldsymbol{z}^T \boldsymbol{S}^{-1}\boldsymbol{z} - 2\boldsymbol{z}^T \boldsymbol{S}^{-1}(\boldsymbol{z}^c - \boldsymbol{\mu}^c)\right]\right) \\
&\propto \mathcal{N}(\boldsymbol{z}|\boldsymbol{z}^c - \boldsymbol{\mu}^c, \boldsymbol{S}),
\end{aligned}
$$

where expressions for $\boldsymbol{\mu}^c$, $\boldsymbol{S}$ are provided in Equations 5.17, 5.18 (recall $\boldsymbol{u}^c = \frac{\boldsymbol{S}}{\rho'}\boldsymbol{r}^c$), and where we appeal to Identity F.5 to obtain the last line. Then

$$
\prod_c p(\boldsymbol{d}^c|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho') = \prod_c \mathcal{N}(\boldsymbol{z}|\boldsymbol{z}^c - \boldsymbol{\mu}^c, \boldsymbol{S}) \propto \mathcal{N}(\boldsymbol{z}|\frac{\sum_c(\boldsymbol{z}^c - \boldsymbol{\mu}^c)}{C}, \frac{\boldsymbol{S}}{C}),
$$

where C is the number of classes. We can also re-write $p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho)$, as a multivariate gaussian, $p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho) = \mathcal{N}(\boldsymbol{z}|\boldsymbol{g}, \boldsymbol{G})$, where, similarly to Equations 5.17 and 5.18, we have (for the 3×3 case to illustrate)

$$
\boldsymbol{G}^{-1} = \begin{pmatrix} \frac{1}{\alpha} + \frac{1}{\rho} & \frac{-1}{\alpha} & 0 \\ \frac{-1}{\alpha} & \frac{2}{\alpha} + \frac{1}{\rho} & \frac{-1}{\alpha} \\ 0 & \frac{-1}{\alpha} & \frac{1}{\alpha} + \frac{1}{\rho} \end{pmatrix}, \qquad \boldsymbol{g} = \frac{\boldsymbol{G}}{\rho}\boldsymbol{r}
$$

Going back to our original goal of the Gibbs step for the parent trace, we need to sample from

$$
\begin{aligned}
\boldsymbol{z} &\sim \frac{1}{Z} p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho) \left(\prod_c p(\boldsymbol{z}^c - \boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}}, \alpha', \rho')\right) \\
&\propto \mathcal{N}(\boldsymbol{z}|\boldsymbol{g}, \boldsymbol{G})\ \mathcal{N}(\boldsymbol{z}|\frac{\sum_c(\boldsymbol{z}^c - \boldsymbol{\mu}^c)}{C}, \frac{\boldsymbol{S}}{C}) \\
&\propto \mathcal{N}(\boldsymbol{z}|\boldsymbol{b}, \boldsymbol{B})\ ,
\end{aligned}
$$

where $\boldsymbol{B}^{-1}$ is tridiagonal (given below with $\boldsymbol{b}$), and hence the gaussian can be efficiently sampled with belief propagation, analogously to the case for sampling $\boldsymbol{d}^c$.

$$\begin{aligned}
\boldsymbol{B}^{-1} &= (\boldsymbol{G}^{-1} + C\boldsymbol{S}^{-1}) \\
\boldsymbol{b} &= \boldsymbol{B}\left(\boldsymbol{G}^{-1}\boldsymbol{g} + C\boldsymbol{S}^{-1}\frac{\sum_c(\boldsymbol{z}^c - \boldsymbol{\mu}^c)}{C}\right) \\
&= \boldsymbol{B}\left(\boldsymbol{G}^{-1}\boldsymbol{g} + \boldsymbol{S}^{-1}\sum_c(\boldsymbol{z}^c - \boldsymbol{\mu}^c)\right) \\
&= \boldsymbol{B}\left(\frac{\boldsymbol{r}}{\rho} + \boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c - \sum_c \frac{\boldsymbol{r}^c}{\rho'}\right)
\end{aligned}$$

Comparing the equations above for $\boldsymbol{B}$ and $\boldsymbol{b}$ to those for $\boldsymbol{S}$ and $\boldsymbol{u}$ in Equations 5.17 and 5.18, which specified the multivariate Gaussian mean and covariance for our Energy Impulse Chain model, we may notice that $\mathcal{N}(\boldsymbol{z}|\boldsymbol{b}, \boldsymbol{B})$ can be written as an Energy Impulse Chain, which is convenient since we have already worked out belief propagation for the Energy Impulse Chain. Let $\frac{1}{\alpha^*} = \frac{1}{\alpha} + \frac{C}{\alpha'}$ and $\frac{1}{\rho^*} = \frac{1}{\rho} + \frac{C}{\rho'} = \frac{\rho' + C\rho}{\rho\rho'}$. Also, notice that

$$\begin{aligned}
\boldsymbol{b} &= \boldsymbol{B}\left(\frac{\boldsymbol{r}}{\rho} + \boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c - \sum_c \frac{\boldsymbol{r}^c}{\rho'}\right) \\
&= \boldsymbol{B}\left(\frac{\rho'\boldsymbol{r} - \rho\sum_c \boldsymbol{r}^c + \rho\rho'\boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c}{\rho\rho'}\right) \\
&= \boldsymbol{B}\left(\left[\frac{\rho'\boldsymbol{r} - \rho\sum_c \boldsymbol{r}^c + \rho\rho'\boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c}{\rho\rho'}\right]\left[\frac{\rho' + C\rho}{\rho\rho'}\right]\left[\frac{\rho\rho'}{\rho' + C\rho}\right]\right) \\
&= \boldsymbol{B}\frac{1}{\rho^*}\left(\left[\frac{\rho'\boldsymbol{r} - \rho\sum_c \boldsymbol{r}^c + \rho\rho'\boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c}{\rho\rho'}\right]\left[\frac{\rho\rho'}{\rho' + C\rho}\right]\right) \\
&= \boldsymbol{B}\frac{1}{\rho^*}\left(\frac{\rho'\boldsymbol{r} - \rho\sum_c \boldsymbol{r}^c + \rho\rho'\boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c}{\rho' + C\rho}\right) \\
&= \boldsymbol{B}\frac{\boldsymbol{r}^*}{\rho^*},
\end{aligned}$$

where $\boldsymbol{r}^* \equiv \frac{\rho'\boldsymbol{r} - \rho\sum_c \boldsymbol{r}^c + \rho\rho'\boldsymbol{S}^{-1}\sum_c \boldsymbol{z}^c}{\rho' + C\rho}$. It now easily follows (from direct comparison with Equations 5.17, 5.18 and 5.13) that

$$\mathcal{N}(\boldsymbol{z}|\boldsymbol{b}, \boldsymbol{B}) = \frac{1}{Z_{\boldsymbol{r}^*}}\left[\exp\left(-\frac{1}{2}\left(\sum_{i=1}^{M-1}\frac{(z_i - z_{i+1})^2}{\alpha^*} + \sum_{i=1}^{M}\frac{(z_i - r_i^*)^2}{\rho^*}\right)\right)\right], \tag{5.37}$$

for which we can use our already derived belief propagation algorithm.

Sampling the Parent and Child Energy Impulses

Here we explain how to sample $\boldsymbol{r}^c \sim \frac{1}{Z} p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho') p(\boldsymbol{r}^c|\{\delta_j^c\})$. Sampling $\boldsymbol{r} \sim \frac{1}{Z} p(\boldsymbol{z}|\boldsymbol{r}, \alpha, \rho) p(\boldsymbol{r}|\mu_{\text{par}}, s_{\text{par}})$ is directly analogous. First we convert $p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho')$ into a large multivariate gaussian, using our result in Equation 5.16, along with Identity F.5 to complete the square, as follows:

$$
\begin{aligned}
p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho') &= \mathcal{N}(\boldsymbol{d}^c|\frac{\boldsymbol{S}\boldsymbol{r}^c}{\rho'}, \boldsymbol{S}) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left[(\boldsymbol{d}^c - \frac{\boldsymbol{S}\boldsymbol{r}^c}{\rho'})^T \boldsymbol{S}^{-1} (\boldsymbol{d}^c - \frac{\boldsymbol{S}\boldsymbol{r}^c}{\rho'})\right]\right) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2\rho'^2}\left[(\boldsymbol{d}^c\rho' - \boldsymbol{S}\boldsymbol{r}^c)^T \boldsymbol{S}^{-1} (\boldsymbol{d}^c\rho' - \boldsymbol{S}\boldsymbol{r}^c)\right]\right) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2\rho'^2}\left[\rho'^2 \boldsymbol{d}^{cT} \boldsymbol{S}^{-1} \boldsymbol{d}^c - 2\rho' \boldsymbol{r}^{cT} \boldsymbol{d}^c + \boldsymbol{r}^{cT} \boldsymbol{S} \boldsymbol{r}^c\right]\right) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2\rho'^2}\left[(\boldsymbol{r}^c - \rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c)^T \boldsymbol{S} (\boldsymbol{r}^c - \rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c)\right]\right) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}\left[(\boldsymbol{r}^c - \rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c)^T \frac{\boldsymbol{S}}{\rho'^2} (\boldsymbol{r}^c - \rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c)\right]\right) \\
&= (2\pi)^{-\frac{M}{2}} |\boldsymbol{S}|^{-\frac{1}{2}} (2\pi)^{\frac{M}{2}} |\boldsymbol{S}^{-1}\rho'^2|^{\frac{1}{2}} \mathcal{N}\left(\boldsymbol{r}^c|\rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c, \boldsymbol{S}^{-1}\rho'^2\right) \\
&= \rho'^M |\boldsymbol{S}^{-1}| \, \mathcal{N}\left(\boldsymbol{r}^c|\rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c, \boldsymbol{S}^{-1}\rho'^2\right).
\end{aligned}
$$

Thus, assuming that r_i^c is drawn either from a gaussian, a mixture of gaussians, or a t-distribution (and thus can be modeled as sampling a variance followed by sampling from a gaussian), we can write

$$
\begin{aligned}
p(\boldsymbol{d}^c|\boldsymbol{r}^c, \alpha', \rho') p(\boldsymbol{r}^c|\{\delta_j^c\}) &\propto \mathcal{N}\left(\boldsymbol{r}^c|\rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c, \boldsymbol{S}^{-1}\rho'^2\right) \prod_{i=1}^{M} \mathcal{N}(r_i^c|m_i, v_i) \\
&= \mathcal{N}\left(\boldsymbol{r}^c|\rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c, \boldsymbol{S}^{-1}\rho'^2\right) \mathcal{N}(\boldsymbol{r}^c|\boldsymbol{m}, \boldsymbol{v}) \\
&\propto \mathcal{N}(\boldsymbol{r}^c|\boldsymbol{c}, \boldsymbol{C}),
\end{aligned}
$$

where $\boldsymbol{v}$ is the diagonal matrix with entries, v_i, each of which correspond to the scalar variance for energy impulse r_i. That is,when sampling the parent energy impulses, $v_i = s_{par}$, and for the child impulses, $v_i \in \{s_{\text{in}}, s_{out}\}$, depending on the value of the corresponding mixture indicator variable, δ_i^c. Similarly, $m_i = 0$ when sampling the child impulses, and $m_i = \mu_{par}$

when sampling the parent impulses, and

$$\boldsymbol{C} = \left((\boldsymbol{S}^{-1}\rho'^2)^{-1} + \boldsymbol{v}^{-1}\right)^{-1} = (\frac{\boldsymbol{S}}{\rho'^2} + \boldsymbol{v}^{-1})^{-1}$$
$$\boldsymbol{c} = \boldsymbol{C}\left(\frac{\boldsymbol{S}}{\rho'^2}\rho'\boldsymbol{S}^{-1}\boldsymbol{d}^c + \boldsymbol{v}^{-1}\boldsymbol{m}\right) = \boldsymbol{C}\left(\frac{\boldsymbol{d}^c}{\rho'} + \boldsymbol{v}^{-1}\boldsymbol{m}\right),$$

where for the parent energy impulses, we would replace $\boldsymbol{d}^c$ by $\boldsymbol{z}$, ρ' by ρ, and $\boldsymbol{S}$ with $\boldsymbol{G}$. The covariance matrix, $\boldsymbol{C}$, will not be diagonal, nor in general have any special properties (nor will the precision matrix), and hence sampling from this multivariate gaussian will be $\mathcal{O}(M^3)$ because of the Cholesky decomposition. We can make the computation for the covariance, $\boldsymbol{C}$, more efficient by noting that we have $\boldsymbol{S}^{-1}$ for free, that $\boldsymbol{v}$ is diagonal, and that $\boldsymbol{v} + \boldsymbol{S}^{-1}$ is tridiagonal, and appealing to the Sherman-Morrison-Woodbury matrix inversion lemma (Identity F.1) as follows

$$\boldsymbol{C} = (\frac{\boldsymbol{S}}{\rho'^2} + \boldsymbol{v}^{-1})^{-1} = \frac{1}{\rho'^2}(\boldsymbol{S}^{-1} - \boldsymbol{S}^{-1}(\boldsymbol{v} + \boldsymbol{S}^{-1})^{-1}\boldsymbol{S}^{-1}).$$

Sampling of the Mixture Model Latent Indicators

We sample $\delta_i^c \sim p(\delta_i^c | r_i^c)$, which is easily accomplished by computing $p(\delta_i^c = 1 | r_i^c)$ and $p(\delta_i^c = 0 | r_i^c)$ and then picking a component proportional to these values. That is, we want to sample

$$\begin{aligned}
\delta_i^c &\sim p(\delta_i^c | r_i^c) && (5.38)\\
&= \frac{p(r_i^c | \delta_i^c) p(\delta_i^c)}{p(r_i^c)} \quad \text{(Bayes' rule)} && (5.39)\\
&= \frac{(\delta_i^c) m_{\text{in}}^c \mathcal{N}(r_i^c | 0, s_{\text{in}}) + (1 - \delta_i^c) m_{\text{out}}^c \mathcal{N}(r_i^c | 0, s_{\text{out}})}{m_{\text{in}}^c \mathcal{N}(r_i^c | 0, s_{\text{in}}) + m_{\text{out}}^c \mathcal{N}(r_i^c | 0, s_{\text{out}})} && (5.40)\\
&\propto (\delta_i^c) m_{\text{in}}^c \mathcal{N}(r_i^c | 0, s_{\text{in}}) + (1 - \delta_i^c) m_{\text{out}}^c \mathcal{N}(r_i^c | 0, s_{\text{out}}). && (5.41)
\end{aligned}$$

Sampling of the Energy Impulse Chain Parameters

For both the parent and child latent traces, we want to sample the corresponding α and β parameters which control how energy flows on our undirected chain. That is, we want to sample $\rho \sim \frac{1}{Z} p(\rho) p(\boldsymbol{d} | \boldsymbol{r}, \rho, \alpha)$ and $\alpha \sim \frac{1}{Z} p(\alpha) p(\boldsymbol{d} | \boldsymbol{r}, \rho, \alpha)$ (and similarly, α', ρ' for the child versions). Since the priors are not conjugate (and even use of Inverse Gamma priors would not provide conjugacy), we use slice sampling with doubling out and shrinkage for this step. This requires belief propagation to evaluate $Z_{\boldsymbol{r}}$ since $Z_{\boldsymbol{r}}$ depends on α and β. To sample α, for

example, we will need to be able to compute the following unnormalized function, $f(\alpha)$,

$$\begin{aligned} f(\alpha) &\equiv p(\alpha)p(\boldsymbol{z}|\boldsymbol{r},\rho,\alpha) \\ &= \frac{1}{Z_{\boldsymbol{r}}}\left[\exp\left(-\frac{1}{2}\left(\sum_{i=1}^{M-1}\frac{(z_i-z_{i+1})^2}{\alpha}+\sum_{i=1}^{M}\frac{(z_i-r_i)^2}{\rho}\right)\right)\right]\frac{1}{\alpha}\mathcal{N}(\log\alpha|h,H), \end{aligned}$$

and similarly so for ρ, α' and ρ'. Note that to calculate $Z_{\boldsymbol{r}}$ we need only run belief propagation in either the forward, or backward direction, but not both, and need not compute the pairwise marginals (conditional or otherwise).

Sampling of the HMM Emission Probability Variances

For the Gibbs step in the HMM emission probability variance, ξ^k, we need to draw samples from

$$\begin{aligned} p(\xi^k|\alpha_h,\beta_h,\boldsymbol{x}^k,\pi^k,\boldsymbol{u}^k) &\propto \text{Inv-Gamma}(\xi^k|\alpha_h,\beta_h)\prod_{i=1}^{N}\mathcal{N}(x_i^k|z_{\tau_i^k}^{w^k}\phi_i^k u_{\tau_i^k}^k,\xi^k) \\ &\propto \text{Inv-Gamma}\left(\xi^k|\alpha_h+\frac{1}{2},\beta_h+\frac{1}{2}(x_1^k-z_{\tau_1^k}^{w^k}\phi_1^k u_{\tau_1^k}^k)^2\right)\prod_{i=2}^{N}\mathcal{N}(x_i^k|z_{\tau_i^k}^{w^k}\phi_i^k u_{\tau_i^k}^k,\xi^k) \\ &\quad\vdots \\ &\propto \text{Inv-Gamma}\left(\xi^k|\alpha_h+\frac{N}{2},\beta_h+\sum_{i=1}^{N}\frac{1}{2}(x_i^k-z_{\tau_i^k}^{w^k}\phi_i^k u_{\tau_i^k}^k)^2\right) \end{aligned}$$

which can be done exactly.

Sampling of the HMM Scaling Spline Parameters

We use slice sampling for the global scaling parameters, u^k, or the scaling spline parameters,μ_j^k, depending on which of these two scaling approaches we are using. For the former, we need sample

$$\mu_j^k \sim \frac{1}{Z}\left(\prod_{i=1}^{N}\mathcal{N}(x_i^k|z_{\tau_i^k}^{w^k}\phi_i^k u_{\tau_i^k}^k,\xi^k)\right)\left(\frac{1}{\mu_j^k}\mathcal{N}(\log(\mu_j^k)|0,\xi_u)\right)$$

where $u_{\tau_i^k}^k$, the interpolated spline values are a function of the spline control points, $\{\mu_j^k\}$ as outlined in Section 4.7. For the case where we just want a single control point (*i.e.*, one global

scaling constant per observed time series), we want to sample from

$$u^k \sim \frac{1}{Z}\left(\prod_{i=1}^{N}\mathcal{N}(x_i^k|z_{\tau_i^k}^{w^k}\phi_i^k u^k, \xi^k)\right)\left(\frac{1}{\mu^k}\mathcal{N}(\log(\mu^k)|0,\xi_u)\right).$$

Sampling of the HMM State Transition Parameters

The HMM time transition parameters, $\boldsymbol{\kappa}^k = (\kappa_1^k, \kappa_2^k, \ldots, \kappa_J^k)$, can be sampled exactly,

$$\begin{aligned}
p(\boldsymbol{\kappa}^k|\boldsymbol{\eta}, \boldsymbol{\tau^k}) &= \frac{1}{Z}p(\boldsymbol{\kappa}^k|\boldsymbol{\eta})\prod_{i=1}^{N}p(\tau_i^k|\tau_{i-1}^k) \\
&= \mathcal{D}(\boldsymbol{\kappa}^k|\boldsymbol{\eta})\prod_{i=1}^{N}\kappa_{\tau_i^k-\tau_{i-1}^k}^k \\
&= \mathcal{D}(\boldsymbol{\kappa}^k)|(\eta_1 + |\{\tau_i^k - \tau_{i-1}^k = 1\}|, \eta_2 + |\{\tau_i^k - \tau_{i-1}^k = 2\}|, \ldots, \eta_J + |\{\tau_i^k - \tau_{i-1}^k = J\}|)),
\end{aligned}$$

and the HMM scale state transition parameters can be sampled in an analogous manner.

Sampling of the Parent Energy Impulse Hyper-Parameters

The parent energy impulse hyper-parameters can be sampled exactly,

$$\begin{aligned}
p(\mu_{\text{par}}|m_{\text{par}}, s'_{\text{par}}, \{r_j\}, s_{\text{par}}) &= \frac{1}{Z}\mathcal{N}(\mu_{\text{par}}|m_{\text{par}}, s'_{\text{par}})\prod_{j=1}^{M}\mathcal{N}(r_j|\mu_{\text{par}}, s_{\text{par}}) \\
&= \mathcal{N}\left(\mu_{\text{par}}|\left(\frac{1}{s'_{\text{par}}} + \frac{M}{s_{\text{par}}}\right)^{-1}\left(\frac{m_{\text{par}}}{s'_{\text{par}}} + \frac{1}{s_{\text{par}}}\sum_{j=1}^{M}r_j\right), \left(\frac{1}{s'_{\text{par}}} + \frac{M}{s_{\text{par}}}\right)^{-1}\right) \\
p(s_{\text{par}}|\alpha_{\text{par}}, \beta_{\text{par}}, \{r_j\}) &= \frac{1}{Z}\text{Inv-Gamma}(s_{\text{par}}|\alpha_{\text{par}}, \beta_{\text{par}})\prod_{j=1}^{M}\mathcal{N}(r_j|\mu_{\text{par}}, s_{\text{par}}) \\
&= \text{Inv-Gamma}\left(s_{\text{par}}|\alpha_{\text{par}} + \frac{M}{2}, \beta_{\text{par}} + \frac{1}{2}\sum_{j=1}^{M}(r_j - \mu_{\text{par}})^2\right).
\end{aligned}$$

Sampling of Child Energy Impulse Hyper-Parameters

Lastly, the child energy impulse hyper-parameters step can be sampled exactly because the prior is conjugate. Recall that we encode mixture component 1 as *in* and 0 as *out*.

$$\begin{aligned} p(m^c_{\text{out}}, m^c_{\text{in}} | m'_{\text{out}}, m'_{\text{in}}, \{\delta_j\}) &= \frac{1}{Z}\mathcal{D}(m^c_{\text{out}}, m^c_{\text{out}} | m'_{\text{out}}, m'_{\text{in}}) \prod_{j=1}^{M} \text{Multinomial}(\delta^c_j | m^c_{\text{in}}, m^c_{\text{out}}) \\ &= \mathcal{D}(m^c_{\text{out}}, m^c_{\text{out}} | m'_{\text{out}} + |\{\delta_j = 0\}|, m'_{\text{in}} + |\{\delta_j = 1\}|) \end{aligned}$$

$$\begin{aligned} p(s_{\text{in}} | \alpha_{\text{in}}, \beta_{\text{in}}, \{\delta^c_j\}, \{r^c_j\})) &= \frac{1}{Z}\text{Inv-Gamma}(s_{\text{in}} | \alpha_{\text{in}}, \beta_{\text{in}}) \prod_{\{j|j=1\}} \mathcal{N}(r^c_j | 0, s_{\text{in}}) \\ &= \text{Inv-Gamma}\left(s_{\text{in}} | \alpha_{\text{in}} + \frac{|\{\delta^c_j = 1\}|}{2}, \beta_{\text{in}} + \frac{1}{2}\sum_{\{j|j=1\}} (r^c_j)^2\right) \end{aligned}$$

$$\begin{aligned} p(s_{\text{out}} | \alpha_{\text{out}}, \beta_{\text{out}}, \{\delta^c_j\}, \{r^c_j\}) &= \frac{1}{Z}\text{Inv-Gamma}(s_{\text{out}} | \alpha_{\text{out}}, \beta_{\text{out}}) \prod_{\{j|j=0\}} \mathcal{N}(r^c_j | 0, s_{\text{out}}) \\ &= \text{Inv-Gamma}\left(s_{\text{out}} | \alpha_{\text{out}} + \frac{|\{\delta^c_j = 0\}|}{2}, \beta_{\text{out}} + \frac{1}{2}\sum_{\{j|j=0\}} (r^c_j)^2\right). \end{aligned}$$

5.5.1 Obtaining Alignments and Normalization After Training

There are a variety of ways that one could obtain an alignment and normalization of the data from the HB-CPM after MCMC has converged:

1. Draw a single HMM state sequence from the posterior, and use this to obtain a single alignment and normalization of the data.

2. Draw a number of samples from the posterior. Each one defines an alignment and normalization in some time reference frame (possibly different from each other). Choose one frame of reference, and map all alignments to it, and then align the data using the adapted alignments, and then average the results.

3. Look for a region of MCMC states that has high likelihood, and appears to be in a region surrounding one mode, and then either 1) use samples from this region as above – by choosing one reference frame and mapping all others to it, or 2) assume their reference frames are similar enough, and simply average the alignments.

4. Look for a region of MCMC states that has high likelihood, and appears to be in a region surrounding one mode, and average the child latent traces in this region. Then use a Viterbi alignment of the data to the averaged latent traces.

Depending on the data set, and the desired end goal, any one of these may be suitable.

5.6 More General Hierarchical Bayesian CPMs

The HB-CPM model we have introduced is appropriate only if we have data from multiple classes. In the case where we seek to align data from only one class, the model would no longer be appropriate because there would be no sense in having a child trace inheriting a parent trace. In this situation, we need to eliminate the child trace layer in the model so that the observed data is now generated directly from the parent trace. We call this the *single-class* HB-CPM.[4]

Furthermore, in the case of only two-class data sets, the HB-CPM, while still appropriate, contains a degeneracy which we may wish to remove. The degeneracy is that the model could choose, with equal probability, to model the first class solely with the parent class, and then model the second class through a difference vector, or, it could do the reverse. It may also choose a number of hybrid situations which lie between these extremes, unless we use hyper-parameter priors to discourage this by making the class components non-exchangeable. However, modifying the prior in this way may not be a strong enough to remove the degeneracy in practice. While this degeneracy is not inherently problematic, it may be desirable to instead force the model, *a priori*, to choose one of these many equally good solutions. We can modify the HB-CPM to allow us to do just that. By explicitly forcing the model to choose one of several equally good solutions, we also have the potential to speed-up convergence of the MCMC.

In this modified HB-CPM, we lock one class in the sense that this class has no child latent trace, and data for this class is generated directly from the parent class. We say that such a class is *locked* in the model (and conversely, a class that is not locked is said to be *unlocked*). There may be data sets with more than two classes where one wishes to also to lock a class. For example, in the three-class LC-UV data set used in Section 5.7.3, one class is, conceptually, on the basis of our knowledge of the experiments, a 'base' class, in that the other two classes are known to be identical to the base class, except for a few differences. That is, we know that it makes sense to model classes two and three as being inherited from the first class. Thus, in such

[4]There might also be domains in which we might model just a single class with a hieararchy, if we believed it made sense for data to be generated in that way.

a case, we may wish to lock the first class. More generally, it is conceivable that one may want to lock several classes, and leave several unlocked. We refer to this most general model, where any number of classes may be locked, as HB-CPM. Hence the model we presented earlier in the chapter as an HB-CPM was but a special case of the more general HB-CPM. Likewise, the single-class Bayesian CPM is in fact a special case of the HB-CPM in which all classes are locked. We now detail the few small changes required to implement this most general model.

This general HB-CPM is easily implemented once the more restricted version of the HB-CPM, as outlined previously in this chapter, has been implemented. As mentioned, only unlocked classes have child traces (and associated mixture component flags, *etc.*). All other structure and parameterization remains identical. As for the MCMC algorithm presented in Section 5.5, only two changes need to be made. The first is trivial, and requires replacing $\boldsymbol{z}^c$ by $\boldsymbol{z}$ for all locked classes, in all equations, except those equations for sampling the parent trace. Sampling of the parent trace is modified as follows.

Let $L \subseteq \{1..C\}$ be the set of locked classes, and $F = \{1..C\} \setminus L$ be the unlocked classes. Then we change Equation 5.36, which shows from which distribution we wish to sample the parent latent trace during the Gibbs sampling, to

$$\begin{aligned}\boldsymbol{z} &\sim \frac{1}{Z} p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho)\left(\prod_{c\in F} p(\boldsymbol{d}^c|\boldsymbol{r}^{\boldsymbol{c}},\alpha',\rho')\right)\left(\prod_{c\in L}\prod_{\{k|w^k=c\}}\prod_{i=1}^{N} p(\boldsymbol{x}^{\boldsymbol{k}}|\boldsymbol{z})\right)\\ &= \frac{1}{Z} p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho)\left(\prod_{c\in F} p(\boldsymbol{z}^c-\boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}},\alpha',\rho')\right)\left(\prod_{c\in L}\prod_{\{k|w^k=c\}}\prod_{i=1}^{N} p(\boldsymbol{x}^{\boldsymbol{k}}|\boldsymbol{z})\right).\end{aligned}$$

Essentially, what we have done, is to continue to use the child latent traces as 'evidence' for the parent trace, for classes that are unlocked, but to instead directly use as evidence, the observed data, for classes that are locked. The first two factors, $p(\boldsymbol{z}|\boldsymbol{r},\alpha,\rho)\left(\prod_{c\in L} p(\boldsymbol{z}^c-\boldsymbol{z}|\boldsymbol{r}^{\boldsymbol{c}},\alpha',\rho')\right)$, multiplied together, were shown in the previous chapter (Equation 5.37) to be equivalent to an energy impulse chain. Furthermore, the third factor, $\prod_{c\in F}\prod_{\{k|w^k=c\}}\prod_{i=1}^{N} p(\boldsymbol{x}^{\boldsymbol{k}}|\boldsymbol{z})$, was shown (Equation 5.29) to reduce to factors of the form $\mathcal{N}(d_j|a_j^c,A_j^c)$, and also shown to be easily incorporated into the belief propagation for the Energy Impulse Chain model (Equations 5.30-5.35). Thus, we already have already developed the machinery to implement this Gibbs step.

5.7 Experiments

We illustrate use of the HB-CPM on two data sets. The first is a two-class data set constructed from some NASA shuttle valve data, and serves as a good illustrative example of some of the behavior of the models, and also nicely illustrates the potential benefits of our model. The second data set a three-class data set arising from a botany study which uses reverse-phase HPLC (high performance liquid chromatography) as a high-throughput screening method to identify genes involved in xenobiotic uptake and metabolism in the model plant *Arabidopsis thaliana*.[5]

We do not here directly compare any results to the EM-CPM or to other algorithms, as no ground truth measure is available to quantitatively assess the different results. However, in the next chapter, we do compare various such algorithms and evaluate their relative benefits by seeing how well they perform in a task of spike-in difference detection for which we have some ground truth measurements, and hence an unbiased way to compare the algorithms.

5.7.1 Initialization

In our experiments, unless otherwise noted, the parent trace was initialized by taking the mean of one smoothed example from each class, where smoothing is performed by a moving average filter with a window that is 20% the length of the vector being smoothed. The child traces were initialized to the initial parent trace. The HMM states were initialized by a Viterbi decoding with respect to the initial values of the other parameters. The scaling factors were initialized to unity, and the child energy impulses to zero. Both data sets used here did not visually appear to require much scaling, and hence we did not use a scaling spline, or HMM scale states, but just a single, global scaling factor for each observed time series.

5.7.2 NASA data

The first data set is the part of the NASA shuttle valve data [26] which measures valve solenoid current against time at a rate of 1ms per sample, with 1000 samples per time series, for some 'normal' runs and some 'abnormal' runs. We subsampled the data by a factor of 7 in time since it was extremely dense and redundant. The raw, subsampled data for each time series can be seen, superimposed upon each other, in Figure 5.6a.

Results from performing posterior inference in the HB-CPM are shown in Figure 5.2, which demonstrates how the outlier mixing proportion changes as MCMC progresses over 40,000 iterations, and also what the parent and child latent traces look like in various regions of a

[5]We thank Sean Cutler and Rachel Puckrin for this data.

single MCMC run. The point of this graph is to show the degeneracy discussed in Section 5.6, and how this leads to the MCMC switching between modes. The parent trace resulting from the second half of the MCMC runs (subplot d) is representative of neither of the classes, but is roughly somewhere between the two, and likewise the parent class in c). However, a) and b) show the parent class dominated by primarily one class or the other. One may not care about the parent classes, but only about how the child classes super-impose. The child traces resulting from each of the various parent latent traces are highly similar in nature, though not identical, and it is not obvious if one is somehow better than the others. The complete log likelihood[6] of each MCMC state for this experiment is shown if Figure 5.3, and the different modes are not visible in this plot, suggesting that the model does not think one is significantly better than the others. Summary plots of other parameters for this experiment are shown in Figures 5.4 and 5.5. Among these, the HMM emission variance, the global scaling, the parent energy impulse mean, and the time state transition parameters are extremely steady. Other parameters are more variable, seeming to track with the modes visible in the outlier mixing proportion.

Figure 5.6 shows the results of using the single-class CPM, as well as the HB-CPM with the second class locked. The single-class alignment, while doing a reasonable job, does so by coercing the two classes to look more similar than they should. This is evident in two particular regions labeled on the graph and discussed in the legend. Essentially a single class alignment causes us to lose class-specific fine detail – the precise information we seek to retain for difference detection. Figure 5.6 shows the complete log likelihood for each of these models, and also the mean of the parent energy impulses.

5.7.3 LC-UV data

The second data set we use to demonstrate use of the HB-CPM is from a botany study which uses reverse-phase HPLC (high performance liquid chromatography) as a high-throughput screening method to identify genes involved in xenobiotic uptake and metabolism in the model plant *Arabidopsis thaliana*.

Components of the analyte in our data set were detected as they came off the LC column with a Diode Array Detector (DAD), yielding UV-visible spectra collected at 540 time points (we used the 280 nm band which is informative for these experiments). We performed background subtraction [14] and then subsampled this data by a factor of four, again because the data was extrememly dense and redundant. This is a three-class data set, where the first class is untreated plant extract, followed by two classes consisting of this same plant treated with com-

[6] By complete log likelihood, we mean the log of the full joint density of all observed and hidden variables in the model.

pounds that were identified as possessing robust uptake *in vivo*, and, hence, when metabolized, provide a differential LC-UV signal of interest.

With this data set we noticed that the HB-CPM would arrive at one of two local minima, and that it did not seem to easily move between them. We used six different initializations for the parent latent trace (setting the child latent traces initially to be equal to the parent) to see the different outcomes. The first five initializations used the average of a smoothed version of the first observed time series in each class, where smoothing was performed with a moving average filter with a window that was respectively 5%, 10%, 50%,90% and 100%, the length of the vector being smoothed. The sixth initialization used a Dynamic Time Warping alignment of all of the observed time series, using the first as a reference. These initializations are shown in Figure 5.8, while Figure 5.9 shows the results from initializations 1 and 4 (as labeled in Figure 5.8). These two initializations are representative of the two local minima found by the MCMC. Initialization 1 had lower probability density than initialization 4. Initialization 3 produced results resembling those of 1, while initialization 2, 5 and 6 resembled 4. The initializations resulgint in higher probability density (2, 4, 5, 6, which had, respectively averages of their complete log likelihood over the last 10,000 MCMC states of -3221, -3234, -3268, -3243) appear visually to be superior to the lower probability initialization runs (1, 3, with average complete log likelihoods over the last 10,000 MCMC states of respectively -3400, -3467) in that the large peaks on the right-hand side appear to be properly aligned in the higher probability solutions. Also, the 'better' solution tended to have a lower outlier mixing proportion for class 1. It's interesting to note that initialization 3 and 4 were very similar, yet arrived at two different local minima, suggesting that chance alone affected the outcomes, rather than a strong bias from the initialization of the latent traces. We also continued to run the MCMC from initialization 1 for another 20,000 iterations (for a total of 40,000), but the MCMC never switched into the 'good' mode.

We also ran the Single-Class HB-CPM, and the HB-CPM with class 1 locked (this is the 'base class' in that it is the experiment that had the untreated plant extract, whereas classes 2 and 3 had this same plant, but treated with a drug), on this same LC-UV data set, using the exact same six initializations. The single class runs appeared not to have a local minima problem, and had overall similar probability masses in the sampled posteriors (averaging the complete log likelihood over the last 10,000 MCMC states produced values of -4549, -4573, -4583, -4627, -4597 and -4567) and visually did not have any dramatic differences in the parent traces (such as seen with the HB-CPM). The HB-CPM with class 1 locked, surprisingly, had the exact same local minima problems as the less constrained HB-CPM, and in fact, produced the same two types of local minima, and in the same pattern with respect to the initializations. The average of the complete log likelihood over the last 10,000 runs were respectively -3188, -2974, -3085,

-3050, -3029, and -2895, for initializations 1-6. Again, initializations 1 and 3 were visually inferior. This then suggests that the different local minima likely do result directly from the different parent trace initializations, and also that using the more constrained HB-CPM with one class locked may not systematically help us alleviate local minima problems. Note also that the 'smart' DTW initialization did not seem to offer any advantage over naive initializations.

Figure 5.10 gives an overview of how the Single-Class HB-CPM results contrast with those of the HB-CPM, while Figure 5.11 zooms in on a particular area of interest to highlight how subtle differences can be detected by the HB-CPM, but not by a single-class alignment scheme. As with the NASA data set, a single-class alignment coerces features across classes that are in fact different to look the same, thereby preventing us from detecting them. The results shown are from the last 10,000 MCMC states, and one might think that for the single-class model, we are blurring over multiple modes (and hence the large standard deviation lines). However, even narrowing the range of states down to a much smaller range of just 100 MCMC states, over different regions of the last 10,000 MCMC states does not visually produce significantly different results. These larger standard deviation lines are simply a result of a single-class model trying to model multi-class data, which it cannot do in a reasonable manner.

Recall that this data set consists of a 'no treatment' plant extract, and two 'treatments' of this same plant. Though our model was not informed of these special relationships, it nevertheless elegantly captures this structure by giving almost no energy impulses to the 'no treatment' class, meaning that this class is essentially the parent trace, and allowing the 'treatment' classes to diverge from it, thereby nicely matching the reality of the situation. In effect, the HB-CPM has deduced appropriately which class to lock.

5.8 Discussion

We have presented a class of Bayesian, hierarchical CPMs for alignment and normalization of multi-class data. The model, while susceptible to local minima, appears nevertheless to have the ability to capture meaningful, elegant solutions. Experiments suggest that running many parallel MCMC chains is more effective than running one long chain in overcoming the local minima problem, and also that using the more constrained HB-CPM which locks one of the classes does not necessarily alleviate the problem. Additionally, it appears clear that use of the multi-class model could be advantageous over the analogous single-class model (and presumably over any single-class model) because it does not tend to blur out small class-specific differences, which are precisely the kind of differences we are interested in maintaining.

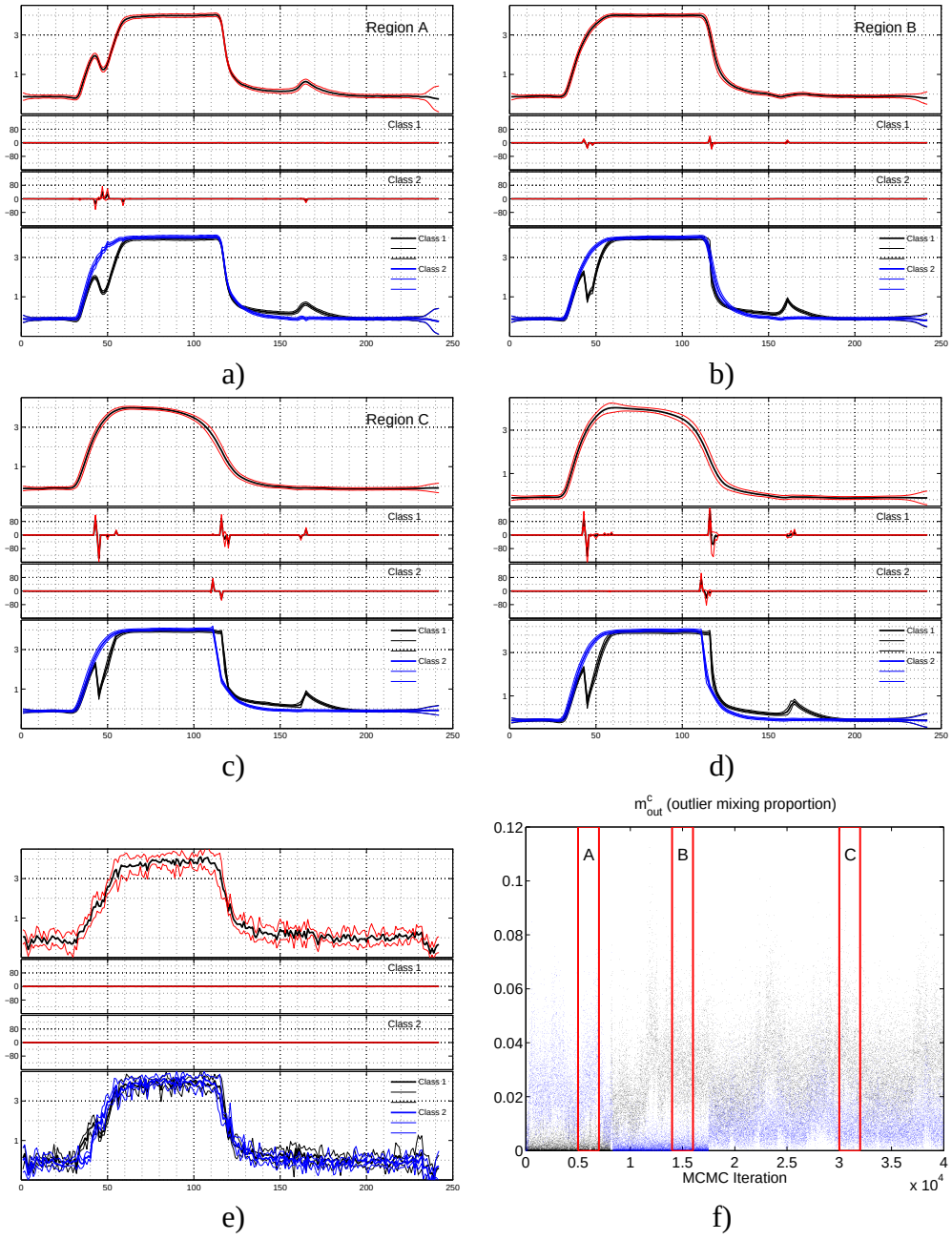


Figure 5.2: Results from using the HB-CPM on the three-class NASA data set. f) shows the value of the outlier mixing component, for each of the two classes, over 40,000 MCMC iterations. The remaining plots all show the parent and child latent traces, and the child energy impulses for each class. These are displayed by showing the average of each in a bolder line, and one standard deviation lines in a thinner line of the same colour. The average is over different regions: a), b) and c) are averaged over regions A,B,C respectively, as annotated on subplot f), while d) shows the average over the last half of the iterations (20,000-40,000) and e) shows the average over the first 10 MCMC iterations.

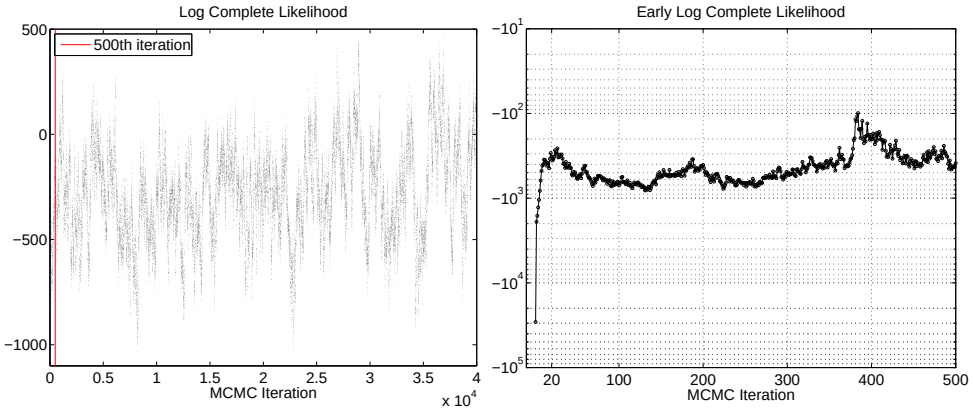


Figure 5.3: The complete log likelihood over 40,000 iterations of MCMC on the NASA data set using the HB-CPM. The right figure shows a zoom in on the first 500 iterations, showing how rapidly the complete log likelihood increases early on.

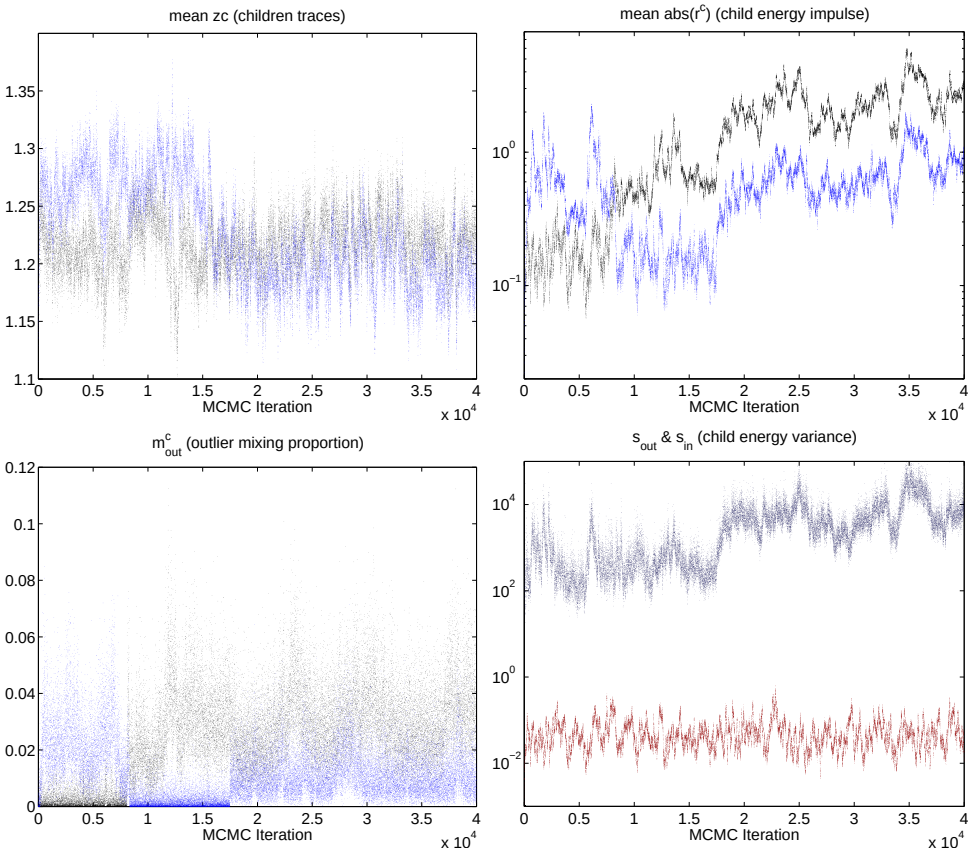


Figure 5.4: Summary plots of various parameters in the model, over 40,000 iterations of MCMC on the NASA data set using the HB-CPM.

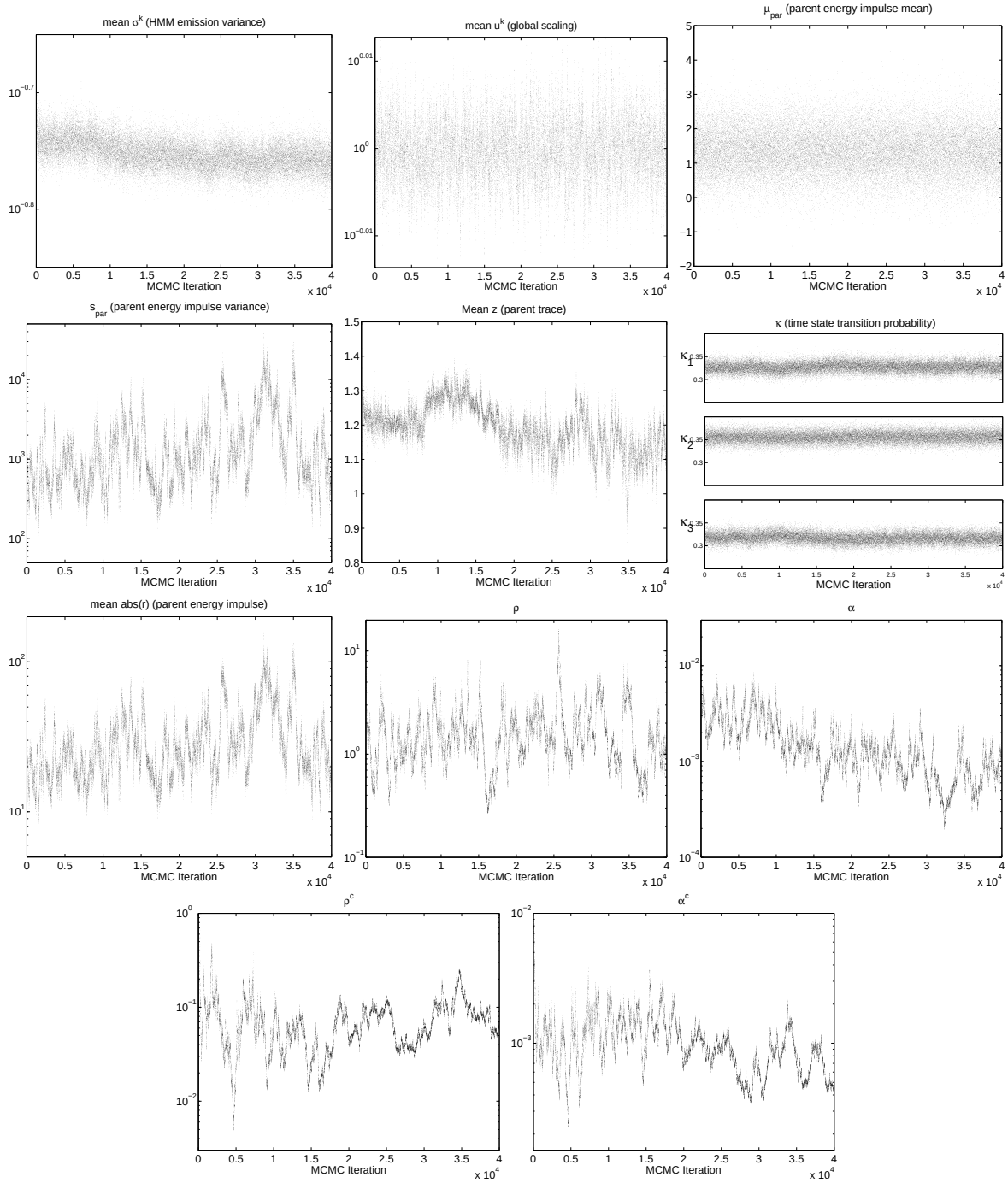


Figure 5.5: Summary plots of various parameters in the model, over 40,000 iterations of MCMC on the NASA data set using the HB-CPM.

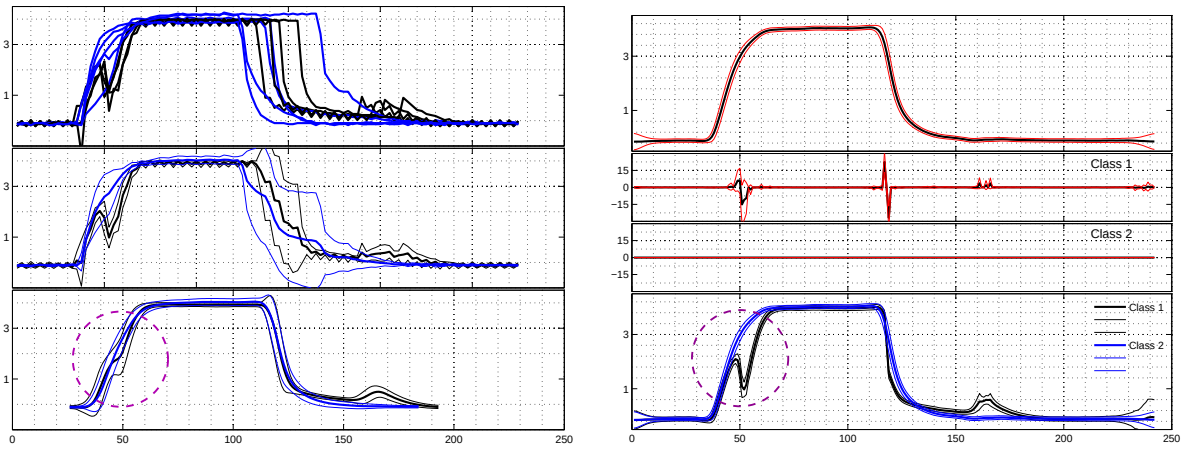


Figure 5.6: Nine time series from the NASA valve solenoid current data set. Four belong to a 'normal' class, and five to an 'abnormal' class. On all figures, the horizontal axis is time (latent time, for figures of latent traces or of observed time series aligned in latent time), and the vertical axis, current amplitude. a) Top: The raw, unaligned data. Middle: Average of the unaligned data within each class in thick line, with the thin lines showing one standard deviation on either side. Bottom: Average of the aligned data (over MCMC samples) within each class, using the single-class alignment version of the model (no child traces), again with one standard deviation lines shown in the thinner style line. b) Mean and one standard deviation over MCMC samples using the HB-CPM with class 1 locked. Top: Parent trace. Middle: Class-specific energy impulses with the top-most showing the class impulses for the less smooth class. Bottom: Child traces superimposed (where the child class for class 1 is taken to be the parent). Note that the single-class alignment coerces the feature highlighted with the dashed circle to be inappropriately collapsed in time for class 1. Also, the overall width of the main broad peak in class 2 is inappropriately narrowed. In contrast, the HB-CPM models these features correctly.

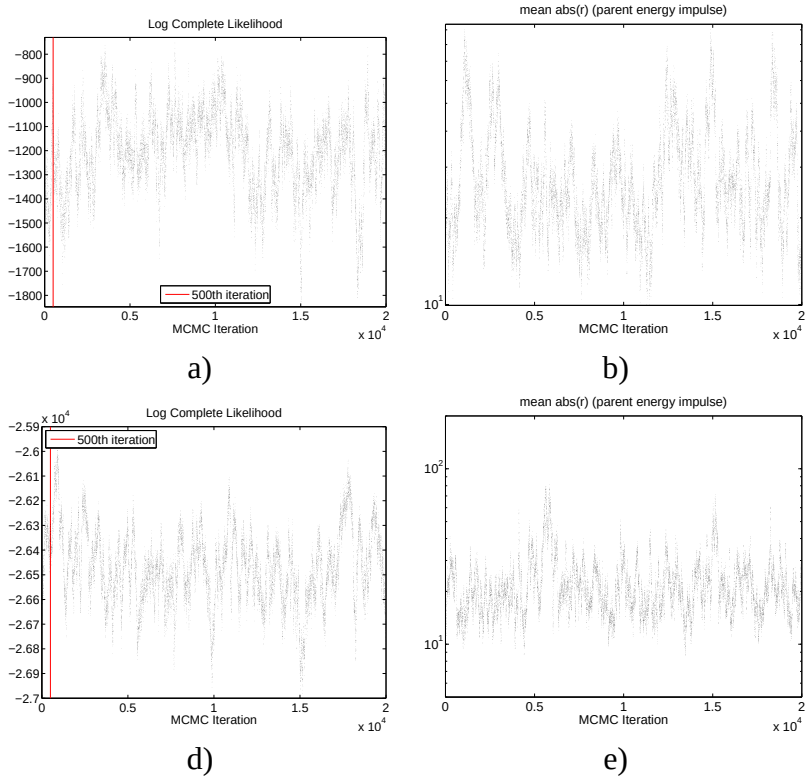


Figure 5.7: The log of the joint density, and the mean parent energy impulse over 40,000 iterations of MCMC on the NASA data set using a), b) the HB-CPM with class two locked, and c), d) the single-class HB-CPM.

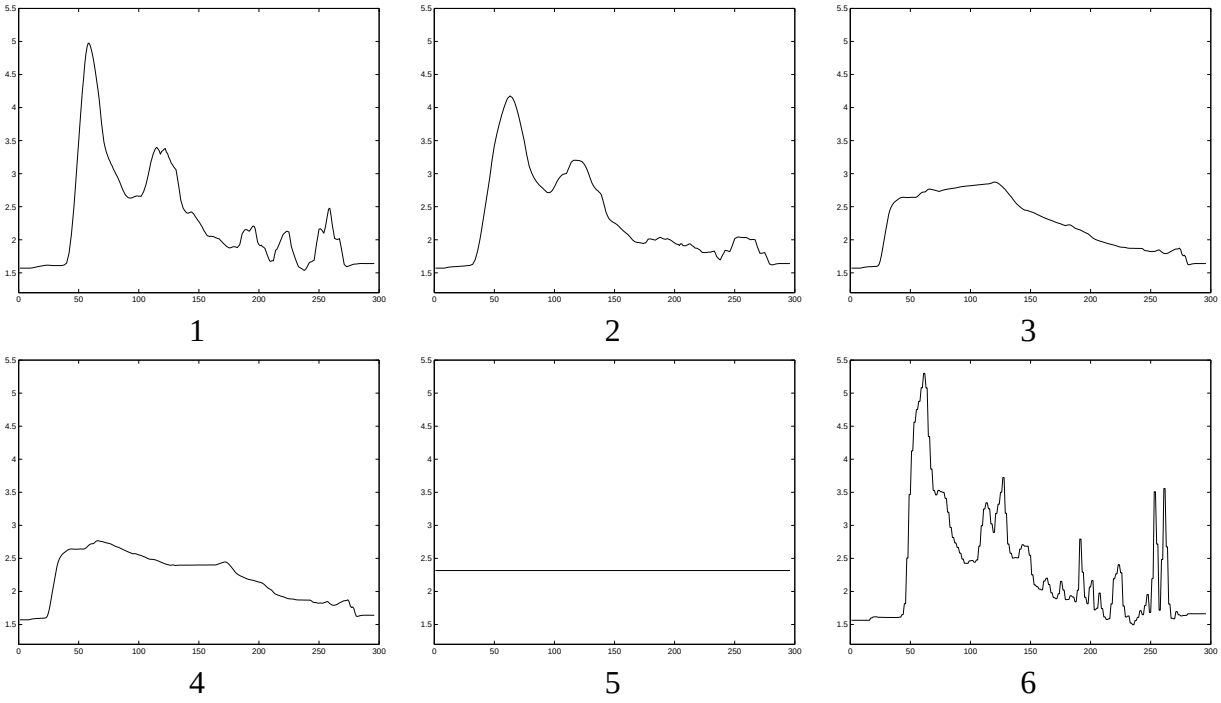


Figure 5.8: Six different parent latent traces used to initialize the MCMC for the LC-UV data set. 1-5 show the result of increasingly more smoothing of one observed time series from each class, and then averaging them, while 6 shows an initialization resulting from use of DTW on all the observed times series.

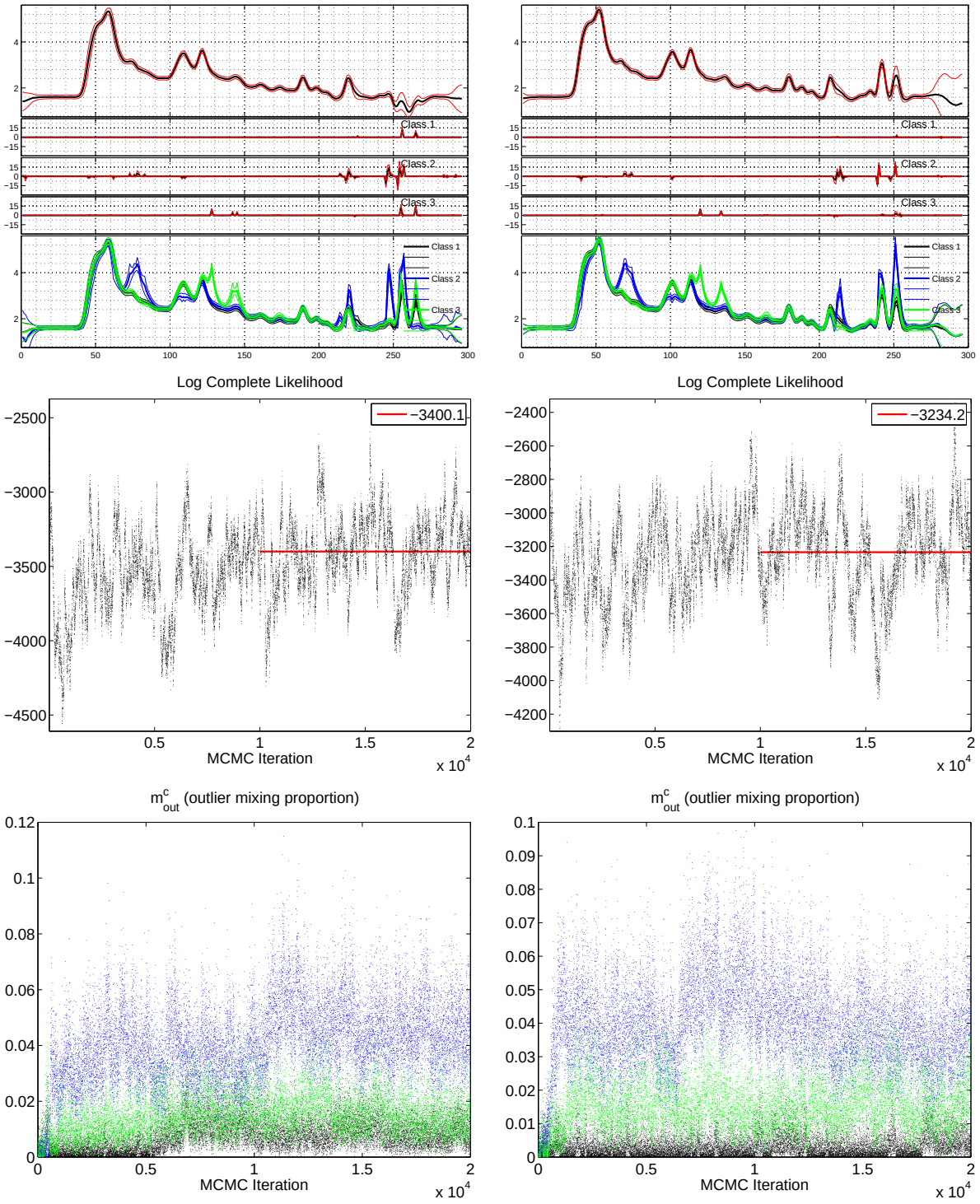


Figure 5.9: Results from initialization 1 on the left, and from initialization 4 on the right, after 20,000 iterations of MCMC. Note that the results from initialization 4 do properly deal with the large peaks on the right hand side (at least what appears to be properly), while the lower probability solution from initialization 1 does not. The average log of the complete log likelihood (i.e the joint density) over the last 10,000 runs is shown in the legend.

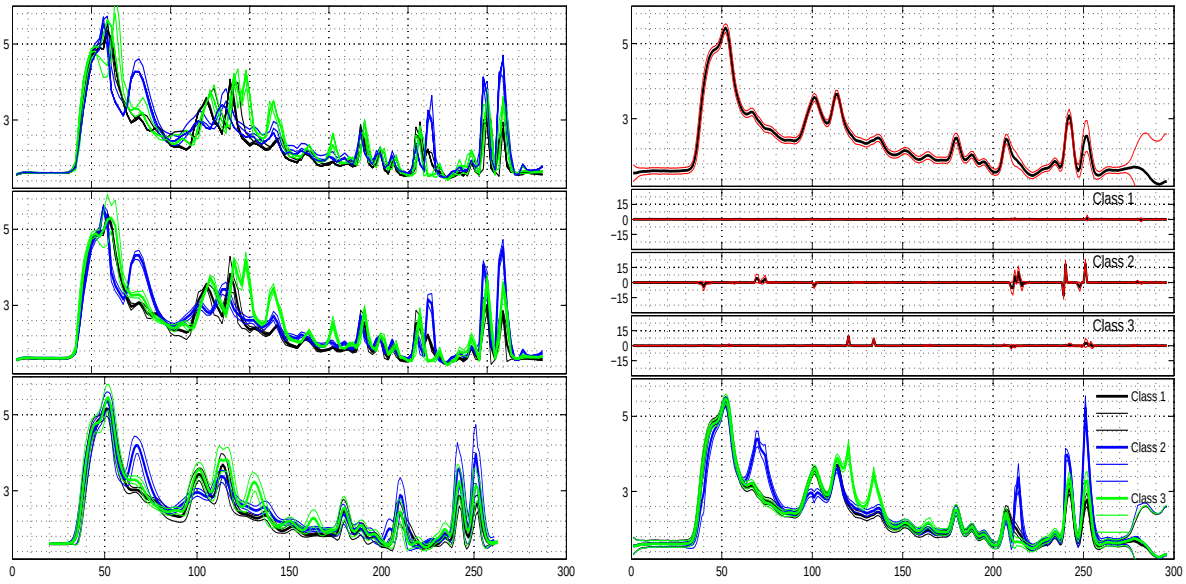


Figure 5.10: Seven time series from each of three classes of LC-UV data. On all figures, the horizontal axis is time (latent time, for figures of latent traces or observed time series aligned in latent time), and the vertical axis, log of UV absorbance. a) Top: The raw, unaligned data. Middle: Average of the unaligned data within each class in thick line, with the thin lines showing one standard deviation on either side. Bottom: Average of the aligned data within each class, using the single-class HB-CPM, again with one standard deviation lines shown in the thinner style line. b) Mean and one standard deviation over MCMC samples using the HB-CPM model. Top: Parent trace. Middle: Class-specific energy impulses, with the top-most showing the class impulses for the 'no treatment' class. Bottom: Child traces superimposed. For the MCMC, the average and standard deviation is with respect to the last 10,000 MCMC states over a 20,000 iteration run, and results are shown from initialization 4 of the latent traces.

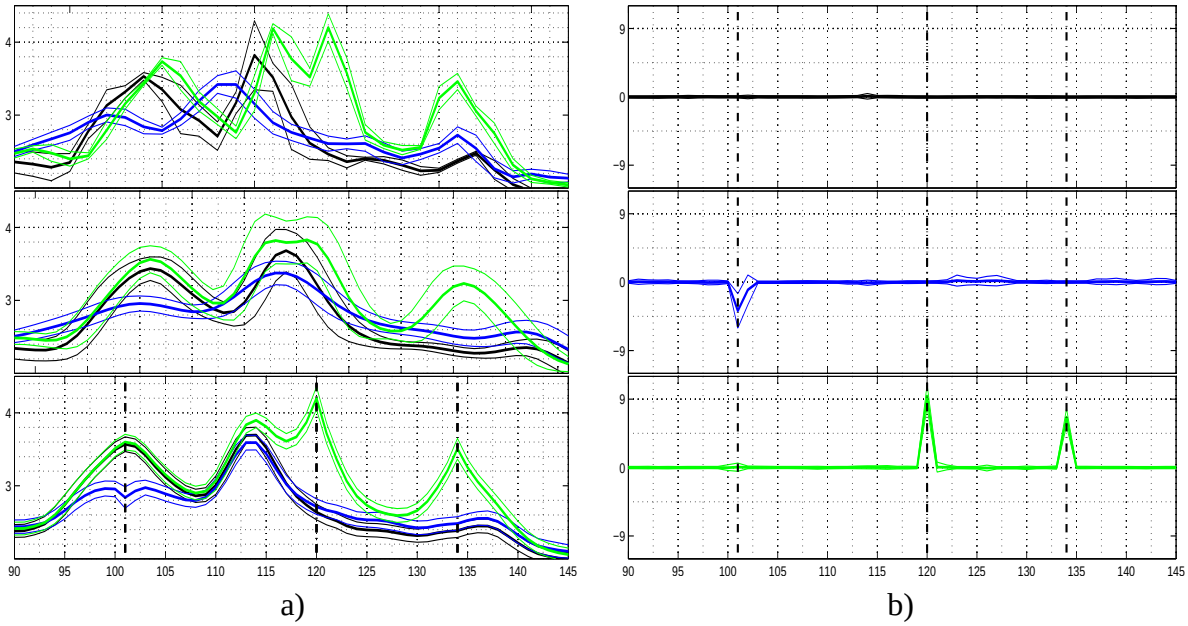


Figure 5.11: a) A zoom in of data displayed in Figure 5.10, from the region of time 100-150 (labeled in that figure in latent time, not observed time). Top: mean and standard deviation of the unaligned data. Middle: mean and standard deviation of the single-class alignment, over the last 10,0000 MCMC samples. Bottom: mean and standard deviation of the child traces from the HB-CPM. A case in point of a difference that could be detected with the HB-CPM and not in the raw or single-class aligned data, is the difference occurring at time point 120. b) The corresponding mean and standard deviation of the child energy impulses, with dashed lines showing correspondences with the child traces in the bottom panel of a).

Chapter 6

Difference Detection in LC-MS Data for Protein Biomarker Discovery

Synopsis: In this chapter we present a technique for discovering differences in protein signal between two classes of samples of LC-MS serum proteomic data without use of tandem mass spectrometry, gels, or labeling. We test our technique on a controlled, realistic experiment (spike-in, serum biomarker discovery) which is therefore verifiable. This set-up allows us to assess different approaches to the alignment problem, by comparing precision-recall curves built from knowledge of the spike-in ground truth. We are thus able to contrast the performance of Dynamic Time Warping,[1] the EM-CPM and the HB-CPM. Refer to Chapter 3 for a general introduction to LC-MS proteomics and the terminology from that area used in this chapter.

6.1 Introduction

Much proteomic work to date has concentrated on the use of tandem mass spectrometry (MS/MS) in which the sequences of some peptides in serum are found, and analysis is based on a list of these peptides [40]. While such an approach has been successful for problems such as building protein catalogues, this approach has shortcomings when applied to biomarker discovery, in which the goal is to find what differences exist between two classes of samples (*e.g.*, cancer versus healthy). We refer to this as the problem of *difference detection*. Since the list of peptides derived from tandem mass spectrometry experiments is never complete for complex mixtures, there are likely to always be protein signals of interest that are missed [5, 66]. One can avoid this problem by looking at all of the raw data from the LC-MS experiment rather

[1] We did not compare to COW (Correlated Optimized Warping – see Chapter 2) here since the implementation available was not easily adaptable to using more than scalar features.

than just the list of sequenced peptides, without reliance on concurrent MS/MS sequencing. Once regions (time, *m/z*) that are different have been identified, these can be characterized by MS/MS [86]. While this chapter provides a statistical/computational technique for biomarker discovery from LC-MS data, such a step is but one of many in the grand goal of true biomarker discovery, which must address issues ranging from sample collection to a feedback loop with basic biology for validation. For more discussion of these issues see [45, 53] and references therein.

Our goals here are to show that in a realistic experiment using human blood serum, with a controlled spike-in of known peptides, with a low-precision mass spectrometry instrument, we can elicit differences of interest between two classes, for which the ground truth is known. Additionally, we make use of this experimental set-up to compare and contrast various alignment algorithms. Parts of this work were first reported by us in [50].

6.2 Related Work

Recently, several approaches have been published for the problem of difference detection between two classes of LC-MS samples, without use of MS/MS or chemical/isotopic labeling. These approaches typically involve a suite of algorithms, starting from data pre-processing such as filtering, background-subtraction and alignment along the LC time axis, and then move on to one of two approaches i) detect and quantify peaks, then do a differential peak analysis (*e.g.,* [5, 75]), or ii) do a peak-free, 'signal-based', differential analysis to find regions of interest that can then be further studied (*e.g.,* [86, 66]), as we do here. In the former approach, peak detection is carried out on one LC-MS run at a time, without the advantage of leveraging across samples, so features could be lost that might be captured by a signal-based approach, since the latter need not first capture discrete features within each LC-MS run [86, 46]. The relative merits of each approach are likely closely linked to the precision of the MS instrument being used. In [5, 75], high-precision instruments are used with a peak-based approach, while [86, 66], with lower-precision data, use a signal-based approach.

Prakash *et al.* [66] use a signal-based approach to the problem of looking for one spike-in peptide added to a base mixture of 4 peptides – a very simple mixture compared to a more realistic setting such as human blood serum. With just their simple mixture of five proteins, Prakash *et al.* note that even "these mixtures are surprisingly complex" and that "thousands of peptide-like features are observed". They align their data, and nicely characterize their scoring function which quantifies the number of shared peaks between mass spectra, but they stop short of using any statistical tests when doing difference detection, relying on differences in ion amplitude (shown in [86] to be sub-optimal), and they do not quantify their difference de-

tection results, using instead qualitative plots. America *et al.* [5] use a peak-based approach with 2D LC-MS to find differences between tomatoes at various stages of ripening. While their suite of algorithms appears to be very solid, they do not show how well they do relative to any ground truth. A very comprehensive study was conducted in [86] where several different spike-in mixtures, consisting of tryptic digests of roughly 8 proteins, were added to a base mixture of 1000 known tryptic peptides, and differences detected. Precision-Recall curves were used to evaluate detected differences with respect to ground truth, but it is not clear how their analysis would generalize to human serum, which we tackle here. Silva *et al.* perform a detailed exploration of spike-in proteins to human blood serum using data from a very high-precision MS instrument [75]. In this study, we show the ability to detect known spike-in proteins in human serum with data from a much lower-precision instrument – the more common 'workhorse' type mass-spectrometer that is widely available and affordable to the proteomics community at present.

6.3 The Data

We use a spike-in LC-MS data set where class 1 consists of a base mixture of human serum, and class 2 consists of this same base mixture of serum, along with three known peptides which were 'spiked-in'.[2] We show later that the amount of spike-in was not trivially easy to find, as was intended by the design of this experiment. For each data set, serum-only runs were alternated with the spike+serum runs (seven of each class) to control for potential time and order-dependent biases, and all samples were run on the same chromatography column.

[2] *Laboratory Methods:* Frozen human serum was thawed on ice. A 2 μl aliquot ($\sim$160 μg protein) was denatured and reduced for 30 min on ice by adding 5 volumes (10 μl) of 8M urea (pH 8.5), 1 mM fresh DTT. This was followed by 5 volumes (10 μl) of acetonitrile and 50 μl of 100 mM Ammonium Bicarbonate, 1 mM $CaCl_2$. The samples were then incubated and digested for two days with 10 μl of covalent trypsin beads (Poroszyme; Applied Biosystems) at 30°C with rotation. Prior to analysis, the samples were further diluted with 70 μl of Buffer A (5%ACN, 0.05%HFBA). For the spiking runs, the reference peptides standard was added at 1.0 pmol (Peptide Calibration Standard #206195, Bruker Daltonics Inc.) to the sample immediately prior to LC-MS. 5 μl (0.2 nmol total protein) of the diluted serum samples were analyzed using standard capillary scale LC-MS profiling methods as described in [68]. Briefly, a quaternary HPLC pump was interfaced using the electrospray ionization method to an LCQ quadrupole ion trap tandem mass spectrometer (Thermo Finnigan; San Jose, CA). The samples were loaded onto a 150 μm i.d. fused silica capillary micro-column (Polymicro Technologies; Phoenix, AZ) bearing a fine nozzle created with a P-2000 laser puller (Sutter Instruments; Novato, CA) and packed with 8 cm of 5 μm Zorbax 300SB-C18 resin (Agilent Technologies, Mississauga, ON, Canada). The ion trap mass spectrometer was operated in dual cycling mode, cycling from precursor scanning to dynamic data-dependent MS/MS scan mode, even though we did not use the MS/MS data in our experiments, and this mode in fact degrades the quality of the data. The 'raw' data from these experiments was data that had been centroided by the mass spectrometry instrument software. Data was generated in the laboratory of Andrew Emili at the University of Toronto.

6.4 Approach to Detection

Although in the last chapter we developed a principled multi-class approach to alignment and detection, for LC-MS data, the space of interest, (*i.e.*, that in which we wish to find differences), of *time* $\times$ *m/z*, is extremely large (our data matrix is approximately of dimensions 500×2400). To use our multi-class HB-CPM on this data set to find differences would require using 2400 *m/z* bins (or say, half this, depending on the desired precision of the differences detected), which with current computational power, would take an onerously long time to run. To side-step this problem (and, unfortunately, side-step the elegance of the solution), one could instead use fewer bins and use only the multi-class alignment portion of the algorithm, followed by an 'unrolling' of the alignment to the original space (see Section 4.6.1), and then subsequently perform difference detection using some independent method in this target space. However, it turns out that for the data set on hand, for which we have access to ground truth, use of a single-class alignment works sufficiently well, because the differences are actually quite small, and of a structure which does not interfere with alignment. In particular, differences are sparse within any given time slice, and within any given *m/z* slice, and hence do not seem to be problematic when a single-class alignment scheme is used. Thus the comparisons in this chapter focus on different single-class alignment schemes. We now explain how we perform difference detection after (and independently from) alignment.

6.4.1 The Data Matrix

After applying an alignment to all samples from each of two classes (*e.g.*, cancer versus not cancer), we have in hand a set of *aligned data matrixes*. That is, we have, for each LC-MS run k, a matrix of data, $\boldsymbol{\mathcal{M}}^k$ where $\mathcal{M}^k_{\tau,i}$ indexes the ion abundance at the τ'th aligned time point for the i'th *m/z* bin in the k'th observed LC-MS run. Note that prior to alignment, all *m/z* values are quantized so that each original *m/z* value maps to one of 2400 quantized values (where each quantized values spans a width of $\frac{1}{2}$*Th* (see Section 3.3.1). These quantized values are retained throughout all other processing (similarly done in [68, 86, 48]). In our data set, we thus had roughly 500 rows corresponding to 500 time points, and 2400 columns corresponding to quantized *m/z* values spanning 400-1600 *Th* (which are then further reduced to a smaller number of bins for the purpose of alignment. When we construct, say B bins for the purposes of alignment, we do so by choosing them in such a way that the *m/z* values contained in each bin are contiguous, and such that the total ion count in any one bin, summing all of the *m/z* values assigned to that bin, are approximately constant. We might alternatively have used, say, the top B principle components from PCA, or used some other dimensionality reduction algorithm, but we did not pursue this avenue.

As in Chapter 4.2, we coarsely aligning and scale each time series as follows as a pre-processing step: First we translate each time series so that the center of mass of each time series is aligned to the median center of mass over all time series. Then we scale the abundance values such that the sum of abundance values in each time series is equal to the median sum of abundance values over all time series.

6.4.2 Smoothing of Residual 'Mis-Alignment'

Even for very good alignments, one can usually observe regions in two different LC-MS runs that appear to correspond to each other, but that are not perfectly and completely overlapping. This may be owing to the fact that we have required all *m/z* bins to be warped together (*i.e.*, take on the same alignment mapping); if different molecules in the chromatography column were affected differently, such a constraint could produce sub-optimal results. In an effort to correct this residual 'mis-alignment', we have found it useful to apply a small local smoothing in time to each data matrix, after alignment has been performed. Additionally, smoothing in *m/z* could help to overcome any possible rounding issues in the *m/z* quantization. This smoothing is implemented by convolving a 2D (time and *m/z*) Hamming filter with each of the data matrixes. We estimate the optimal amount of smoothing to be used (corresponding to the width and height of the filter) by using the following intuition.

If we do not smooth at all, then some *m/z* bins across experiments will not be in good correspondence with one another, and likewise, some LC times will not be in good correspondence with one another. Thus, if we use one of these LC-MS runs and ask how well it 'predicts' the other replicates in this class, it will 'predict less well' because of the regions of poor correspondence. Going to the other extreme, if we heavily smooth this same LC-MS run, and then ask how well it 'predicts' the others, its signal will be completely smeared out, and it will not 'predict' them well. In between these two regimes lies an amount of smoothing for which this one LC-MS run optimally 'predicts' the others. More formally, we use the Total Variation Distance (TVD) between a smoothed LC-MS run and the other runs to see how well it 'predicts' the other runs. The TVD is simply a symmetric measure of how much two probability distributions diverge, and is formally defined, for discrete distributions, $p_1, \ldots, p_N$, and $q_1, \ldots, q_N$, to be $TVD(p_1, \ldots, p_N, q_1, \ldots, q_N) = \frac{1}{2}\sum_{i=1}^{N} |p_i - q_i|$. To be able to use the TVD, we pretend that each data matrix, $\mathcal{M}^k$ is a distribution by forcing its components to sum to one. Formally, our method operates as follows:

1. normalize each $\mathcal{M}^k$ to sum to 1

2. apply Hamming smoothing to one LC-MS run k'

3. measure how 'close' this smoothed run, k', is to every other non-smoothed run in the same class by computing the TVD

4. do this for each run in turn and sum together the TVD values

5. repeat for variously sized Hamming filters.

Too little smoothing produces larger TVDs because of mismatched peaks, while too much smoothing washes out the signal so that the smoothed run is 'further' from the non-smoothed signals and the TVD is greater. In between these regimes lies the 'optimal' amount of smoothing. Note also that we perform this TVD cross-validation separately for each class, and then average the results.

An example of the outcome of this cross-validation experiment is shown in Figure 6.1. Note that we only use odd widths (where width spans *m/z*) of the filter because for very narrow widths, the filter looks considerably different for even and odd widths due to the discrete nature of the filter, and we found that odd width filters had superior performance. Note that a filter width of 3 spans only 1.5 *Th* because of our bin quantization.

Different alignment methods produce different density reference time frames. For example, the CPM methods tend to produce reference frames which are twice as dense in time as the original time series (because we use a latent time space which is twice as dense), whereas the implementation of DTW that we use here uses an interpolation scheme which maps all of the DTW reference time frames back to one which is approximately the same density as the original time series. Thus this alone would cause us to expect that different amounts of smoothing would be optimal for different algorithms (in addition to the different quality of the alignment algorithms which would also affect the optimal amount of smoothing).

For our experimental data, we optimized the amount of smoothing by computing the TVD cross-validation using 4 *m/z* bins, and all 14 replicates (7 per class). We find that 10 time units and 0 *m/z* units were optimal for the CPM (using the EM-CPM) and that 15 time units and 0 *m/z* units were optimal for DTW (using the DTW regularizations setting which produced the best alignments – min/max slope of 0.80/1.25.[3]) This result is slightly counter-intuitive, in that one might think that the CPM should require more smoothing since its alignment lies in a higher density reference time frame. However, the quality of the alignment also plays a large factor here (poorer quality alignments will tend to prefer more smoothing). The fact that the DTW provides worse quality alignments is also reflected in the fact that the no smoothing TVD for DTW is higher than that of the EM-CPM. Also note that both DTW and the EM-

[3]The same optimal smoothing is found in using DTW with min/max slope of 0.8/1.25 with time series 7 as a reference template. We tried to perform this TVD experiment on the most promising regularization settings of DTW.

CPM prefer no smoothing on the *m/z* axis. This corroborates the general consensus that mass spectrometers produce more reproducible results than chromatography columns (as noted in Chapter 3). Also, the fact that both DTW and EM-CPM are in agreement about the amount of *m/z* smoothing makes sense since alignment in time does nothing to the *m/z* axis.

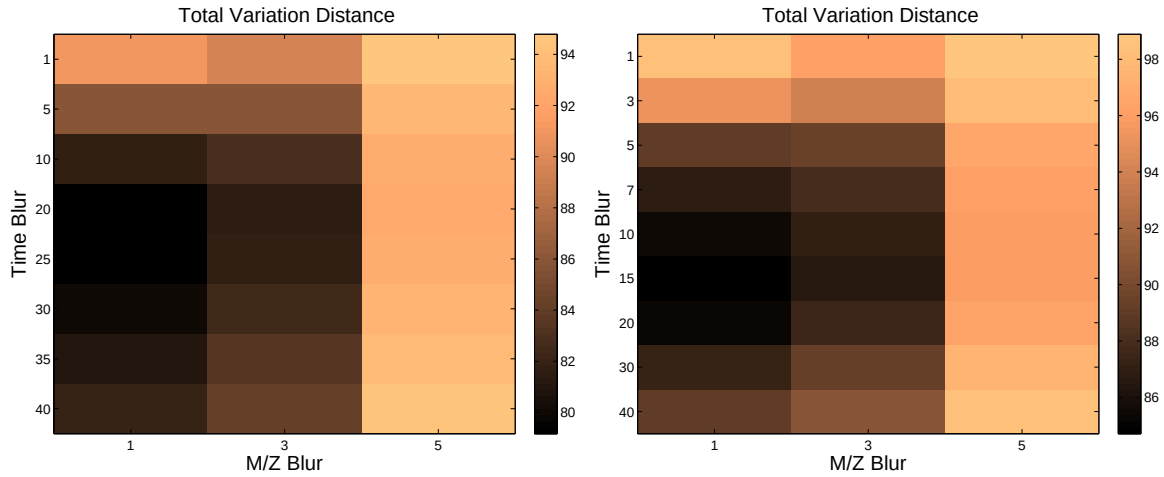


Figure 6.1: Results of TVD cross-validation to find the optimal amount of smoothing to correct for residual 'mis-alignment'. Left shows the results for the EM-CPM (optimal at 10,0). Right shows the results for DTW with min/max slope of 0.86/1.15 (optimal at 15,0), using time series 6 as a reference template. The lower the TVD, the better the result.

6.4.3 A Spatial Test Statistic for Difference Detection

We devise a simple statistical test, based on a t-statistic which is applied to the data set. First we compute a (Welch) t-statistic at each *(time, m/z)* in the set of data matrixes, $\{\mathcal{M}^k\}$. Let $C^r_{\tau,i}$ be the class mean over all samples in class r at the τ'th time point and i'th *m/z* bin, *i.e.*, $C^r_{\tau,i} \equiv \frac{\sum_{\{k|k\in r\}} \mathcal{M}^k_{\tau,i}}{N_r}$, where N_r is the number of samples in each class. Similarly, let $V^r_{\tau,i}$ be the class variance over all samples in class r at the τ'th time point and i'th *m/z* bin, *i.e.*, $V^r_{\tau,i} \equiv \frac{\sum_{\{k|k\in r\}} (\mathcal{C}^k_{\tau,i} - \mathcal{M}^k_{\tau,i})^2}{N_r - 1}$. Also, let $S'_{\tau,i} \equiv \sqrt{V^1_{\tau,i}/N_1 + V^2_{\tau,i}/N_2}$ be the pooled standard deviation. Then we calculate a signed t-statistic, $t_{\tau,i} \equiv \frac{C^1_{\tau,i} - C^2_{\tau,i}}{S'_{\tau,i}}$. If $V^1_{\tau,i} = 0$ and $V^2_{\tau,i} = 0$ and $C^1_{\tau,i} = C^2_{\tau,i}$ then[4] we set $t_{\tau,i} = 0$. After this point-wise statistic is calculated, we modify it to account for the fact that we know, *a priori*, that signal of interest in this type of data will span more than one time point, because elution off the LC column is not instantaneous, and possibly more than one *m/z* bin, because of isotope shoulders. We therefore calculate what

[4]This occurs when there is no ion abundance in any of the samples at this time and *m/z*. Also, in our data sets there were no cases where $S'_{\tau,i} = 0$ other than this particular case.

we call a *spatial* t-statistic, $t'_{\tau,i}$, by performing 2D smoothing on the matrix consisting of t-statistics. This is implemented by convolving the matrix of values $t_{\tau,i}$ with a 2D Hamming filter corresponding to our prior beliefs about peak sizes of interest. For CPM-based experiments, we used a Hamming filter of length 10 in aligned time (*i.e.*, approximately 5 in experimental time) and length 3 *m/z* bins (*i.e.*, 1.5 *Th*), while for DTW, we used a Hamming filter of length 5 in aligned time (*i.e.*, approximately 5 in experimental time) and length 3 *m/z* bins (*i.e.*, 1.5 *Th*).

Note that regions of interest which are smaller than this filter can still be captured if their signal is strong enough. That is, this method does not immediately throw out smaller peaks, as is the case in [86] where a hard threshold of length in time is used, it only requires that such regions provide more evidence for themselves. Without this last step of moving to a spatial t-statistic, we observed that the ability to reliably detect differences across classes was diminished, as can be observed in Figure 6.2. While smoothing the t-statistic removes the possibility of using corresponding t-statistic p-values, these p-values would in any case provide only a ranking given the massive degree of multiple testing, so nothing is lost in this regard.

To detect differences between classes, we simply look for areas with the largest magnitude spatial t-statistics, $t'_{\tau,i}$. This provides a ranked set of hypotheses. One could of course use different version of the t-statistic, such as those which attempt to correct for spuriously large values from tiny variance estimates, such as those discussed in [76], namely, [52, 82, 25], although we did not try these here.

6.4.4 Comparison to Ground Truth

To characterize the performance of our algorithms (and in particular, the be able to compare our alignment algorithms), we ran spiked-in reference peptides in buffer-only (no serum) through the LC-MS to obtain eight 'ground-truth' runs. Clearly, extracting a single ground truth from these runs is itself not a trivial problem. We might have used our CPM alignment/normalization to help extract the ground truth, but we sought a method which was completely independent from that which was used to do difference detection. Because the reference peptides are relatively few in number, peak detection was not overly difficult, though we note that it is still unlikely to be perfect. We extracted ground truth peaks in a fairly simple manner which involves four main steps, i) thresholding, ii) clustering, iii) *m/z* extraction, iv) voting across eight ground truth LC-MS runs.

Note that our ground truth comparisons use *m/z*, not time, since time is not consistent, producing a large dependency on alignment, which we wished to bypass for comparison with ground truth. However, the main signal of interest, and that with the highest precision and

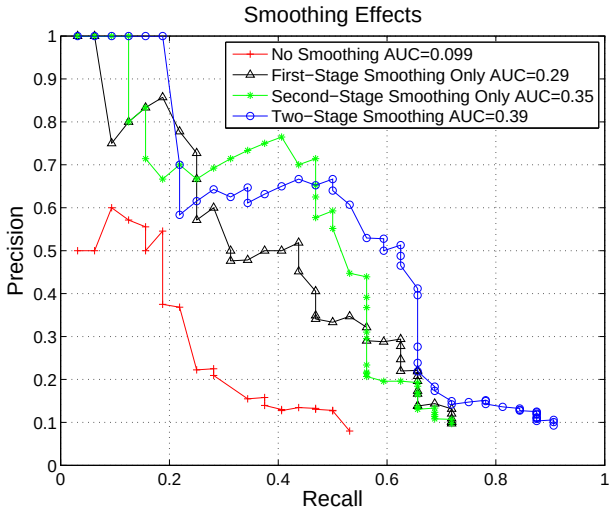


Figure 6.2: Precision-recall curves resulting from using 1) neither a 'residual misalignment' corrective smoothing, nor a t-test smoothing, shown in red, 2) using just the first stage of smoothing ('residual misalignment' corrective smoothing), shown in black, 3) using just the second stage of smoothing, shown in green, and 4) using both stages of smoothing, shown in blue. Use of two-stage smoothing is best, although use of just the second stage of smoothing provides similar results.

accuracy, lies in *m/z*, not time. We did however visually verify that detected differences did correspond to the spiked-in peptides in both time and *m/z*. Additionally, when our method is used in a setting that does not involve comparison to ground truth, it is simple to obtain LC-time estimates for all detected differences by mapping back from the alignment reference time frame to real observed time and averaging over replicates to obtain a time window.

Formally, for each ground truth run, w, we i) created a data matrix, $\mathcal{M}^w$, by binning the *m/z* as described earlier, ii) smoothed the matrix using 2D Hamming filter of length 10 in time and 3 *m/z* bins (spanning 1.5 *Th*), iii) normalized the total ion abundance in each matrix to be 1, iv) thresholded the abundance at each (time, *m/z*), so that if the abundance was less than T_0, then we set $\mathcal{M}^w_{i,\tau} = 0$, v) for all remaining non-zero entries in $\mathcal{M}^w$, call a peak any group of tuples (i, τ) that are connected (*i.e.*, are linked by a path of $\mathcal{M}^w_{i',\tau'} > 0$, where link is defined with respect to a neighbourhood consisting of the eight surrounding points in (time, *m/z*)). We calculated the *m/z* value of each peak to be the weighted-average *m/z* value of all tuples assigned to that peak, where the weights are proportional to each tuple's ion abundance. The threshold, T_0, was chosen on the basis of histograms of the data, and in such a manner as to attempt to cut off background noise while maintaining the greatest possible signal (we used $T_0 = 5 \times 10^{-5}$). We set this threshold low (retaining as much as possible) knowing that we would force replicate LC-MS ground truth runs to later agree, reducing the possibility of spuriously detected peaks.

Finally, we calculated the 'strength' of each peak as the sum of the ion abundance of all tuples assigned to it. If several peaks had the same *m/z*, their strengths were added to obtain the strength for that *m/z*. This strength attribute is later used to see if 'stronger' peaks are those identified more easily (Figure 6.8).

Given a list of *m/z*'s for all of the eight ground truth replicates, we then clustered the *m/z* values using complete-linkage, Euclidean distance hierarchical clustering, using a distance cut-off that produced clusters such that no two *m/z* values assigned to a cluster differ by more than 1 *Th*. That is, although the clusters are hierarchically constructed, the final use of the clusters uses just one level of clustering. Lastly, each unique cluster is voted on in a binary fashion from the 8 ground truth peak extractions (using a tolerance of 1 *Th*), and those clusters with at least 6 votes were considered to be in our final ground truth. In this manner, we extracted 32 ground truth *m/z* values. The strength of each of these was determined by the average strength of the peak corresponding to that *m/z* value, over the LC-MS replicates that voted for it.

We also had access to theoretical predictions for our tryptically digested peptides. While these theoretical predictions are not a bad guide, ultimately, it is unknown which charge-state ions will be observed, whether non-tryptic peptides will be found [66], and what the ionization efficiency of each tryptic peptides is (and hence whether it will be observed). We note however, that our experimentally extracted ground truth peaks contained 7 of the 8 theoretical predictions.

Lastly, we note that our naive method of extracting ground truth peaks cannot be applied to the actual difference detection problem itself, since in realistic settings, using for example serum (such as the experiments we use for difference detection here), there are so many peptide peaks that it is impossible to segment them from background in the naive way described in this section, and more sophisticated methods would need to be applied. Additionally, such a peak extraction would not address the alignment problem.

6.4.5 Precision-Recall Curves

In order to compute precision-recall curves for the difference detection task, we found regions of interest and compared their *m/z* values to our extracted ground truth list. *Precision* is the proportion of our detected differences that appear in the ground truth, while *recall* is the proportion of the ground truth values that we actually managed to recover.

To find a single point on the precision-recall curve, we i) choose a threshold, R_0, for the spatial-t statistic, and refer to all *m/z* values with a t-statistic larger in magnitude than R_0 as 'on', ii) find all unique 'on' *m/z* values, iii) obtain the number of true positives by counting how many of our 'on' *m/z* appear in our extracted ground truth list within a tolerance of 1 *Th*, (and

which also appear in the correct direction – negative in this experiment), not allowing more one than true positive count per ground truth peak, iv) for all 'on' *m/z* which did not match to a ground truth peak, cluster them (again, complete-linkage, Euclidean distance hierarchical in a way similar to before) such that every cluster contains only *m/z* values no more than 1 *Th* apart, v) use the number of unique clusters as the number of false positives. We repeated these steps for decreasing thresholds, R_0 to trace out the precision-recall curve, stopping after the precision fell below 10%. We also calculate the AUCs (Area Under the Curve) which we calculate by creating rectangles whose x-axis boundaries are neighbouring recall points, and whose y-axis boundaries are 0.1 at the bottom (because we stop computing the Precision-Recall curve when recall reaches 10%) and at the top, the average of the precision for each of the x-axis boundaries. Note that for algorithms which had extremely poor performance, that the precision would start below 10% – for these we continued to compute the precision/recall for 10 iterations, and continued longer if this brought the precision above 10%.

Note that precision-recall curves differ significantly from ROC curves. While a diagonal line on an ROC curve shows that a classifier is guessing no better than random, a diagonal line on a precision/recall curves shows that one can, for example, extract 50% of known ground truth values while incurring a false positive rate of 50%, which may be a non-trivial achievement.

6.5 Experiments and Results

In this section, we examine how the number of *m/z* bins used and the number of replicates affects our ability to detect differences. We also compare the sensitivity of DTW to the reference template and the sensitivity of EM-CPM to the latent trace initialization. We also compare the performance of DTW to the EM-CPM and the HB-CPM, as well as investigate how output from the HB-CPM can be used for difference detection.

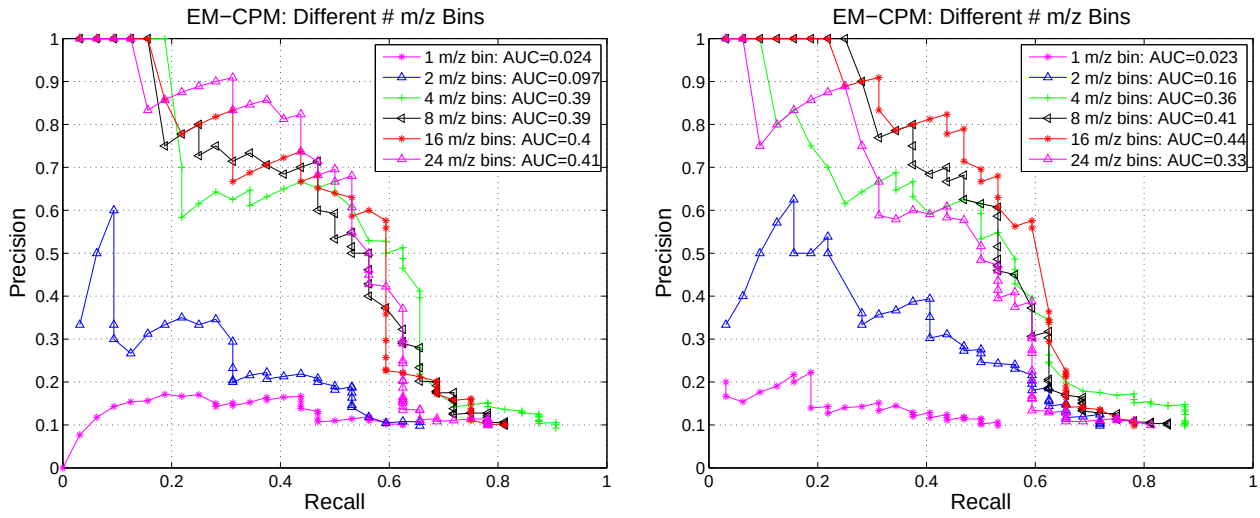
After investigating different alignment algorithms, we use the EM-CPM to perform a '1D' versus '2D' analysis to demonstrate the non-triviality of the spike-in problem we are using. Lastly, we show that difference detection is a much harder problem than classification on our data set.

We use George Tomasio's matlab implementation [81] of DTW for all of our experiments, which we have modified so that it can perform alignment using numerous *m/z* bins and then unroll the alignment to the full space using the same interpolation scheme for the output of the DTW. In all experiments in this section, we used the log transform of the data for alignment (and only for alignment).

6.5.1 Effect of Number of *m/z* Bins

In Chapter 4.2 we compared the performance of the EM-CPM when using different numbers of *m/z* bins on an LC-MS data set. However, we had no true way of assessing the quality of the alignments, other than through measures such as our 1D and 2D scores. With our current experimental set-up, we are in a position to assess the relevance of this factor in a more objective way. Figure 6.3 shows the precision-recall curves when using 1 (TIC), 2, 4, 8, 16, and 24 *m/z* bins. We also compared use of just a global scaling factor to using 7 HMM scale states with a global scale factor. The alignment of TICs (*i.e.*, 1 *m/z*bin) leads to quite horrendous results. Increasing to 2 *m/z* bins provides a dramatic improvement, with use of 4 *m/z* bins providing yet an even bigger win. Beyond 4 *m/z* bins, the returns are diminishing. In computing these precision-recall curves, we do not scale the data according to the global scaling factor (or HMM scale states), since doing so provided almost identical results. Likewise, throughout the rest of this chapter, we do not re-scale the data, since it is not required (note that we did re-scale the data globally in our pre-processing of the data). Note that alignment itself might be improved by allowing simultaneous scaling, even if the post-processing does not require a scaling of the data. That is, the quality of the alignment may be improved by simultaneous scaling, even if we do not use the recovered scaling when computing our t-tests. However, as can be seen in Figure 6.3, use of 7 HMM scale states does not provide a systematic improvement in the results over using just a global scaling factor. In fact, for 24 *m/z* bins, the result is worse when using the scale states, perhaps due to the fact that the scale states introduce more local minima in the problem, or, perhaps due to non-alignment related issues in our proxy measure of the quality of alignment. Based on this assessment – that scaling provides no real benefit on this particular data set, we only use a single, global scale factor throughout the remainder of this chapter.

As we mentioned in Chapter 4.2, the fact that aligned signals are ‘closer’ to each other (for example as measured by our 1D scores) is not necessarily a reflection of the quality of an alignment, and Figure 6.4 makes this point again. It shows the results of 1) no alignment, 2) alignment using the TIC (1 *m/z* bin), and 3) alignment using 24 *m/z* bins. The data is plotted by showing the TICs. It’s clear that the alignment resulting from 1 *m/z* bin looks much cleaner than that from the 24 bin alignment (and would have a better 1D score), however, from the precision-recall curves in Figure 6.3, we know that in fact the 24 bin alignment is a significantly better one.



a) b)

Figure 6.3: Precision-Recall curves resulting from use of the EM-CPM using 1, 2, 4, 8, 16 and 24 *m/z* bins to do the alignment. a) Shows the results from using EM-CPM with no scale states or scaling spline (just a single global scaling factor), while b) used the EM-CPM with 7 hidden scale states and a global scaling factor. In neither case was the scaling applied to the data for the purposes of difference detection.

6.5.2 Stability of DTW Versus EM-CPM

Figure 6.5 shows how DTW is very sensitive to choice of both the reference template, and also the amount of regularization. The spread of precision-recall curves (and AUCs) is obvious across both of these variables. In contrast, the EM-CPM, although susceptible to local minima (in that it frequently finds different solutions from different initializations), does so in a way that consistently provides good quality solutions, with far less spread in quality. In other words, the EM-CPM local minima are not a problem, practically speaking, on this realistic data set. We did not notice an obvious correlation between the EM-CPM log likelihood of each initialization and the AUC, when plotted. As a baseline, if one does no alignment on this data set (other than by virtue of the pre-processing and residual mis-alignment smoothing), the resulting AUC is 0.20. Each initialization of the EM-CPM was obtained by naively upsampling (simply repeating every measurement, next to itself), and then smoothing one of the observed time series, where smoothing was performed by a moving average filter with a window that was 20% the length of the vector being smoothed.

The take home message is that DTW and (presumably similar algorithms) are extremely sensitive to the the template chosen, and the amount of regularization. Furthermore, there is no principled way to choose these parameters, unless one has access to ground truth experiments

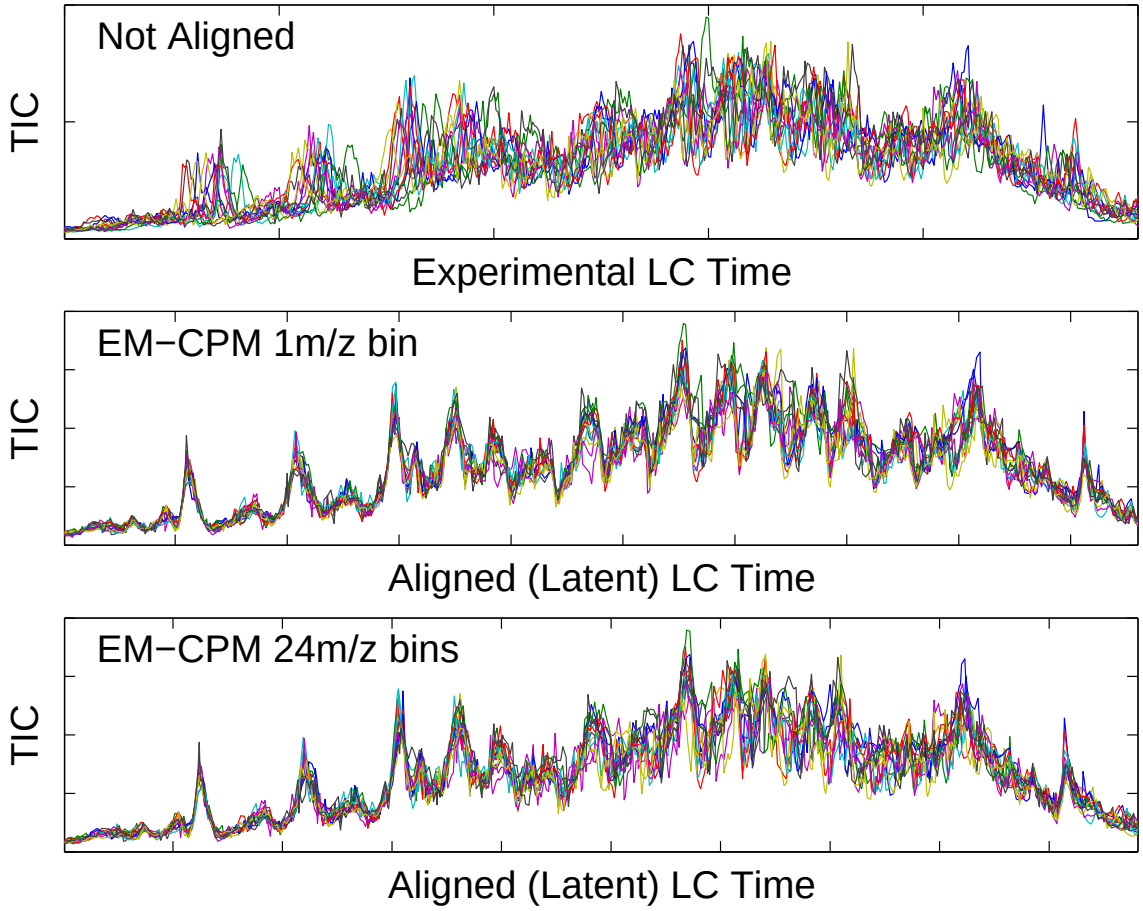


Figure 6.4: Unaligned and aligned TIC of the 14 LC-MS data sets. Although the TIC is shown, alignment was done using either 1 or 24 *m/z* bins, as noted on the figure. 'Not Aligned' actually refers to data that has been coarsely pre-processed with simple linear offsets (shifts) and global scaling.

which one believes to be very similar in nature to the real experiments of interest (a difficult condition to meet). One could try to use the TVD as a proxy, however, we found that the TVD was not a good predictor of AUC for the DTW alignments (although for the CPM alignments it was). In the present context, we can only see which is best because we know the ground truth, but in a real experimental setting, we would not know which template or regularization was really working best. Furthermore, even if one magically knew the best setting, it seems that the DTW results would be unlikely to beat those of the EM-CPM, whose worst AUC across all initializations is not very different from the best across all DTW templates and regularizations used in this experiment. Note that DTW often does worse than performing no alignment at all

(AUC=0.20), and that even when regularization is optimal[5], there is still a large dependency on the template.

6.5.3 Effect of Using Different Number of Replicates

One would expect that using more replicates per class would make it easier to detect differences, since more information would be available, both for alignment and for difference detection. It's interesting to ask what effect the number of replicates has on each of these tasks. Thus, we performed two experiments (using 4 *m/z* bins) with the EM-CPM. In the first, we aligned all 14 time series (7 replicates per class), and then performed difference detection on 2, 3, 4, 5, 6, and 7 replicates per class. Then, we performed *both* alignment and detection using 2, 3, 4, 5, 6 and 7 replicates per class. This allows us to tease apart the contributions of having more data for alignment and for post-processing (difference detection). The results are shown Figure 6.6, and we see that alignment itself does benefit from access to all of the data – more so than the post-processing.

6.5.4 Obtaining Alignments from the HB-CPM

Unlike DTW or the EM-CPM, the HB-CPM does not straight-forwardly produce a single alignment upon which we can perform our difference detection. If alignment were performed in the space of interest, that is the full 2D LC-MS space, then we could naturally use the latent traces to directly look for regions of difference, as the model was intended. However, in our experiments in this chapter, we perform alignment in a reduced dimensionality space because of the huge number of *m/z* values which would otherwise make running of the algorithm infeasible under the constraints of current computational speeds.

Use of MCMC with the HB-CPM, a fully Bayesian model, gives us an approximation to the posterior over different alignments (and other parameters), and there are a number of ways in which we could use this posterior. The standard use of the posterior in a fully Bayesian context, is to compute some probability of interest (*e.g.,* the probability that a new test point belongs to class A in a classification problem), integrating out the posterior. In the case where one has MCMC samples to represent the posterior, one computes the probability of interest for each sample, and then averages the results to approximate the desired integral. However, our output (an alignment) cannot naturally be averaged over, because each alignment lies in its own reference frame. We can instead try mapping each sample's reference frame to some common reference time frame. We tried such an approach, as described in Algorithm 2. Intuitively, this

[5]We could not try every possible regularization since there are many different kinds, but we used warping slope constraints and continued to increase this regularization until AUC performance decreased.

merging algorithm works as follows. We choose one of the observed time series to define a common reference time frame. For one MCMC sample, we evenly spread out the observed time points of this reference sample across our common reference frame (and this operation is independent of the MCMC sample, thus providing us with the common reference frame across all MCMC samples). Note that we must leave 'vacant'[6] common reference times in between each of the reference samples observations so that mappings of other observed time series to this reference frame remain monotonic (time does not move backward or stand still). To maintain monotonicity, we need to place precisely $J-1$ vacant reference time points between every observed time point in the reference frame, where J is the maximum allowable number of HMM hidden time states that can be advanced in a single transition (specified in the CPM). Then for each time series, for this given MCMC sample, we map one of the non-reference time series to this common reference frame by comparing its HMM alignment (defined by a series of enumerated hidden states) to the HMM alignment of the reference time series, and offsetting it appropriately in the common reference frame. Note that there could be sensitivity in this method to the choice of the reference sample, but this sensitivity is expected to be considerably less than that of DTW to its template, since here we have already, in principle, aligned the data, and are only doing post-processing. Indeed, when we systematically tried all 14 observed time series in turn to create a common reference time frame, (using 7 replicates per class), the AUC did not differ tremendously, as shown in Figure 6.6d, which merged 100 samples (every 25^{th} MCMC sample from the last half of 5,000-long runs) from the posterior. Since this merging of MCMC samples creates a more dense reference time frame than the EM-CPM, by a factor of about 1.5, we used 1.5 times the amount of smoothing in these experiments.[7]

Arbitrarily using the first sample to provide a common reference frame to merge 100 HB-CPM posterior sample alignments, and systematically varying the number of replicates in each class, we obtained results as shown in Figure 6.6c. Somewhat counter-intuitively, the HB-CPM seems to do (slightly better) than the EM-CPM with more data, but slightly worse with less data. This could be just random variation in the AUCs, or an artifact of our merging algorithm for the posterior alignments. It could also be the case that the merging works better for posteriors which are more peaked, which would occur with increasing amounts of data.

To get a sense of the variation in the quality of alignments from typical samples in the posterior, we systematically took every 25^{th} MCMC sample from the last half of 5,000-long runs (*i.e.*, 100 samples evenly spaced in the last 2,500 iterations), and computed the precision recalls using each of these 100 samples alone, as shown in Figure 6.7. The joint log likelihoods

[6] By vacant we mean that none of the reference sample's observed time points map to these points.

[7] We later performed our TVD cross-validation and found that a factor of 2 in smoothing was optimal as measured by the TVD. However, a factor of 2 produced AUC results almost identical to those of using a factor of 1.5.

Algorithm 2 Merging MCMC samples from the HB-CPM into a common reference time frame

1. Choose one of the observed time series, $\hat{k}$, which will define the common reference time frame. This is the 'reference sample'.

2. Enumerate the time points in the common reference time frame as $t_1, \ldots, t_T$ (where T will fall out of the algorithm below), and let $s_j^{kl} \in \{t_1, \ldots, t_N\}$ be the time in the common time frame for the j^{th} point of the k^{th} observed time series, according to MCMC sample l. Next we compute the s_j^{kl}.

3. Let J be the maximum allowable number of HMM hidden time states that can be advanced in a single transition (specified in the CPM).

4. For each MCMC sample, l, being merged,

   - For each observed time series, k, starting with $k = \hat{k}$,
     - Get the HMM state sequence from the current MCMC sample and denote it $(\tau_1, \ldots, \tau_N)$.
     - If $k = \hat{k}$, then map the first HMM time state to the first time point, t_1, and the second HMM time state to $t_1 + J$, *etc.* (*i.e.*, evenly spread out the reference sample in the common reference time space). That is, let $(s_1^{kl}, s_2^{kl}, \ldots, s_N^{kl}) = (1, 1 + J, \ldots 1 + (N-1)J)$.
       Also, let $(\tau'_1, \ldots, \tau'_N) = (\tau_1, \ldots, \tau_N)$ for future use.
     - If $k \neq \hat{k}$, then compute $(\delta_1, \ldots, \delta_N) \equiv (\tau_1, \ldots, \tau_N) + (\tau'_1, \ldots, \tau'_N)$, that is, the difference from the hidden time states of the reference sample. Then $(s_1^{kl}, \ldots, s_N^{kl}) = (\tau_1, \ldots, \tau_N) - (s_1^{\hat{k}l}, \ldots, s_N^{\hat{k}l})$. (This may cause some s_1^{kl} to be negative or zero, which we will fix afterward.)

5. Let $t' = \min(\{s_1^{kl}\})$ (across all k, l). For every s_j^{kl}, correct it so that the minimum time index is 1. That is, $\forall k, \forall l, \forall j, s_j^{kl} = s_j^{kl} + t' - 1$. (Now we see that $T = \max(\{s_N^{kl}\})$.)

6. For every observed time series and every MCMC sample, linearly interpolate to obtain values for points t_j which were not mapped to.

7. For every observed time series, average its aligned and interpolated values from every MCMC sample to obtain its new representation.

across MCMC samples did not appear to be a correlated with the AUC. Out of the 100 MCMC samples, most of the posterior alignments are quite good, and if one were to use just one of these at random, chances are it would produce good results, although there would be no guarantee that we hadn't chosen a poor quality one. Note that we have used the same 100 samples here as when merging samples from the posterior in Figure 6.6, and that the average of the AUCs in Figure 6.7 is comparable those in Figure 6.6c.

We initialized the (parent) latent trace in each of these experiments to be that from the corresponding EM-CPM experiment, and did not update the latent trace for the first 500 iterations so as to obtain a good initialization. We then updated as usual for the remaining 4500 iterations. Results from using a more naive start (smoothed version of one observed time series) were comparable, except for when using 4 replicates per class, in which case the naive initialization did much worse, presumably caught in a poor local minimum. The EM initialization seemed to confer a small gain in terms of number of iterations to convergence, of about 1000 iterations.

6.5.5 Assessing The Difficulty of the Difference Detection Problem

If the amount of reference peptides spiked in were very large, the difference detection problem might be trivially easy. How can one get at how difficult our problem is? If the amount of spike-in were very large relative to the base mixture, then the spike-in would swamp the signal of the base mixture, and a simple *1D analysis* looking at the ion count at each *m/z*, irrespective of time (*i.e.,* summing out the ion count out over time) would be able to tease out all differences of interest. We apply such a technique here, and show that the 1D analysis is inferior to the full *2D analysis*, where the 2D analysis that was done in Section 6.4.

Formally, the 1D analysis involves converting each data matrix, $\boldsymbol{\mathcal{M}}^k$, to a vector representing the total ion count at each *m/z* bin. Thus we define the Total Time Ion Count (TTIC), for *m/z* bin i for the k'th LC-MS run, T_i^k, as $T_i^k \equiv \sum_\tau \mathcal{M}_{i,\tau}^k$. Then the 1D analysis is exactly the same as the 2D analysis, only it operates on $\{T_i^k\}$ rather than on the set $\{\mathcal{M}_{i,\tau}^k\}$. A direct comparison of the 1D versus the 2D approach is shown in Figure 6.8. The 2D approach allows us to more precisely find subtler regions of interest. To assess the stability of this relationship, we redid this analysis seven times, each using a different subset of size six of the seven replicates per class, and found the pattern to be somewhat unstable in the region of recall less than 0.3, indicating there are no statistically significant differences in this region. However, for recall larger than 0.3, the pattern was stable, suggesting a statistically significant difference for regions of larger recall. Thus we have provided evidence that the spike-in experiment we have been working with is not trivially easy, and benefits from a full 2D analysis. Figure 6.8

also shows how 'stronger' peaks tend, loosely, to be detected before 'weaker' peaks. The correlation is better for the 1D analysis, which is not surprising given that our measure of peak strength was defined in a 1D manner.

We want to emphasize that the point here is not to contrast two difference detection methods (as they are essentially the same), but to show that the signal of interest is indeed swamped out, to a certain extent, by baseline peaks, indicating that the amount of spike-in is not so large as to rise above this, and thus trivially easy to find. Equivalently, we have also demonstrated that indeed there are regimes in which the LC component of LC-MS is of benefit over a simple MS experiment.

6.5.6 Predictive Modeling

In this section we briefly look at our data set from a classification perspective in order to emphasize that difference detection is the more challenging task.

Earlier in the chapter we were unable to achieve perfect difference detection results, (where perfect would mean always having precision=1 and recall=1). However, here we report that with just a single, simple attempt at building a supervised predictive model, we were able to achieve perfect classification results (as assessed by an ROC curve). We used regularized logistic regression [32] as a prediction model, with features consisting of every (time, *m/z*) in the aligned data matrixes, $\{\mathcal{M}^k\}$, described earlier.[8] We used L2 regularization, which is similar to putting a Gaussian prior with spherical covariance over the LR weights.

Over a range of 7 orders of magnitude for the regularization term, we always achieved perfect sensitivity/specificity as measured by leave-one-out cross validation (*i.e.*, a perfect ROC curve). This demonstrates that the problem of classification is far easier than that of difference detection, and that good classification performance provides little information about the ability to do comprehensive difference detection. Of course if one were unable to build a good classifier, this would suggest that the task of difference detection would likely also be difficult. However, if one were able to successfully perform difference detection, then one might reasonably expect to be able to build a good classifier. On a related note, one could consider using classification as a means to verify posited differences. That is, after performing difference detection on a subset of the data, one could use the remainder of the data to cross-validate a linear model using only the regions of detected differences as features. If this model had good predictive ability, that would provide some indication that some of the chosen features were real. However, use of permutation style tests would make better use of the data and likely

[8] For computational efficiency, we actually do logistic regression in an equivalent 14-dimensional feature space obtained from PCA (Principal Components Analysis) which gives identical results to doing logistic regression in the original feature space [58].

provide a better sanity check.

6.6 Compute Times

To convey a sense of how long various algorithms took to run, we now briefly report CPU times. This is meant only to convey broad differences, and should not be be taken as a detailed study of fine-tuned implementations in a fast language. All computations were done in Matlab, on machines with dual 3 GHz Pentium 4 processors which were also shared by other users at the same time. We report on alignment of the data set used in this chapter (all 14 samples, with 4 *m/z* bins).

DTW alignment generally took about 1 minute. The EM-CPM (with no cross-validation), without learning state transition probabilities, and without using HMM hidden states or a scaling spline, took about 1.5 hours, and if we included the HMM hidden scale states (and no scaling spline) took about 6 hours. However, we probably ran the EM-CPM much longer than needed to ensure convergence (100-200 iterations). The single-class HB-CPM, which used no scale states, or scaling spline, ran through 5000 iterations in about 2 days. Subsequent merging together of 100 posterior samples required about another hour since the numerous linear interpolations are computationally expensive.

6.7 Discussion and Conclusion

We have presented and evaluated a technique for discovering differences in protein signal in serum, between two classes of samples of LC-MS data, without use of tandem mass spectrometry, gels, or labeling. Along the way, we demonstrated how to show that a particular spike-in difference detection problem is not trivial, and in particular that our difference detection problem was not trivial.

This set-up allowed us to investigate the relative merits of different alignment algorithms, such as DTW, our EM-CPM and single-class HB-CPM. We found that EM-CPM is stable to initialization in terms of the quality of alignments, and that DTW is not. Furthermore, DTW is heavily dependent upon the amount of regularization used, whereas the EM-CPM was not, which may have to do with the fact that the EM-CPM can learn many of the parameters which are hard-coded in DTW, such as those related to the error function. We also investigated how to use posterior samples from the HB-CPM in the context of an experiment where we can not directly appeal to class-specific latent traces to look for differences. We found that it is possible to merge the alignments of different samples into a single alignment to obtain a good quality

solution, although the approach we chose may not be optimal, and alternatives might be worth exploring.

An interesting point to note in using the CPM with LC-MS data is that the CPM allows us to tackle not only the time alignment problem, but also the mass spectrometer stochastic sampling problem that arises when the mass spectrometer is run in alternating scan mode (*i.e.*,doing tandem mass spectrometry), as mentioned in Chapter 3. That is, the CPM, through use of its latent trace, accounts both for the differences in timing within and between LC-MS runs, and also for the stochastic MS sampling, because when the latent trace has more time points than the observed traces, points in time across different observed traces need not be put directly into correspondance with each other, and thus signal gaps are naturally handled.

Lastly, we showed that the problem of supervised prediction, sometimes mistaken as a solution to biomarker discovery, is actually a much simpler problem.

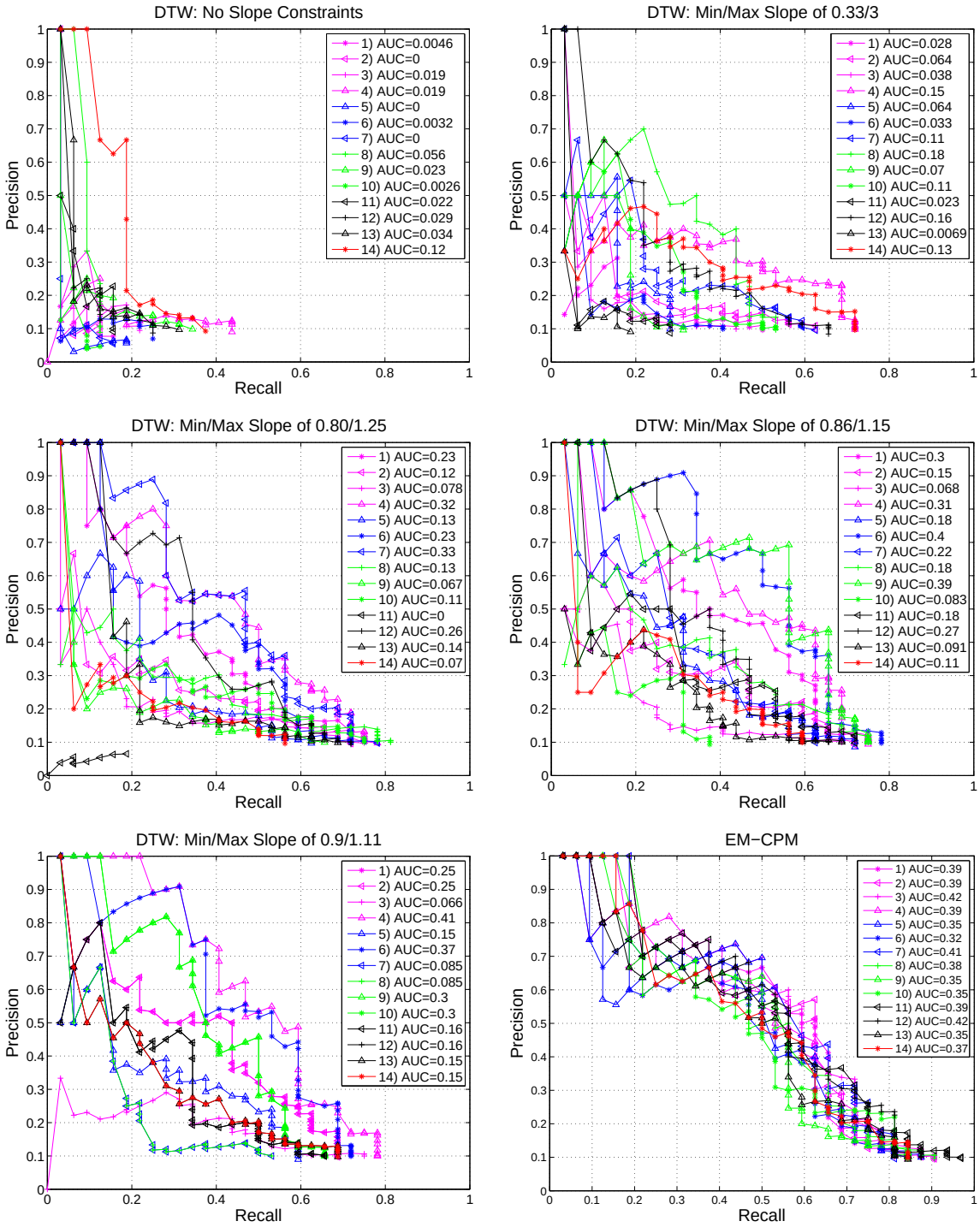


Figure 6.5: Precision-Recall curves and AUC demonstrating sensitivity of DTW to its template, and of EM-CPM to its latent trace initialization. DTW was used with different slope constraints on the warping path (as annotated in the titles), and the EM-CPM was used without scale states, using a smoothed version of each of the 14 time series as initialization (20% width moving average). More rigid contraints on the DTW warping path slopes than those shown here did not improve the quality of results (in terms of AUC, or sensitivity to the template). Four *m/z* bins were used in all cases.

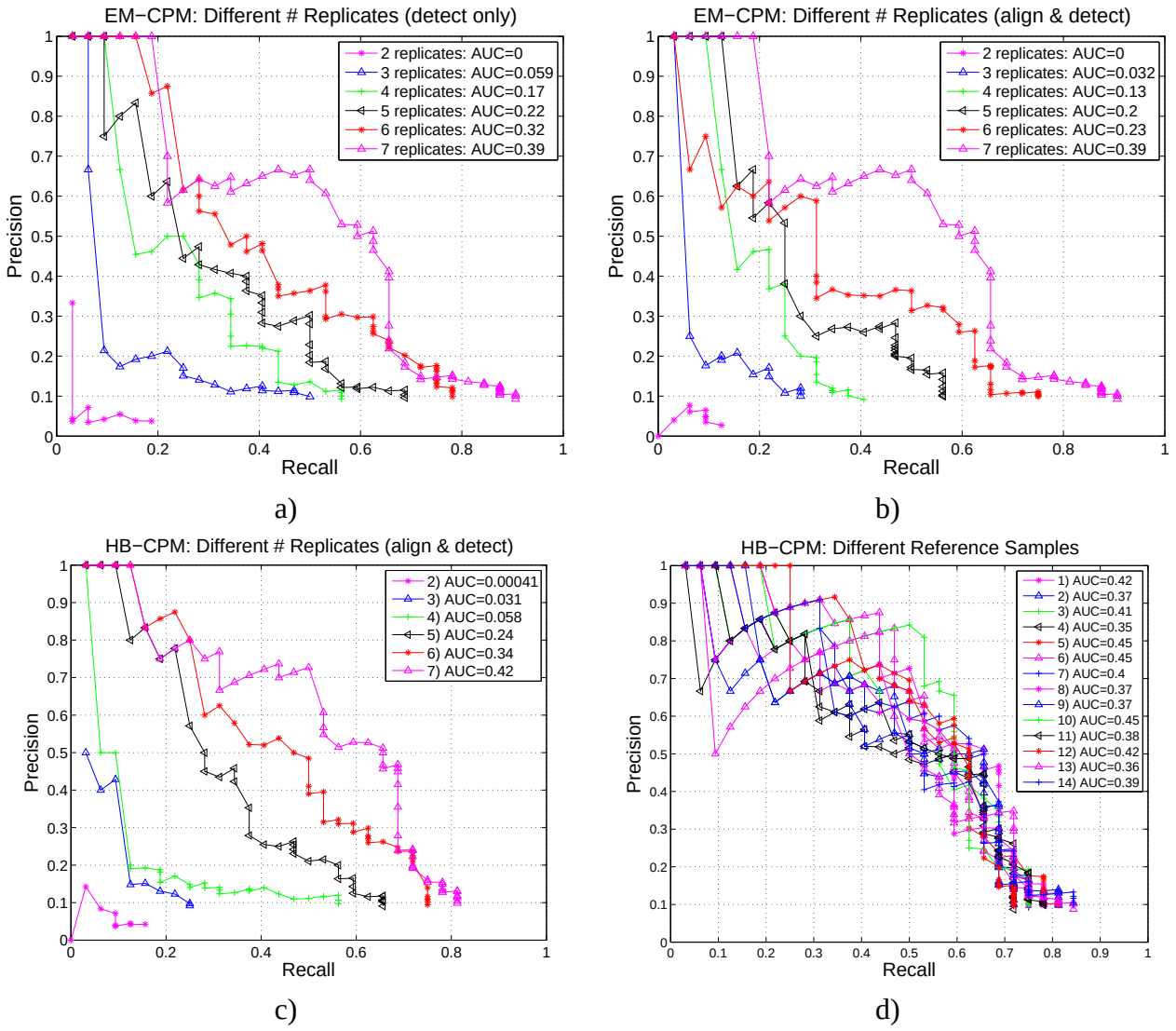


Figure 6.6: Precision-Recall curves for the EM-CPM and HB-CPM, using various number of replicates (and always 4 *m/z* bins). In a) all 14 of the observed time series were used for alignment, followed by difference detection with the specified number of replicates (per class), whereas in b) both alignment and difference detection were performed with the specified number of replicates. c) shows the results from the single-class HB-CPM when merging 100 posterior alignments, while d) shows the results of using each of the 14 observed time series to produce a common reference frame when merging alignments from posterior samples of the HB-CPM, using all 7 replicates.

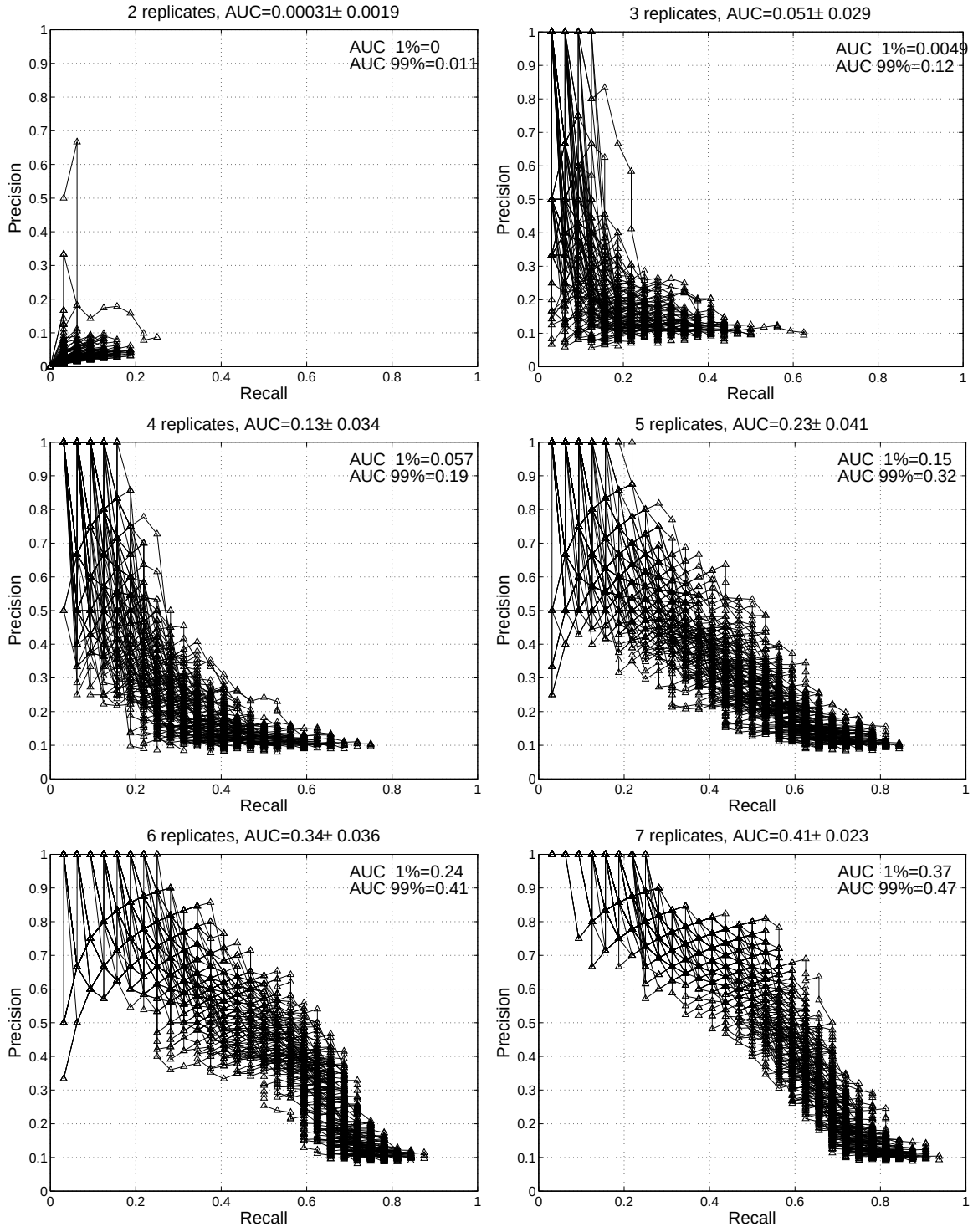


Figure 6.7: Precision-Recall curves resulting from use of the single-class HB-CPM using every 25^{th} MCMC sample from the last half of a 5,000-long run. The 1 and 99 percentile AUC scores are shown in the main plot, and the mean and standard deviation shown in the title of each plot. Results are shown for use of each of 2 replicates per class all the way through 7 per class. Recall that the HB-CPM AUCs from merging these same 100 samples were respectively 0.0004, 0.031, 0.058, 0.24, 0.34, 0.42 for 2, 3, 4, 5, 6, and 7 replicates per class.

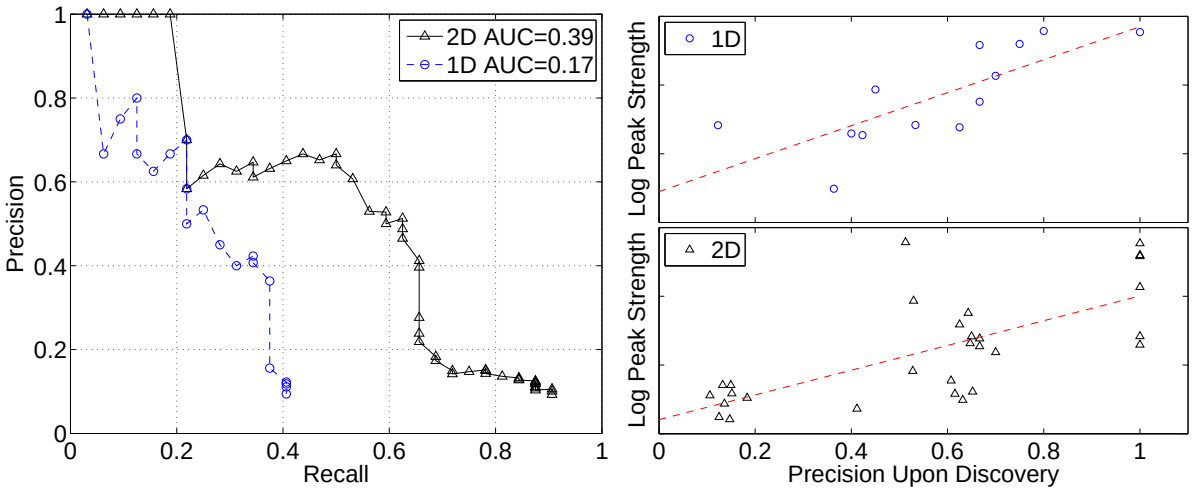


Figure 6.8: Left: Precision-Recall curve for 1D analysis (dashed-circle) versus a full 2D analysis (solid-triangle). Right: The precision level when each peak is detected along with the strength of the peak. Overlaid is linear fit of the plotted data points to more easily visualize the trends. As expected, stronger peaks are discovered at higher precision levels, though the trend is noisy. Both plots used seven replicates from each class. The 1D plot contains 20 ground truth peaks, while the 2D plot contains 29.

Chapter 7

Summary and Future Directions

Many practical problems over a wide range of domains require synthesizing information from time series data. Two distinct, yet related, problems in time series data are alignment and difference detection. Both problems are pervasive, spanning, for example, speech and music processing, equipment and industrial plant diagnosis/monitoring, and analysis of biological time series from microarray and liquid/gas chromatography-based laboratory (LC/GC) data (such as mass spectrometry (LC-MS) and ultraviolet diode arrays). Thus development, refinement and examination of solutions to these problems has the potential to make a large impact on the success of real, practical areas of research. Of particular interest to us is the domain of chromatography-based laboratory experiments which themselves play a major role in many active research areas, from basic biology through to translational and clinical research. We focused on the liquid chromatography domain for examination and demonstration of our methods, although our methods are in no way confined to this domain.

This thesis has introduced a principled, probabilistic approach to the problem of alignment. This approach takes the form of a class of models called Continuous Profile Models (CPM) for simultaneously analyzing sets of sibling time series (those which contain shared sub-structure, but may also differ). In this type of of generative model, each time series belonging to one class is generated as a noisy transformation of a single *latent trace* in the model. The latent trace lies at the core of this class of models, and provides the basis for alignment and difference detection. If multiple classes of data exist, then there is one latent trace per class, and data is generated from the appropriate class-specific latent trace. A latent trace can be viewed as an underlying, noiseless representation of the set of replicated, observable time series. Output time series are generated from this model by moving through a sequence of hidden states in a Markovian manner and emitting an observable value at each step, as in an HMM (Hidden Markov Model). Each hidden state corresponds to a particular location in the latent trace, and the emitted value from the state depends on the value of the latent trace at that position.

Additionally, to account for changes in the amplitude of the signals across and within replicates, the latent time states are augmented by a set of scale states, which control how the emission signal will be scaled relative to the value of the latent trace. Thus the full hidden state space is the product state space of 'hidden times' and 'hidden scales'. Lastly, each time series can be scaled by a constant global scaling factor, or a scaling spline parameter.

Our approach provides several benefits over traditional approaches:

1. We can simultaneously leverage data from all time series at the same time to find an alignment which is globally good for the entire data set. This is in contrast to most other approaches which often use a greedy heuristic to successively merge and align time series in a pair-wise fashion.

2. Using probabilistic models allows us the option of using a parameter point-estimate (*e.g.*, MAP estimate), or employing a fully Bayesian approach which gives us access to to the full posterior should we wish to get a sense of the 'error bars' on our solution. The fully Bayesian approach also allows us to avoid cross-validation when fitting prior/penalty parameters, which one would have to do in a model which was not fully Bayesian (*i.e.*, EM-based, or any non-probabilistically based model). Cross-validation can be especially problematic in situations where data is scarce, since it requires dividing the data into partitions.

3. We can tailor our model to the data on hand, in an automatic and principled way, avoiding *a priori* and *ad hoc* ways of specifying a reference time frame and a distance/error function for measuring how similar a set of observed time series are to one another. Instead, we use parameters/latent variables which represent these quantities and learn them from the data in a statistically sound way.

Additionally, our class of Continuous Profile Models allows us to introduce the first (to our knowledge) approach to the multi-class alignment problem for sets of sibling time series data. Rather than assuming that all the data come from the same distribution and attempting to align them all together, explaining any differences as random noise, we develop a multi-class Continuous Profile Model which reflects our prior belief that there is both substantial shared structure between the class distributions and, simultaneously, systematic differences between them, allowing for more refined and accurate modeling of the data. This framework has the potential not only to perform multi-class alignment, but also, difference detection, by virtue of the latent variables which are learned from the data. Thus, Continuous Profile Models also provide a unified approach to the two coupled problems of multi-class alignment and difference detection.

7.0.1 Limitations

We now discuss some of the limitations of the CPM framework. The most obvious limitation is the computational time required to train the model (or do inference), especially as compared to algorithms such as DTW. DTW takes an order of magnitude less time than the EM-CPM, and the HB-CPM takes an order of magnitude longer than the EM-CPM. Granted, each order of magnitude buys increased modeling power, so one can easily start with the simplest approach to see if it is sufficient for the problem at hand.

Another limitation is the susceptibility of the CPM to fall into a local minimum. With enough random restarts, one would hope to overcome any major local minima hurdles, but there could be no guarantees. Note that models which do not have a local minimum problem likely are not rich enough to model the intricacies of sets of complex time series (such as multi-class time series, or time series requiring alignment with scaling). Again, we see a trade-off between simple models which can avoid some computational issues, but at the cost of a coarser level of modeling.

As with any unsupervised model, it is extremely difficult to gauge whether the output from the CPM is indeed correct. If one had access to the ground truth on similar data sets, one could hope to characterize the behaviour of the algorithm on those first, with the assumption that the lessons learned would generalize to the real data set. However, it is difficult to obtain experiments with ground truth which exactly mimic the real problem of interest. As a proxy, in the multi-class setting, one could use permutation style tests to see if the model is prone to finding spurious differences. For example, given a set of 10 time series from each of two classes, one could scramble the labels, and then use the HB-CPM to do inference. Since the labels are scrambled, if the HB-CPM were behaving as we intended, it would model all variation as variation in the parent trace, rather than as class specific differences. If it did not, we should be suspicious of the results arising from the true class partition on that same data set.

The fact that the latent traces and scale states are discretized in the CPM means that the underlying canonical representation will in most cases not be optimal, and correspondingly, nor will the estimated emission variance. In particular, the discretization of the scale states will directly effect the estimated variance. The fewer the number of scales states, and the further apart they are, the larger the learned HMM emission variance will tend to be.

7.1 Future Directions

Continuous Profile Models provide a nice framework for analyzing sibling time series, and could be extended in a number of interesting and potentially useful ways which we now briefly discuss.

One could apply the HB-CPM in a completely unsupervised setting where one learns not only the canonical class representations, but also obtains the posterior over the class labels by introducing a latent class indicator variable. Such a model would perform joint alignment and clustering of the data, similar to the work of Gaffney *et al.* [28], but the CPM extension would provide a more flexible set of alignments (not just linear warps with offsets). This would allow solutions to problems such as semi-supervised biomarker discovery, in which one is interested in finding markers which differentiate between two classes, and also, finding new patterns within a known class.

Another extension would be to modify the CPM for analysis of cyclical data, such as electrocardiogram (ECG) data. In such a model, an observed trace being generated by the model would be allowed to cycle back to the start of the latent trace, and the smoothness constraints on the trace would apply to the beginning and end of the traces, coercing these to be similar. Such a model could allow one to do anomaly detection in cyclic data, as well as segmentation.

If one had particular kinds of information about how time series data was collected, then one could modify the CPM to account for this *a priori* knowledge. For example, suppose that the way each experimental time point was generated was by averaging a sequence of measured values collected from just after the previous time point (so as not to lose information that was available, but could not be recorded after every 'event'). One could incorporate this generative process in the model, altering the way the CPM generates an observed time point from the latent trace. Rather than generating a time point from a gaussian centered on the current point in the latent trace, one could generate a time point from a gaussian whose mean was calculated to be the mean of the previous t time points in the latent trace, mimicking the way the data was generated. The size of the averaging window, t, would be the number of discrete time points in the latent trace from the latent time of the last emitted symbol. In the case where different time series were generated at systematically different overall sampling rates (frequencies), one could thereby naturally incorporate information from all frequencies in a principled manner. Additionally, if the true underlying windows were overlapping (but not completely overlapping), then it would be possible to obtain a higher resolution latent trace than any of the observed time series (similarly to how one could obtain a higher resolution latent trace in the simpler case presented in this thesis).

One could consider handling outlier time points in any time series by making the emission

distribution a mixture of 'inlier' and 'outlier', where the inlier would be the emission distribution currently in the model, and the outlier one would be larger in order to robustly handle the outliers. Additionally, one might consider handling entire outlier time series by having a single latent variable per observed time series which stipulates whether that trace is an outlier or not. If an observed time series were an outlier, it could be generated using exclusively a high variance emission distribution.

If one thought that the order in which time series were generated had a systematic effect on their relative alignments (*e.g.*,if samples off a liquid chromatgraphy column tended to come off faster and faster with each new sample, or more offset from the beginning with each new sample), then one might consider incorporating such information into the model, such as in the form of a prior on the starting latent time state, or the size of the transition jumps, and then tying these together according to their order through the chromatography column.

One could create richer hierarchies than those used in the HB-CPM. For example, one might choose to model both patient-specific effects, as well as class-specific effects, if one had technical replicates from patients belonging to different classes in an LC-MS biomarker discovery setting.

It might be useful to try to attack the problem of simultaneous peak detection and alignment in a CPM-like model. For example, NMR (Nuclear Magnetic Resonance) spectral peaks are often modeled as the super-position of many Cauchy-shaped peaks (called Lorentzians by NMR scientists). One could use a prior for the CPM latent trace which was a mixture of Lorentzians to simultaneously detect features and align data sets.

In our extensions from scalar to vector time series, we treated the components of the vector largely independently. It would be interesting to examine richer alternatives in which, for example, neighbouring components are coupled. Or better yet, to examine ways in which different components could actually be aligned differently.

A current bottleneck of the CPM algorithms is how the time complexity scales with the length of the observed time series being modeled. In the EM-CPM it scales roughly linearly, which isn't bad. But in the HB-CPM, sampling of the energy impulses scales cubically with the length of the sequences. One could consider two approaches to help alleviate this problem, which are not centered on extensions to the model, but on practical approaches to dealing with this issue. The first approach would be to devise a coarse-to-fine alignment strategy in which one first aligns subsampled versions of the time series, and then initializes a more refined alignment with the solution. A second approach would be to trade off some of the globality of the solution in order to parallelize the algorithm. For example, one could split up each time series into say 2 or 3 overlapping chunks, and align each chunk separately, and then choose a method to calibrate the overlapping ends. Such an approach could have huge time savings

for long time series. One might also consider mixing these two ideas together, that is, using a coarse-to-fine, parallelized version of the algorithms. For the EM-CPM, one could parallelize the E-Step without any loss of globality, by running each time series' portion of the E-Step on its own processor.

Appendix A

Probability Distributions Notation

Scalar Normal: $\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sqrt{(2\pi\sigma^2)}}\exp\left(-\frac{1}{2\sigma^2}(x-\mu)^2\right)$

Multivariate Normal: $\mathcal{N}(\boldsymbol{x}|\boldsymbol{\mu},\boldsymbol{S}) = (2\pi)^{-\frac{D}{2}}|\boldsymbol{S}|^{-\frac{1}{2}}\exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T\boldsymbol{S}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right)$

Gamma: $\text{Gamma}(x|\alpha,\beta) = \frac{\beta^\alpha x^{\alpha-1}e^{-\beta x}}{\Gamma(\alpha)}$

Inverse Gamma: $\text{Inv-Gamma}(x|\alpha,\beta) = \frac{\beta^\alpha x^{-\alpha-1}e^{-\frac{\beta}{x}}}{\Gamma(\alpha)}$

Student-t: $T_\nu(x|\mu,\sigma) = \frac{\Gamma((\nu+1)/2)}{\Gamma(\nu/2)\sqrt{\nu\pi\sigma}}\left(1+\frac{1}{\nu}\left(\frac{x-\mu}{\sigma}\right)^2\right)^{-(\nu+1)/2}$

Dirichlet: $\mathcal{D}(x_1,\cdots x_n|\alpha_1,\cdots\alpha_n) = \frac{\Gamma(\alpha_1+\cdots+\alpha_n)}{\Gamma(\alpha_1)\cdots\Gamma(\alpha_n))}x_1^{\alpha_1-1}\cdots x_n^{\alpha_n-1}$

Beta: $\text{Beta}(x|\alpha,\beta) = \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}$

Multinomial (n=1): $\text{Multinomial}(x=a_k|\theta_1,\cdots\theta_K) = \theta_k,$

$$\left(\text{where } \sum_{i=1}^{K}\theta_i = 1 \;\; a_k \text{ is the } k^{\text{th}} \text{ value}\right)$$

Appendix B

Hidden Markov Models

Since a main component of a CPM is a Hidden Markov Model (HMM), we now briefly go over the relevant basics of HMMs but refer the reader to [23] for a good beginner's introduction to HMMs, to [35] to see how HMMs fit in to the broader context of graphical models and their associated algorithms, and to [65] for a foundational HMM paper.

An HMM is typically used to model time series data, discrete or continuous-valued, and can be viewed in two main ways: 1) as a stochastic finite-state machine (where transition between states is stochastic, as is emission of symbols from a state), or 2) as a graphical model in which hidden/unobserved states/variables are connected in a directed chain to each other, and where each hidden node has a child which is an observed variables (shown in Figure B.1). Viewing the HMM as a graphical model nicely places the HMM in the broader context of related models such as State Space Models [35] (which use Kalman filtering), mixtures of gaussians, *etc.* and naturally allows one to extend/modify the model by providing access to the machinery of the machine learning/statistical modeling community, such as Belief Propagation (BP) to perform inference [35, 34], Expectation-Maximization (EM) to perform learning [18], Markov Chain Monte Carlo (MCMC) [59, 54] to perform learning/inference – algorithms relied upon in this thesis for the various models introduced. Whichever view one prefers, several core concepts remain:

1. Observed samples are generated by moving stochastically through a set of hidden/unobserved states, stochastically emitting a symbol at each state.

2. Which observation is generated depends only on the currently occupied hidden state, and which state is currently occupied depends only on the previously occupied state (the latter condition provides the Markov property).

3. Computation in an HMM can be performed efficiently by way of Dynamic Programming (DP) which works by re-using partial calculations toward larger calculations.

A typical setting for use of an HMM is one in which several related time series have been observed, and one is interested in uncovering underlying, common patterns among the time series. For example, in computational biology, one might have a set of DNA sequences, and might be interested modeling the characteristics of which regions of DNA are 'CpG islands' (see [23] for details) – regions enriched with the nucleotides C and G. In such a problem, we would be provided with the DNA sequences, and thus would know the alphabet of letters from which they're built (nucleotides A,C,G,T). Our prior knowledge would also tell us that there are two types of states 'CpG island' and 'not CpG island', but *not* which regions of our observed DNA sequences were or were not 'CpG islands'. We would not know the transition probabilities between such states, nor the probability of emitting a particular symbol (one of the four nucleotides) in either state. Tasks we would want to solve would include 1) learning the transition probabilities between the states, 2) learning the emission probability of each symbol for each state, and 3) after learning 1) and 2), finding out, for the observed sequences, which regions we believe to belong to the 'CpG island' (*i.e.,* finding out for each component of each sequence, whether it should be labeled as 'CpG island' or 'not CpG island'). These are three of the standard problems one needs to solve when using an HMM, and problems we face in using the CPM. Additionally, one may be interested in calculating the likelihood of the data under the model.

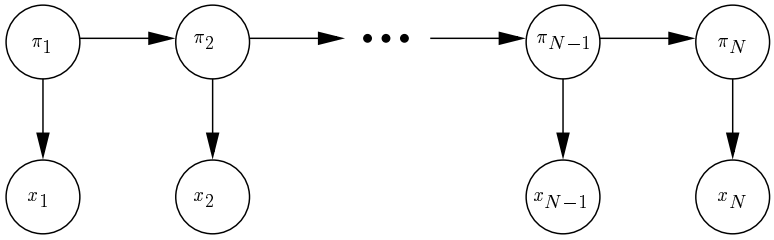


Figure B.1: Graphical model depiction of an HMM. Top nodes, π_i are the hidden state variables underlying each observation, x_i.

B.0.1 Formal Introduction to HMMs and Relevant Algorithms

Let $\boldsymbol{x}^k = x_1^k, x_2^k, \ldots, x_N^k$ be the k^{th} observed scalar time series of length N (although it could just as easily be a vector time series). Let the set of possible HMM states be $\mathcal{S} = \{s_1, s_2, \ldots, s_M\}$ (where M is specified ahead of time), and the (hidden) labels of the observed time series be $\boldsymbol{\pi}^k = \pi_1^k, \pi_2^k, \ldots, \pi_N^k$, where $\pi_i^k \in \mathcal{S}$. Let the state transition probabilities between states s_m and s_j be $T_{s_m,s_j} \equiv p(\pi_i = s_j | \pi_{i-1} = s_m)$, and let the probability of emitting x_i^k given that the state at time i is s_j be $A^k(i, s_j) \equiv p(x_i^k | \pi_i^k = s_j)$ (referred to as 'emission probabilities'). We

can now write down the log likelihood, $\mathcal{L}$, of the data under this model. Since the π_i^k are unobserved, we need to sum over all possible values (though we condition on the state transition probability and emission probability parameters, which we will learn).

$$\begin{aligned}
\mathcal{L} &= \log \prod_k p(\boldsymbol{x}^k | \{T_{s_m,s_j}\}, \{A^k(i,s_j)\}) \\
&= \sum_k \log p(\boldsymbol{x}^k | \{T_{s_m,s_j}\}, \{A^k(i,s_j)\}) \\
&= \sum_k \log \sum_{\boldsymbol{\pi}^k} p(\boldsymbol{x}^k, \boldsymbol{\pi}^k | \{T_{s_m,s_j}\}, \{A^k(i,s_j)\}) \\
&= \sum_k \log \sum_{\boldsymbol{\pi}^k} p(\boldsymbol{x}^k | \boldsymbol{\pi}^k, \{T_{s_m,s_j}\}, \{A^k(i,s_j)\}) p(\boldsymbol{\pi}^k | \{T_{s_i,s_j}\}) \\
&= \sum_k \log \sum_{\boldsymbol{\pi}^k} T_{0,\pi_i^k} \left(\prod_{i=1}^N A^k(i,\pi_i^k) \right) \left(\prod_{i=2}^N T_{\pi_i^k,\pi_{i-1}^k} \right) \qquad \text{(B.1)}
\end{aligned}$$

Notice that the joint likelihood for one time series factors into three terms, as a result of the conditional independencies implicit in the model (which can be deduced from Figure B.1 – see [35]),

$$p(\boldsymbol{x}^k, \boldsymbol{\pi}^k | \{T_{s_m,s_j}\}, \{A^k(i,s_j)\}) = T_{0,\pi_i^k} \left(\prod_{i=1}^N A^k(i,\pi_i^k) \right) \left(\prod_{i=2}^N T_{\pi_i^k,\pi_{i-1}^k} \right),$$

where the first term is related to the probability of starting in a particular state, the second to emitting a particular symbol in each state, and the third to transitioning between states. This factorized view can help one to better understand the recursive algorithms we present shortly in that it becomes clear that the joint is formed by 1) finding the probability of starting in the first state, 2) emitting a symbol, 3) transitioning to a new state, 4) emitting a symbol, 5) and so on and so forth, and that each of these actions is probabilistically factored from the others.

We will want to use maximum likelihood to estimate the parameters $\{T_{s_m,s_j}\}, \{A^k(i,s_j)\}$, but because HMMs (and the EM-CPM) is a latent (hidden) variable model, we use the Expectation-Maximization algorithm (referred to as the Baum-Welch algorithm in the context of HMMs [65]), introduced in Appendix C. However, in order to use Expectation-Maximization, we will need to be able to compute several probabilities. In particular, we will need to compute the marginals of the posterior probabilities of each state, $p(\pi_i^k | \boldsymbol{x}^k)$, and if we want to learn the transition probabilities, we will also need the pair-wise marginals of the posterior $p(\pi_i^k, \pi_{i+1}^k | \boldsymbol{x}^k)$. Each of these can be computed relatively efficiently in an HMM by relying on a Dynamic Programming algorithm called the Forward-Backward in the context of HMMs [65].

B.0.2 Forward-Backward algorithm

Suppose we wanted to calculate the likelihood of the data for a given parameter setting (*i.e.*, a setting of the state transition probability and emission probability parameters). Looking at Equation B.1, we see that naively, this would involve summing over all possible state sequences, $\boldsymbol{\pi}^k$, with a time complexity of $\Theta(KM^N)$, where K is the number of observed time series, each of length N, and M is the number of hidden states in the HMM. However, by realizing that many sub-paths are shared across state-sequences, this calculation can be performed far more efficiently. In particular, the following recursion will perform the required sum in time complexity $\Theta(KMN)$ [1]. Letting $\alpha^k(i,j) \equiv p(x_1^k, x_2^k, \ldots, x_i^k, \pi_i^k = s_j)$ (we drop the parameter conditioning in this section for notational clarity), we initialize and recurse (in the 'forward' direction, from $i = 1$ to $i = N$) as follows, for all k and j,

$$\alpha^k(1,j) = A^k(1,s_j)T_{0,s_j} \qquad \text{(initialization)}$$

$$\alpha^k(i,j) = A^k(i,s_j)\sum_{t=1}^{M}\alpha^k(i-1,t)T_{s_t,s_j} \qquad \text{(recursion)}$$

Thus, it follows that after the recursion is complete, we can compute

$$p(\boldsymbol{x}^k) = \sum_j p(x_1^k, \ldots, x_N^k, \pi_N^k = s_j) = \sum_j \alpha^k(N,j),$$

and thus the log likelihood as

$$\mathcal{L} = \sum_k \log p(\boldsymbol{x}^k) = \sum_k \log \sum_j \alpha^k(N,j).$$

But how can we compute the desired marginals of the posterior, $p(\pi_i^k|\boldsymbol{x}^k)$ and $p(\pi_i^k, \pi_{i+1}^k|\boldsymbol{x}^k)$, efficiently? Using Bayes' rule, and the rules of conditional probability we see that

$$\begin{aligned}
p(\pi_i^k = s_j|\boldsymbol{x}^k) &= \frac{p(\boldsymbol{x}^k|\pi_i^k = s_j)p(\pi_i^k = s_j)}{p(\boldsymbol{x}^k)} \\
&= \frac{p(x_1^k, \ldots, x_i^k|\pi_i^k = s_j)p(x_{i+1}^k, \ldots, x_N^k|\pi_i^k = s_j)p(\pi_i^k = s_j)}{p(\boldsymbol{x}^k)} \\
&= \frac{p(x_1^k, \ldots, x_i^k, \pi_i^k = s_j)p(x_{i+1}^k, \ldots, x_N^k|\pi_i^k = s_j)}{p(\boldsymbol{x}^k)} \\
&= \frac{\alpha^k(i,j)p(x_{i+1}^k, \ldots, x_N^k|\pi_i^k = s_j)}{p(\boldsymbol{x}^k)},
\end{aligned}$$

[1] The time complexity will be even less if the transition matrix is sparse.

and

$$\begin{aligned} p(\pi_i^k = s_j, \pi_{i+1}^k = s_t | \boldsymbol{x}^k) &= \frac{p(\boldsymbol{x}^k | \pi_i^k = s_j, \pi_{i+1}^k = s_t) p(\pi_i^k = s_j, \pi_{i+1}^k = s_t)}{p(\boldsymbol{x}^k)} \\ &= \frac{p(x_1^k, \ldots, x_{i-1}^k, \pi_{i-1}^k = s_j) p(x_{i+1}^k, \ldots, x_N^k | \pi_i^k = s_t) p(\pi_i^k = s_t | \pi_{i+1}^k = s_j)}{p(\boldsymbol{x}^k)} \\ &= \frac{\alpha^k(i-1, j) p(x_{i+1}^k, \ldots, x_N^k | \pi_i^k = s_t) T_{s_j, s_t}^k}{p(\boldsymbol{x}^k)}, \end{aligned}$$

and thus it becomes clear that doing a 'backward' recursion (from $i = N$ to $i = 1$) to calculate probabilities of the form $\beta^k(i, j) \equiv p(x_{i+1}^k, \ldots, x_N^k | \pi_i^k = s_j)$ would allow us to compute the desired probabilities. We do just that as follows. For all k and j,

$$\begin{aligned} \beta^k(1, j) &= 1 && \text{(initialization)} \\ \beta^k(i+1, j) &= \sum_{t=1}^{M} T_{s_j, s_t} A^k(i+1, s_t) \beta^k(i+1, t) && \text{(recursion)} \end{aligned}$$

Summary: We perform a forward (α) and a backward (β) recursion (belief propagation), giving us forward and backward 'messages', $\alpha^k(i, j)$ and $\beta^k(i, j)$, and then combine these messages to obtain the desired marginals of the posterior,

$$\begin{aligned} \alpha^k(i, j) &\equiv p(x_1^k, \ldots, x_i^k, \pi_i^k = s_j) \\ \beta^k(i, j) &\equiv p(x_{i+1}^k, \ldots, x_N^k | \pi_i^k = s_j) \\ p(\boldsymbol{x}^k) &= \sum_j \alpha^k(N, j) = \sum_j \alpha^k(i, j) \beta^k(i, j) \\ p(\pi_i^k = j | \boldsymbol{x}^k) &= \frac{\alpha^k(i, j) \beta^k(i, j)}{p(\boldsymbol{x}^k)} \\ p(\pi_{i-1}^k = s_j, \pi_i^k = s_t | \boldsymbol{x}^k) &= \frac{\alpha^k(i-1, j) T_{s_j, s_t}^k A^k(i, s_t) \beta^k(i, t)}{p(\boldsymbol{x}^k)}. \end{aligned}$$

Notice that $p(\boldsymbol{x}^k)$ can be computed at any time point i, providing a sanity check for debugging code.

B.0.3 Using the posterior of the hidden state sequence

So far we have shown how to efficiently compute the likelihood and marginals of the posterior – computations we will need when learning the parameters by way of EM. After training is complete, we will likely be interested in the posterior of the hidden state sequence, $p(\boldsymbol{\pi}^k | \boldsymbol{x}^k)$, which represents our belief about which state sequence may have generated the observed data,

after having seen the data (and trained the model). We can use the posterior either by taking its maximum, that is, the *maximum a posteriori* (MAP) sequence for every observed time series, which can be again be accomplished with Dynamic Programming. Alternatively, we may wish to sample from the posterior, which can be accomplished with a recursive stochastic traceback, as shown below.

MAP state sequence

To compute the MAP state sequence for the k^{th} observed time series, $\hat{\boldsymbol{\pi}}^k = \text{argmax}_{\boldsymbol{\pi}^k} p(\boldsymbol{\pi}^k|\boldsymbol{x}^k)$, it is helpful to first realize that $p(\boldsymbol{\pi}^k|\boldsymbol{x}^k) \propto p(\boldsymbol{\pi}^k, \boldsymbol{x}^k)$, since $p(\boldsymbol{\pi}^k|\boldsymbol{x}^k) = \frac{p(\boldsymbol{\pi}^k, \boldsymbol{x}^k)}{p(\boldsymbol{x}^k)}$. Thus if we can find the path that gives us the highest joint likelihood (which turns out to be more convenient), then this same path is the MAP path. Thus, we let $\rho^k(i,j)$ be the probability of the most probable (that with the highest joint likelihood) partial path, from the first time point, and ending at time i in state s_j. That is,

$$\rho^k(i,j) \equiv p(\check{\pi}_1^k, \ldots, \check{\pi}_{i-1}^k, \pi_i^k = s_j, x_1^k, x_2^k, \ldots, x_i^k),$$

where $\check{\pi}_i^k$ denotes the state at the i^{th} time point along this most probable path. Then if we knew the value of $\rho^k(N,j)$ for all j, we could easily find the last state in the MAP state sequence (via the joint),

$$\hat{\pi}_N^k = s_t, \quad \text{where } \; t = \underset{j}{\text{argmax}}\, \rho^k(N,j).$$

Also, if we knew the value of $\rho^k(N-1,j)$ for all j, we could then find the second last state in the MAP path (assuming we already computed the last state in the MAP path),

$$\hat{\pi}_{N-1}^k = s_t, \quad \text{where } \; t = \underset{j}{\text{argmax}}\, \rho^k(N-1,j) T_{s_j, \hat{\pi}_N^k},$$

and so on and so forth. Thus the MAP estimate can be computed by running a forward recursion to obtain ρ messages (a recursion identical to the α recursion except the summation operator is replaced by the maximum operator). We also keep track of which state transition lead us to the maximum at each time step, and then trace back this path to obtain the MAP state sequence, $\hat{\boldsymbol{\pi}}^k$. Specifically, we run the following recursion, keeping track of which argument the $\max$ operator returns each time so that we can do a linear-time trace-back after the recursion. For

all j and all k, and working from $i = 1$ to $i = N$,

$$\begin{aligned}
\rho^k(1,j) &= A^k(1,s_j)T_{0,s_j} && \text{(initialization)} \\
\rho^k(i,j) &= A^k(i,s_j)\max_t \rho^k(i-1,t)T_{s_t,s_j}, && \text{(recursion)} \\
\tau^k(i,j) &= \operatorname*{argmax}_t \rho^k(i-1,t)T_{s_t,s_j} && \text{(storage of backward pointers)}
\end{aligned}$$

and then fill in the MAP state path starting at time $i = N$ and working backward through the stored arguments of the max operator, $\tau^k(i,j)$,

$$\begin{aligned}
\hat{\pi}_N^k &= s_t, \quad \text{where } t = \operatorname*{argmax}_j \rho^k(N,j) && \text{(last time step)} \\
\hat{\pi}_i^k &= s_t, \quad \text{where } t = \operatorname*{argmax}_j \tau^k(i+1,j) && \text{(for } i = N-1 \text{ to } i = 1)
\end{aligned}$$

This algorithm for finding the MAP sequence in an HMM is known as the Viterbi algorithm.

Sampling from the posterior

If instead we wish to obtain a sample from the posterior, $\tilde{\boldsymbol{\pi}}^k$, we sample,

$$\begin{aligned}
\tilde{\boldsymbol{\pi}}^k &\sim p(\boldsymbol{\pi}^k|\boldsymbol{x}^k) \\
&= p(\pi_1^k|\pi_2^k,\ldots,\pi_N^k,\boldsymbol{x}^k)p(\pi_2^k,\ldots,\pi_N^k|\boldsymbol{x}^k) \\
&= p(\pi_1^k|\pi_2^k,\ldots,\pi_N^k,\boldsymbol{x}^k)p(\pi_2^k|\pi_3^k,\ldots,\pi_N^k,\boldsymbol{x}^k)\ldots p(\pi_N^k|\boldsymbol{x}^k) \\
&= \prod_{i=1}^N p(\pi_i^k|\pi_{i+1}^k,\ldots,\pi_N^k,\boldsymbol{x}^k),
\end{aligned}$$

which can be easily accomplished by a recursive, stochastic traceback after having run the Forward algorithm, as follows,

$$\begin{aligned}
\tilde{\pi}_N^k &\sim p(\pi_N^k|\boldsymbol{x}^k) = \alpha^k(N,\pi_N^k) && \text{(initialization)} \\
\tilde{\pi}_i^k &\sim p(\pi_i^k|\tilde{\pi}_{i+1}^k,\ldots,\tilde{\pi}_N^k,\boldsymbol{x}^k) && \text{(recursion)} \\
&= p(\pi_i^k|\tilde{\pi}_{i+1}^k,x_1^k,\ldots,x_i^k) && \text{(conditional independence)} \\
&\sim p(\pi_i^k,\tilde{\pi}_{i+1}^k,x_1^k,\ldots,x_i^k) \\
&= T_{\pi_i^k,\tilde{\pi}_{i+1}^k}\alpha^k(i,\pi_i^k)
\end{aligned}$$

Practical Issues

In practice, there are numerical problems that can arise in forward-backward and Viterbi. As more and more of the sequence is accounted for during these dynamic programming algorithms, the joint (or conditional) probabilities can become so small that care must be taken to avoid overflow. One is generally safe either by working in $\log$ space, or by re-scaling the messages as one moves along the sequence, keeping track of how we re-scale, and then correcting for this afterward. In our implementation, we perform Viterbi in $\log$ space, and use scaling for forward-backward. In particular, for the forward algorithm, after computing $\alpha^k(t,j)$ for a given t and all j, we compute $\rho(t) \equiv \sum_j \alpha^k(t,j)$, and then re-scale the messages, $\bar{\alpha}^k(t,j) = \alpha^k(t,j)/\rho(t)$. Then we modify the backward algorithm, by computing

$$\bar{\beta}^k(i+1,j) = \sum_{t=1}^{M} T_{s_j,s_t} A^k(i+1,s_t)\bar{\beta}^k(i+1,t)\frac{1}{\rho(i+2)},$$

and then use these modified forward and backward messages, $\bar{\beta}^k(i+1,j)$ and $\bar{\alpha}^k(t,j)$ as we did their unmodified counterparts.

Appendix C

The Expectation-Maximization Algorithm

The Expectation-Maximization algorithm, popularized by Dempster *et al.* in [18], and further generalized by Neal and Hinton in [57], is a procedure for finding a maximum likelihood parameter setting in the presence of hidden variables[1] when the structure of the likelihood takes on a particular form. The form of the likelihood required for EM to be of practical benefit is one in which it would be relatively easy to compute the best parameter values, *if we knew the setting of the hidden variables*, and also one in which the posterior over the hidden states can be efficiently computed. If the posterior over the hidden states cannot be efficiently computed, then one can appeal to approximate algorithms such as variational EM (see for example [34]). Also, by 'relatively easy to compute', we mean either that the solution is analytically straight-forward, or that a numerical optimization routine can be used (*i.e.*, that the appropriate function and derivatives can be computed). If this is not the case, one might still use EM with a generalized M-step which need only be able to find 'better' parameter settings.

EM makes use of the structure of the likelihood by iterating between an 'Expectation'/E-step and a 'Maximization'/M-step. Intuitively, the E-step can be seen to be 'completing' the 'missing' data (hidden variables). However, rather than finding one good setting for these data, it actually provides a full distribution over possible values of these data (that is, it finds the posterior over the hidden variables). The M-Step then proceeds by assuming that the E-step has done a reasonable job of completing the missing data, and then uses standard maximum likelihood techniques on the newly completed data. While this may seem like a chicken and egg problem, it can be shown (*e.g.*, [18]) that iterating between these two steps is guaranteed to monotonically increase the log likelihood. One must guess at an initial setting of the parameters

[1] Traditionally, *parameters* refer to those variables in the model that are unobserved, and for which we would like to obtain a single best point-estimate, while *hidden variables* refer to those variables in the model that are unobserved, and which we would like to integrate out of the solution, or, possibly, obtain their posterior distribution – that is, we want to account for the uncertainty in their values. We use this terminology convention in this thesis.

to start off the process, and EM can be quite sensitive to this initialization.

For the EM-CPM, it turns out that we can compute the posterior over the hidden variables (states) exactly (as in an HMM), and hence do not need variational or other techniques. For the M-step, some updates are analytical, while others rely on iterative numerical solvers.

We now briefly sketch a derivation of the EM algorithm for a generic, probabilistic, generative model with parameter vector, $\boldsymbol{\phi}$, K observed, real-valued data points, x_i for $i = 1..K$, and corresponding, real-valued hidden variables, π_i. For example, in trying to cluster data using a mixture of gaussians, the x_i would be the data we are trying to cluster, the π_i would be the cluster label associated with each data point, and $\boldsymbol{\phi}$ would contain the mean, variance and mixing proportion parameters for each gaussian component. We assume that the data is identically and independently distributed (i.i.d.) and that the probabilistic model has been specified so as to easily be able to write an expression for $p(x_i, \pi_i|\boldsymbol{\phi})$, as would be the case for most directed, acyclic, graphical models (DAGs). We seek a maximum likelihood solution – that is, we want to find parameter settings, $\boldsymbol{\phi}'$ which maximize the likelihood of the data, $\mathcal{L}$, given by

$$\mathcal{L} = p(x_1, x_2, \ldots, x_K|\boldsymbol{\phi}) \tag{C.1}$$

$$= \prod_{i=1}^{K} p(x_i|\boldsymbol{\phi}) \tag{C.2}$$

$$= \prod_{i=1}^{K} \int_{\pi_i} p(x_i, \pi_i|\boldsymbol{\phi})\, d\pi_i \tag{C.3}$$

To perform maximum likelihood, we maximize with respect to the log likelihood. In the presence of hidden variables, the integral over the hidden parameters prevents us from easily doing this. However, appealing to a few tricks, we can get around this problem. We will first show how to bound the log likelihood from below, and subsequently, allude to how this bound gives us the EM algorithm. The first trick will be to let $q(\pi_i|x_i)$ be an arbitrary probability distribution, and then to multiply and divide by this factor. Then we will appeal to Jensen's inequality which states that if f is a convex function (such as the log function) and X is a random variable, then $f(E_{q(x)}[X]) \geq E_{q(x)}[f(X)]$, where $E[.]_{q(x)}$ denotes expectation with respect to the

probability distribution $q(x)$. Now we are ready to derive the lower bound on the log likelihood:

$$\log \mathcal{L} = \sum_{i=1}^{K} \log \int_{\pi_i} p(x_i, \pi_i|\boldsymbol{\phi})\, d\pi_i \tag{C.4}$$

$$= \sum_{i=1}^{K} \log \int_{\pi_i} q(\pi_i|x_i) \frac{p(x_i, \pi_i|\boldsymbol{\phi})}{q(\pi_i|x_i)}\, d\pi_i \tag{C.5}$$

$$\geq \sum_{i=1}^{K} \int_{\pi_i} q(\pi_i|x_i) \log \frac{p(x_i, \pi_i|\boldsymbol{\phi})}{q(\pi_i|x_i)}\, d\pi_i \quad \text{(by Jensen's inequality)} \tag{C.6}$$

$$= \sum_{i=1}^{K} \left(\int_{\pi_i} q(\pi_i|x_i) \log p(x_i, \pi_i|\boldsymbol{\phi})\, d\pi_i - \int_{\pi_i} q(\pi_i|x_i) \log q(\pi_i|x_i)\, d\pi_i \right) \tag{C.7}$$

$$= \sum_{i=1}^{K} \left(E_{q(\pi_i|x_i)}[\log p(x_i, \pi_i|\boldsymbol{\phi})] + \mathcal{H}(q(\pi_i|x_i)) \right) \tag{C.8}$$

$$= -F(q, \boldsymbol{\phi}) \tag{C.9}$$

$F(q(\pi_i|x_i), \boldsymbol{\phi})$ is referred to for historical (statistical physics) reasons as the Free Energy, $H(q)$ denotes the entropy of distribution q, and $p(x_i, \pi_i|\boldsymbol{\phi})$ is referred to as the *complete likelihood*, $E_{q(\pi_i|x_i)}[\log p(x_i, \pi_i|\boldsymbol{\phi})]$ is the *expected complete log likelihood*. It turns out that the bound is satiated – that is, $-F(q, \boldsymbol{\phi}) = \log \mathcal{L}$, when $q(\pi_i|x_i) = p(\pi_i|x_i, \boldsymbol{\phi})$, as can readily be seen by plugging this in to the Free Energy equation, and using $p(\pi_i|x_i, \boldsymbol{\phi}) = \frac{p(x_i, \pi_i|\boldsymbol{\phi})}{p(x_i|\boldsymbol{\phi})}$, as follows,

$$-F(p(\pi_i|x_i, \boldsymbol{\phi}), \boldsymbol{\phi}) = \sum_{i=1}^{K} \int_{\pi_i} p(\pi_i|x_i, \boldsymbol{\phi}) \log \frac{p(x_i, \pi_i|\boldsymbol{\phi})}{p(\pi_i|x_i, \boldsymbol{\phi})}\, d\pi_i \tag{C.10}$$

$$= \sum_{i=1}^{K} \int_{\pi_i} p(\pi_i|x_i, \boldsymbol{\phi}) \log \frac{p(x_i, \pi_i|\boldsymbol{\phi}) p(x_i|\boldsymbol{\phi})}{p(x_i, \pi_i|\boldsymbol{\phi})}\, d\pi_i \tag{C.11}$$

$$= \sum_{i=1}^{K} \int_{\pi_i} p(\pi_i|x_i, \boldsymbol{\phi}) \log p(x_i|\boldsymbol{\phi})\, d\pi_i \tag{C.12}$$

$$= \sum_{i=1}^{K} \log p(x_i|\boldsymbol{\phi}) \int_{\pi_i} p(\pi_i|x_i, \boldsymbol{\phi})\, d\pi_i \tag{C.13}$$

$$= \sum_{i=1}^{K} \log p(x_i|\boldsymbol{\phi}). \tag{C.14}$$

Thus, if an oracle told us the correct posteriors over the hidden variables, $p(\pi_i|x_i, \hat{\boldsymbol{\phi}})$ (where $\hat{\boldsymbol{\phi}}$ is be the maximum likelihood parameter setting), we could could maximize the Free Energy with respect to the parameters, $\boldsymbol{\phi}$, to obtain the maximum likelihood parameter settings for

our model. This would be equivalent to maximizing the expected complete log likelihood with respect to the parameters (since we would have fixed $q(\pi_i|x_i) = p(\pi_i|x_i, \boldsymbol{\phi})$ and thus the entropy term would be irrelevant). Alternatively, if an oracle told us the maximum likelihood parameter settings, $\boldsymbol{\phi}'$, we would be able to compute the posterior of our hidden variables simply by using Bayes rule. Intuitively, the EM algorithm works by pretending that an oracle has told us the maximum likelihood parameter settings, and then calculating the the posterior over the hidden variables (E-step). Only, after the posterior has been computed, the algorithm realizes that the oracle was a bit off, and so goes back to update the parameter setting, based on the current estimate of the posterior over the hidden variables (M-step). This procedure is doing coordinate ascent in the Free Energy space, and when it reaches a local maximum of the Free Energy, the bound must have been satiated, and hence that the local maximum is also a local maximum of the log likelihood. Thus the procedure, if iterated until convergence, is guaranteed to find a local maximum of the log likelihood function.

When deriving one's own EM algorithm for a particular model, one needs to calculate the posterior over the hidden states (or marginals of the posterior, as with HMMs – see Section 5.5) for the E-Step. For the M-Step, one keeps the posterior from the E-Step fixed, and calculates the optimal parameter settings for the expected complete log likelihood. M-Step updates, when analytical, look very much like regular maximum likelihood updates in a non-latent variable model. For example, in a Mixture of Gaussians model (the probabilistic version of k-means clustering), when updating the mean parameter for each cluster, one takes a posterior-weighted average of the data points, where the posterior indicates the belief that a particular data point belongs to a particular cluster. If the posterior had mass at only one cluster, then this calculation would reduce to the simple average of each point assigned to the cluster being updated. This phenomenon occurs in most analytical M-Steps, where the update looks like a weighted version of a regular maximum likelihood type update. This is apparent in Section 5.5 which gives the M-Step updates for the EM-CPM.

Appendix D

M-Step Derivations for the EM-CPM

We now derive each of the M-Steps that are reported in Section 4.2. Recall that we have K samples, S states, D dimensions at each time point. The marginals of the posteriors are abbreviated as $\gamma_s^k(i) \equiv p(\pi_i^k = s|\boldsymbol{\mathcal{X}}^k)$ and $\xi_{s,s'}^k(i) \equiv p(\pi_{i-1}^k = s, \pi_i^k = s'|\boldsymbol{\mathcal{X}}^k)$. These are computed during the E-step, and held constant during the M-step. Recall then that the expected complete log likelihood for the data given our model is given by

$$\left\langle \mathcal{L}_{\text{comp}}^p \right\rangle = \mathcal{P} + \sum_{s=1}^{S}\sum_{k=1}^{K} \gamma_s^k(1) \log T_{0,s}^k + \sum_{s=1}^{S}\sum_{i=1}^{N}\sum_{d=1}^{D} \gamma_s^k(i) \log \mathcal{N}(\mathcal{X}_{i,d}^k | \mathcal{Z}_{\tau_s,d}^{\omega^k} \phi_s u^k, \xi_d^k) + \ldots$$
$$\ldots + \sum_{s=1}^{S}\sum_{s' \neq s} \log T_{s,s'}^k \sum_{i=2}^{N} \xi_{s,s'}^k(i),$$

where $\mathcal{P}$ is defined in Equation 4.13 as

$$\mathcal{P} \equiv -\lambda \sum_{c=1}^{C} \bar{u}^c \sum_{j=1}^{M-1}\sum_{d=1}^{D} (\mathcal{Z}_{j+1,d}^c - \mathcal{Z}_{j,d}^c)^2 - \nu \sum_{c=1}^{C}\sum_{c'<c}\sum_{d=1}^{D}\sum_{j=1}^{M} \log\left(1 + \frac{(\mathcal{Z}_{j,d}^c - \mathcal{Z}_{j,d}^{c'})^2}{\varsigma}\right) + \ldots$$
$$\ldots + \sum_{k=1}^{K} \log \mathcal{D}(\kappa_v^k | \{\eta_v^k\}) + \log \mathcal{D}(s_v | \{\eta_v'\}) + \sum_{k=1}^{K} \log \mathcal{N}(\log u_k | 0, w),$$

where $\bar{u}^c \equiv \frac{\sum_{k|\omega^k=c} {u_k}^2}{K}$.

We now derive the parameter updates for each parameter in turn, namely, $\{\boldsymbol{\mathcal{Z}}^c\}$, $\{u^k\}$, $\{\kappa_i^k\}$ s_0, s_1 and $\{\xi_d^k\}$ by finding the value of each parameter which maximizes $\left\langle \mathcal{L}_{\text{comp}}^p \right\rangle$. In the case where the update is analytical, we provide the update, and in the case where it is not, we provide the derivative which can be fed to a numerical solver.

The M-Step updates for u^k, ξ_d^k, and $\boldsymbol{\mathcal{Z}}^c$ are coupled (the value of one appears in the derivate with respect to the other). Thus we arbitrarily pick an order to update them and as one is

updated, its new values are used in the updates for the coupled parameter updates that follow it. This procedure maintains the guarantee of convergence that EM provides.

D.1 Latent Trace M-Step

Let $\boldsymbol{\mathcal{Z}}^{\boldsymbol{c}}_{\boldsymbol{d},:} \equiv \mathcal{Z}^{c}_{d,1}, \ldots, \mathcal{Z}^{c}_{d,N}$ and consider the following derivative

$$
\begin{aligned}
\frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \left\langle \mathcal{L}^{p}_{\text{comp}} \right\rangle &= \frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{i=1}^{N} \sum_{d=1}^{D} \gamma^{k}_{s}(i) \log \mathcal{N}(\mathcal{X}^{k}_{i,d} | \, \mathcal{Z}^{\omega^k}_{\tau_s,d} \phi_s u^k, \xi^{k}_{d}) - \ldots \\
&\ldots - \frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \lambda \sum_{c=1}^{C} \bar{u}^{c} \sum_{j=1}^{M-1} \sum_{d=1}^{D} (\mathcal{Z}^{c}_{j+1,d} - \mathcal{Z}^{c}_{j,d})^2 - \ldots \\
&\ldots - \frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \nu \sum_{c=1}^{C} \sum_{c^*<c} \sum_{d=1}^{D} \sum_{j=1}^{M} \log \left(1 + \frac{(\mathcal{Z}^{c}_{j,d} - \mathcal{Z}^{c^*}_{j,d})^2}{\varsigma} \right) \\
&= \sum_{\{k|\omega^k=c'\}} \sum_{\{s|\tau_s=j'\}} \sum_{i=1}^{N} \gamma^{k}_{s}(i) \frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \frac{(\mathcal{X}^{k}_{i,d'} - \mathcal{Z}^{\omega^k}_{\tau_s,d'} \phi_s u^k)^2}{2\xi^{k}_{d'}} - \ldots \\
&\ldots - \lambda \bar{u}^{c'} \left(-2(\mathcal{Z}^{c'}_{j'+1,d'} - \mathcal{Z}^{c'}_{j',d'}) + 2(\mathcal{Z}^{c'}_{j',d'} - \mathcal{Z}^{c'}_{j'-1,d'}) \right) - \ldots \\
&\ldots - \nu \sum_{c^*<c'} \frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \log \left(1 + \frac{(\mathcal{Z}^{c'}_{j',d'} - \mathcal{Z}^{c^*}_{j',d'})^2}{\varsigma} \right) \\
&= - \sum_{\{k|\omega^k=c'\}} \sum_{\{s|\tau_s=j'\}} \sum_{i=1}^{N} 2\gamma^{k}_{s}(i) \phi_s u^k \frac{(\mathcal{X}^{k}_{i,d'} - \mathcal{Z}^{\omega^k}_{\tau_s,d'} \phi_s u^k)}{2\xi^{k}_{d'}} - \ldots \\
&\ldots - \lambda \bar{u}^{c'} \left(4\mathcal{Z}^{c'}_{j',d'} - 2\mathcal{Z}^{c'}_{j'-1,d'} - 2\mathcal{Z}^{c'}_{j'+1,d'} \right) - \ldots \\
&\ldots - \nu \sum_{c^*\neq c'} \left(1 + \frac{(\mathcal{Z}^{c'}_{j',d'} - \mathcal{Z}^{c^*}_{j',d'})^2}{\varsigma} \right)^{-1} \left(\frac{2(\mathcal{Z}^{c'}_{j',d'} - \mathcal{Z}^{c^*}_{j',d'})}{\varsigma} \right).
\end{aligned}
$$

When $\nu = 0$ (*i.e.*, in the single-class case), the derivative $\frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \left\langle \mathcal{L}^{p}_{\text{comp}} \right\rangle$ does not depend on the latent trace values for any class other than c', nor on any dimension other than d'. Furthermore, it only depends on the components $\mathcal{Z}^{c'}_{j',d'}$, $\mathcal{Z}^{c'}_{j'-1,d'}$, and $\mathcal{Z}^{c'}_{j'+1,d'}$. Thus, we can compute
$\frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \left\langle \mathcal{L}^{p}_{\text{comp}} \right\rangle$ for all $j' = 1..M$, and setting each to zero, we obtain M linear equations in M unknowns. The system is tri-diagonal, and easily solved by any standard linear solver such as the backslash operator in Matlab. If additionally, $\lambda = 0$, then we simply have M independent equations, each in one unknown, and the system can be solved even faster.

When $\nu \neq 0$, all values of all latent traces across all classes become coupled, and in a

non-linear fashion. The system is no longer analytically tractable. In this situation we provide the derivative functions, $\frac{\partial}{\partial \mathcal{Z}^{c'}_{d',j'}} \left\langle \mathcal{L}^{p}_{\text{comp}} \right\rangle$, and also the expected complete log likelihood function to a numerical minimization routine such as conjugate gradient. Note that in this case, each dimension is still independent of every other dimension, and thus we can solve for each dimension seperately. Alternatively to using a numerical solver in this situation, one could update each latent trace, conditioned on the current value of the others, although we did not use this approach.

D.2 HMM Emission Variance M-Step

The HMM emission variances, ξ^k_d for each observed time series, k, and each dimension of the signal, d are treated as independent from one another in the model. Thus each can be updated independently from the other, and this can be done analytically, as follows:

$$\begin{aligned}
\frac{\partial}{\partial \xi^{k'}_{d'}} \left\langle \mathcal{L}^{p}_{\text{comp}} \right\rangle &= \frac{\partial}{\partial \xi^{k'}_{d'}} \sum_{s=1}^{S} \sum_{i=1}^{N} \gamma^k_s(i) \log \mathcal{N}(\mathcal{X}^{k'}_{i,d'} | \mathcal{Z}^{\omega^{k'}}_{\tau_s,d'} \phi_s u^{k'}, \xi^{k'}_{d'}) \\
&= \sum_{s=1}^{S} \sum_{i=1}^{N} \gamma^k_s(i) \frac{\partial}{\partial \xi^{k'}_{d'}} \left[\log \frac{1}{\sqrt{\xi^{k'}_{d'} 2\pi}} - \frac{(\mathcal{X}^{k'}_{i,d'} - \mathcal{Z}^{\omega^{k'}}_{\tau_s,d'} \phi_s u^{k'})^2}{2\xi^{k'}_{d'}} \right] \\
&= \sum_{s=1}^{S} \sum_{i=1}^{N} \gamma^k_s(i) \left[-\frac{1}{2\xi^{k'}_{d'}} + \frac{(\mathcal{X}^{k'}_{i,d'} - \mathcal{Z}^{\omega^{k'}}_{\tau_s,d'} \phi_s u^{k'})^2}{2(\xi^{k'}_{d'})^2} \right],
\end{aligned}$$

which we then set equal to zero and solve for $\xi^{k'}_{d'}$, giving

$$\xi^{k'}_{d'} = \frac{1}{N} \sum_{s=1}^{S} \sum_{i=1}^{N} \gamma^k_s(i) (\mathcal{X}^{k'}_{i,d'} - \mathcal{Z}^{\omega^{k'}}_{\tau_s,d'})^2.$$

See Section 4.5.2 for the post-processing step used here which implements an extra set of constraints on the $\xi^{k'}_{d'}$.

D.3 Time State Transition Parameter M-Step

The time transition parameters are the the parameters of a multinomial, and must add up to one, that is $\sum_{j=1}^{J} \kappa^k_j = 1$. Thus we need to perform constrained minimization, and can do so by way of Lagrange multipliers which use the fact that the gradient of our constraint function should be parallel to the gradient of the function we seek to minimize, at the point at which our solution lies (if the constraint is active). Thus we require the following equations, where, Λ_k

are Lagrange multipliers,

$$\begin{aligned}\frac{\partial}{\partial \kappa_v^k}\left(\langle \mathcal{L}_{\text{comp}}^p \rangle - \Lambda_k \sum_{j=1}^{J} \kappa_j^k\right) &= \frac{\partial}{\partial \kappa_v^k}\left(\log \mathcal{D}(\kappa_v^k|\{\eta_v^k\}) + \sum_{s=1}^{S}\sum_{s'=1}^{S}\sum_{i=2}^{N} \xi_{s,s'}^k(i) \log T_{s,s'}^k\right) - \Lambda_k \kappa_v^k \\ &= \frac{\partial}{\partial \kappa_v^k} \log \sum_{j=1}^{J} (\kappa_j^k)^{\eta_j^k} + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i) \frac{\partial}{\partial \kappa_v^k} \log T_{s,s'}^k - \Lambda_k \\ &= \frac{\eta_v^k}{\kappa_v^k} + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i) \frac{1}{\kappa_v^k} - \Lambda_k\end{aligned}$$

For every k, we have J such equations (for $v \in 1..J$). Setting each of these J derivative to zero, multiplying each one through by κ_v^k, and adding the resulting equations together, we obtain:

$$\sum_{v=1}^{J}\left(\eta_v^k + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i) - \kappa_v^k \Lambda_k\right) = 0.$$

Then using the constraint that $\sum_{j=1}^{J} \kappa_j^k = 1$, we can solve for Λ_k, and then for κ_v^k as follows,

$$\Lambda_k = \sum_{v=1}^{J}\sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i) + \sum_{v=1}^{J} \eta_v^k \tag{D.1}$$

$$\kappa_v^k = \frac{\eta_v^k + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i)}{\Lambda_k} \tag{D.2}$$

$$\kappa_v^k = \frac{\eta_v^k + \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=v\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i)}{\sum_{j=1}^{J} \eta_j^k + \sum_{j=1}^{J} \sum_{s=1}^{S} \sum_{\{s'|\tau_{s'}-\tau_s=j\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i)} \tag{D.3}$$

We see that the Dirichlet prior parameters end up as pseudo-count data.

D.4 Scale State Transition Parameter M-Step

The derivation for the updates for the scale transition probabilities are directly analogous to those of the time state transition parameters, and are given by

$$s_v = \frac{\eta_v' + \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{\{s' \in H(s,v)\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i)}{\sum_{v=1}^{2} \eta_v' + \sum_{k=1}^{K} \sum_{s=1}^{S} \sum_{\{s' \in H(s,1),H(s,0)\}} \sum_{i=2}^{N} \xi_{s,s'}^k(i)}$$

where $H(s,v) \equiv \{s'|s' \text{is exactly } v \text{ scale states away from } s\}$, and $v \in 0..1$.

D.5 Global Scaling Constant M-Step

The update for each u^k is independent of the others. With the prior that we have put on the u^k, the update is not analytically tractable, although without the prior, it is. We now compute the derivative with respect to the expected complete log likelihood:

$$\begin{aligned}\frac{\partial \left\langle \mathcal{L}^p \right\rangle_\pi}{\partial u_k} &= \sum_{s=1}^{S}\sum_{i=1}^{N}\sum_{d=1}^{D} \gamma_s^k(i) \frac{\partial}{\partial u_k} \log \mathcal{N}(\mathcal{X}_{i,d}^k | \mathcal{Z}_{\tau_s,d}^{\omega^k} \phi_s u^k, \xi_d^k) + \frac{\partial}{\partial u_k} \log\left(\frac{1}{u^k}\mathcal{N}(\log u_k; 0, w)\right) \\ &= -\sum_{s=1}^{S}\sum_{i=1}^{N}\sum_{d=1}^{D} \gamma_s^k(i) \frac{\partial}{\partial u_k} \frac{(x_i^k - z_{\tau_s}\phi_s u_k)^2}{2\xi_k} - \frac{\partial}{\partial u_k}\log u^k - \frac{\partial}{\partial u_k}\left(\frac{(\log u_k - 0)^2}{2w}\right) \\ &= \sum_{s=1}^{S}\sum_{i=1}^{N}\sum_{d=1}^{D} \gamma_s^k(i) \frac{z_{\tau_s}\phi_s(x_i^k - z_{\tau_s}\phi_s u_k)}{\xi_k} - \frac{1}{u^k} - \frac{\log u_k}{w^2 u_k}.\end{aligned}$$

With no prior on the u^k, the solution is analytical, with a solution,

$$u_k = \frac{\sum_{d=1}^{D}\sum_{s=1}^{S} z_{\tau_s}\phi_s \sum_{i=1}^{N} \gamma_s^k(i) x_i^k}{\sum_{d=1}^{D}\sum_{s=1}^{S}(z_{\tau_s}\phi_s)^2 \sum_{i=1}^{N}\gamma_s^k(i)}$$

However, we include the prior, and hence resort to use of a numerical minmization routine.

D.6 Scaling Spline M-Step

For the M-Step update of the scaling spline parameters, we can make use of the following derivative,

$$\begin{aligned}\frac{\partial \left\langle \mathcal{L}^p_{\text{comp}} \right\rangle}{\partial \mu_m^k} = &-\left(\frac{2\mu_m^k\lambda}{D|\{k'|\omega^{k'}=\omega^k\}|}\sum_{d=1}^{D}\sum_{j=1}^{M-1}(\mathcal{Z}_{j+1,d}^{\omega^k} - \mathcal{Z}_{j,d}^{\omega^k})^2\right) - \frac{\log\mu_m^k}{w^2\mu_m^k} - \frac{1}{\mu_m^k} + \dots \\ &\dots + \sum_{d=1}^{D}\sum_{s\in S_m^-}\frac{\mathcal{Z}_{s,d}\phi_s}{\xi_d^k(c_s^+ - c_s^-)}(s - c_s^-)\sum_{i=1}^{N}\gamma_s^k(i)\mathcal{X}_{i,d}^k - \dots \\ &\dots - \sum_{d=1}^{D}\sum_{s\in S_m^-}\frac{(\mathcal{Z}_{s,d}\phi_s)^2 u_{\tau_s}^k}{\xi_d^k(c_s^+ - c_s^-)}(s - c_s^-)\sum_{i=1}^{N}\gamma_s^k(i) + \dots \\ &\dots + \sum_{d=1}^{D}\sum_{s\in S_m^+}\frac{\mathcal{Z}_{s,d}\phi_s}{\xi_d^k(c_s^+ - c_s^-)}(c_s^+ - s)\sum_{i=1}^{N}\gamma_s^k(i)\mathcal{X}_{i,d}^k - \dots \\ &\dots - \sum_{d=1}^{D}\sum_{s\in S_m^+}\frac{(\mathcal{Z}_{s,d}\phi_s)^2 u_{\tau_s}^k}{\xi_d^k(c_s^+ - c_s^-)}(c_s^+ - s)\sum_{i=1}^{N}\gamma_s^k(i),\end{aligned}$$

where S_m^+ denotes the set of time states which lie in the integer interval $[m, c_m^+)$ and S_m^- denotes the set of time states which lie in the integer interval (c_m^-, m).

Appendix E

The Bayesian Paradigm and Sampling Methods

We present here a *very brief* introduction to the Bayesian modeling paradigm, and to a few sampling algorithms used in this context, as these topics pertain to development of the HB-CPM in Chapter 5. We refer the reader to [13, 29, 54, 59] for broader, more detailed and more rigorous exposition of these topics.

E.1 The Bayesian Modeling Paradigm

Suppose we have observed data, x, and a probabilistic model with parameters or hidden variables a, which specify a joint probability distribution $p(x|a)$. The *posterior* over the parameters is given by Bayes' rule:

$$p(a|x) = \frac{p(a)p(x|a)}{p(x)}. \tag{E.1}$$

The posterior informs us how likely different parameter settings are, after we have seen the data, x, and merged this information with our prior beliefs about different parameter settings. The factor $p(x|a)$ is called the *likelihood* function, and can be thought of as a measure of goodness-of-fit of the model $p(x|a)$ with parameter settings, a, to the data. The likelihood is a *function* of the parameters a, derived from a probability *distribution* over the space of data, x. The term $p(a)$ is called the *prior* over the parameters, and it specifies how likely we believe a particular parameter setting to be before seeing any of the data (that is, *a priori*). For a given

problem, $p(x)$ is typically fixed (since the training data does not change), and

$$p(a|x) \propto \text{prior} \times \text{likelihood}.$$

In *maximum likelihood learning*, we seek the single parameter setting (that is, the setting of a) which optimizes just the likelihood. We might then use this single parameter setting for the task at hand (*e.g.*, regression, classification, *etc.*). However, if the model and parameter space is very expressive, this approach can overfit the model, in which case generalization to new unseen data will be poor. To overcome this, one can use a variety of techniques, all of which seek to *regularize* the model – that is, they seek to constrain the complexity of the model, for example by constraining the parameter settings in some way (*e.g.*, requiring a diagonal or spherical covariance matrix rather than a full covariance matrix, or forcing neural network weights to be shared, or using 'early-stopping', *etc.*). One approach to regularization is the MAP (*maximum a posteriori*) approach, in which one seeks the single parameter setting which maximizes the posterior, given fixed data. The MAP approach can be viewed as an instantiation of the Occam's Razor principle which states that simpler explanations are preferable to more complicated ones, all else being equal: the likelihood says that we want a good fit of the data to the model, but we balance this objective with the desire for a simpler model by penalizing 'complicated' parameter settings through the prior, where it is up to us to define 'complicated'. MAP is sometimes referred to as penalized maximum-likelihood, though the latter need not necessarily have correctly specified priors in the sense that they need not be well-defined distributions.

In contrast to maximum-likelihood and MAP estimation, the fully[1] Bayesian approach seeks the full posterior distribution over the parameters, so as to use this full distribution for the task at hand, rather than using just a single 'best' parameter setting. For example, suppose we have a density estimation problem and that a contains the parameters of the model, $p(x|a)$, and that x is our training data. Assume that we have somehow computed the posterior distribution $p(a|x)$. Then, to predict density of a new case, x^*, we make use of the 'trained' prior, $p(a|x)$, which will have taken into account the training data:

$$\begin{aligned} p(x^*|x) &= \int_a p(x^*, a|x)\, da \quad \text{(marginalization)} \\ &= \int_a p(x^*|a, x) p(a|x)\, da \quad \text{(chain rule of probabilities)} \\ &= \int_a p(x^*|a) p(a|x)\, da \quad \text{(independence of } x^* \text{ and } x \text{ given } a) \end{aligned} \tag{E.2}$$

[1]We say 'fully' because sometimes people refer to use of Bayes' rule in the MAP approach as a Bayesian approach.

where the Equation E.2 is allowed because we assume that that the test and training data are independently distributed. Recognize that $p(a|x)$ is the posterior of the parameters resulting from the training data, and acts as the prior at test time. In contrast to this fully Bayesian approach, a MAP practitioner would not integrate out over parameter space, but would find the single 'best' parameter setting during training,

$$\begin{aligned}\hat{a} &= \underset{a}{\operatorname{argmax}}\, p(a|x) \quad \text{(maximum value of the posterior)} \\ &= \underset{a}{\operatorname{argmax}}\, \frac{p(x|a)p(a)}{p(x)} \quad \text{(Bayes' rule)} \\ &= \underset{a}{\operatorname{argmax}}\, p(x|a)p(a) \quad \text{(fixed training data)}\end{aligned}$$

and then predict the density of a test point, x^*, by directly computing $p(x^*|\hat{a})$, which is the trained model evaluated at the test point.

The take-home point is that the Bayesian practitioner seeks to obtain and use the full posterior distribution over the model parameters, and not make do with just point-estimates of the parameters. Advantages of this approach include being able to obtain uncertainty estimates of the parameter settings, and also being able to use expressive model classes without worrying about overfitting, because the richer the model, the larger a parameter space will have to be integrated over, and hence very unlikely parameter settings with high-likelihood will be blurred out in the integral and not given much weight in the final answer, unless truly warranted by the data. In a sense, a fully Bayesian analysis can be thought of as using a stability measure of the parameter settings to gauge their utility (as one might intuitively do otherwise) – a parameter setting with high likelihood which is unstable to small perturbations will tend not to have much posterior probability mass and hence tend not to have much influence on the final solution.

Note that as more observed data becomes available, the role of the prior is diminished, and the posterior is likely to become more peaked, and hence better approximated by its MAP value. Thus the Bayesian paradigm is one especially well suited to limited-data scenarios, or scenarios where the complexity of the model thought to be appropriate to the modeling task is 'greater' than the amount of data available to train it.

Detractors of the Bayesian paradigm argue that specifying a prior introduces bias into what should be a neutral, scientific approach, while Bayesian practitioners retort that non-Bayesian, 'neutral', methods make equally strong assumptions, only that these are not made explicit and hence are potentially even more dangerous. A further discussion of these issues is beyond the scope of this thesis.

E.2 Estimation of the Posterior by Sampling

We have boiled down the fully Bayesian approach to a requirement for finding the full posterior over the parameters, $p(a|x)$. Most models of real practical utility will be complicated enough that the posterior distribution is not analytically tractable, and thus one must resort to sampling techniques in which the full distribution is approximated by a collection of samples $\{a_i\}$, which are chosen in such a way that their relative occurrence in the finite set asymptotically approaches their relative probabilities in the posterior distribution, in the limit as the size of the set becomes infinite.

Sampling is a well-developed area, with much wider applicability than Bayesian inference in probabilistic models. A nice introduction is provided in [54], while a comprehensive report about sampling in the context of Bayesian inference, along with related theory, is given in [59]. We now go over just the intuitive basics of sampling algorithms, and then introduce Gibbs sampling and slice sampling, both used in Chapter 5.

Monte Carlo methods refer to methods which approximate analytical solutions by using a large number of synthetically and stochastically generated samples to calculate an answer, such as when approximating the posterior distribution by a finite collection of samples and then making use of these samples to say predict the label of a test case in a classification problem.

Some sampling techniques choose samples without any reliance on what samples have been previously chosen. These methods include *importance sampling* and *rejection sampling*, and are known generally not to be effective in high dimensional parameter spaces. *Markov Chain Monte Carlo* (MCMC) methods are sampling methods in which the value of a sample chosen can depend on the previously chosen sample (*i.e.*, in a Markov manner). A *chain* refers to the sequence of states that have been sampled in such a manner. Gibbs sampling is an MCMC method and is very widely used; we discuss it in more detail shortly.

On an intuitive level, one can think of sampling as starting at any random state in the sampling space (*e.g.*, in the space of parameters a in our example), and then following a set of stochastic transition rules to obtain a new sample. If the transition rules are selected in particular ways, then our desired asymptotic results will hold (see [59]). Formally, we want to construct a *Markov chain* whose *stationary distribution* is the distribution we want to sample from. The different ways in which valid Markov chains can be constructed lead to different sampling algorithms. Two conditions for constructing Markov chains can loosely be described as:

1. The Markov chain leaves the posterior distribution *invariant*. This is often accomplished by assuring the stronger condition of *detailed balance/reversibility* which says that the probability that we will be in state x, and then transition to state x', is the same as the

probability that we will be in state x' and then transition to state x. In other words, one generates the same distribution of trajectories whether going 'forward' or 'backward', and the transition rules are thus directionless.

2. The transition rules give non-zero probability of moving between any two states which both have non-zero probability in the distribution being sampled, through some sequence of state transitions.

One generally runs a sampler for some period of time, and then checks for *convergence* of the chain. Intuitively, a chain is said to have converged, if it is generating samples which are 'typical' of the posterior. In practice, one checks for convergence by making trace plots of various parameters or parameter summary statistics to see if they have become 'stable' (*i.e.*, are no longer exploring new space) (see [38] for a nice practical discussion of sampling in the context of Bayesian modeling). Although the theory of sampling essentially says that convergence should occur from any starting state, starting several parallel chains can help aid detection of convergence by allowing the user to potentially see different local minima in different chains. However, deducing convergence is an imperfect art, and is very difficult for any problems which actually require sampling. A rule of thumb is to run a chain for no less than twice as long as it takes to converge, and then to discard the *burn in* samples (pre-convergence samples), while retaining the others for the task at hand.

In Chapter 5 we rely on two MCMC methods for training our HB-CPM – Gibbs Sampling, and Slice Sampling.

Gibbs Sampling

Gibbs sampling is one of the simplest sampling algorithms, and is widely used in practice. It requires that one can sample from each hidden variable/parameter (or group of these), conditioned on the others. One samples individual variables or blocks of variables conditioned on the remaining variables, in an iterative manner, cycling through each block of variables.

For example, suppose we have observed data, x, and a probabilistic model with parameters or hidden variables a, b, c, f, which specify a joint probability distribution that factors as, say,

$$p(a, b, c, f, x) = p(a)p(b)p(c|a, b)p(f|c)p(x|f).$$

Then one might first sample $a \sim p(a|b, c, f, x)$, then $b \sim p(b|a, c, f, x)$, then $c \sim p(c|a, b, f, x)$, and finally $f \sim p(f|a, b, c, x)$, each time conditioning on the latest values available.

For each step of Gibbs sampling, one need only collect factors that contain the variables

that are being sampled for in that step. Suppose we want to sample $c \sim p(c|a, b, f, x)$. Then

$$\begin{aligned} p(c|a, b, f, x) &= \frac{p(a, b, c, f, x)}{p(a, b, f, x)} \\ &= \frac{p(a, b, c, f, x)}{\int_c p(a, b, c, f, x)dc} \\ &= \frac{p(a)p(b)p(c|a, b)p(f|c)p(x|f)}{\int_c p(a)p(b)p(c|a, b)p(f|c)p(x|f)dc} \\ &= \frac{p(f|c)p(c|a, b)}{\int_c p(f|c)p(c|a, b)dc}. \end{aligned}$$

All factors not involving c (the variable we're sampling in this step) cancel out, and we need only collect the others. If $p(f|c)p(c|a, b)$ is proportional to a known distribution, such as when $p(c|a, b)$ is a *conjugate prior* for $p(f|c)$, we will know the normalizing constant, $\int_c p(f|c)p(c|a, b)dc$, and will likely know how to sample exactly from the resulting distribution. If the prior is not conjugate, or the resulting distribution is not amenable to direct sampling, then one can resort to transitions such as Metropolis-Hastings (MH), or slice sampling.

Slice Sampling

Slice sampling [60] is a method which can adapt to the characteristics of the distribution being sampled, thereby making the sampling more efficient. It requires that one can evaluate the density function up to some constant, and also that one provide an estimate of a 'typical slice size'. This latter requirement is somewhat analogous to the requirement of other techniques for specifying a *proposal distribution*, although slice sampling is much less sensitive to the choice of this parameter than are most other methods to the choice of proposal distribution.

We discuss slice sampling only in the context of real-valued, univariate distributions, though it can also be applied to multivariate distributions. The central idea behind slice sampling is that one can sample from a univariate distribution by sampling uniformly from the two-dimensional space that lies under the density function, and then discard the vertical coordinates, keeping only the samples in the space of interest (the horizontal space).

The uniform sampling in this two-dimensional space can be accomplished by alternating between two types of sampling: 1) vertical sampling, in which one samples uniformly from the vertical interval defined from zero on the vertical axis, up to the value of the density at the current state, and 2) horizontal sampling, in which one samples uniformly from the horizontal *slice* which spans the domain of the density function, at the horizontal position given by the

current vertical position from step (1). The vertical step is trivially achieved if we can compute the density function up to some constant, while the latter step is more complicated in general, and usually not sampled exactly, but rather in an indirect way which maintains the properties we require for correct sampling.

One way to perform the horizontal step is to first choose a horizontal interval surrounding the current horizontal position which contains a sizable chunk of the entire slice, and then to draw a sample from this region in an appropriate manner. Finding the interval is typically done with a 'doubling out' or a 'stepping out' procedure, and then sampling from this interval can be done with 'shrinkage'. The more of the true slice we are able to use for this interval, the more efficient will tend to be the overall sampling, since we can move further away in one step. However, if the chosen interval is too large (encompasses too much region lying above the sampling density), then sampling from the slice becomes less efficient. Doubling or stepping out, respectively, doubles the initially chosen interval, or incrementally adds a fixed amount to its size. This is performed until both ends of the interval lie outside the slice. A sample is then chosen from this interval by sampling uniformly until a point in the slice is found. If desired, each time a sample is drawn which is not in the slice, a 'shrinkage' procedure can be applied to reduce the interval, making the next sample more likely to be from within the slice.

Appendix F

Identities Used for HB-CPM Inference

F.1 Sherman-Morrison-Woodbury matrix inversion formula

If you have $\boldsymbol{A}^{-1}$ handy, and need to compute $(\boldsymbol{A}+\boldsymbol{UCV})^{-1}$, then you can do it more efficiently (in some cases), by using:

$$(\boldsymbol{A}+\boldsymbol{UCV})^{-1} = \boldsymbol{A}^{-1} - \boldsymbol{A}^{-1}\boldsymbol{U}(\boldsymbol{C}^{-1}+\boldsymbol{V}\boldsymbol{A}^{-1}\boldsymbol{U})^{-1}\boldsymbol{V}\boldsymbol{A}^{-1}$$

which saves you from having to invert $\boldsymbol{A}$ and then again inverting $(\boldsymbol{A}+\boldsymbol{UCV})$ [30].

F.2 Gaussian mixing property

Here we show[1] how to multiply together two multivariate gaussians together which have the same input, of length D, but different means and covariances:

$$\mathcal{N}(\boldsymbol{y}|\boldsymbol{a},\boldsymbol{A})\mathcal{N}(\boldsymbol{y}|\boldsymbol{b},\boldsymbol{B}) = z_c\mathcal{N}(\boldsymbol{y}|\boldsymbol{c},\boldsymbol{C})$$

where $\boldsymbol{C} = (\boldsymbol{A}^{-1}+\boldsymbol{B}^{-1})^{-1}$ and $c = \boldsymbol{C}[\boldsymbol{A^{-1}a}+\boldsymbol{B^{-1}b}]$. The normalization constant is given by

$$z_c = (2\pi)^{-\frac{D}{2}}|\boldsymbol{A}|^{-\frac{1}{2}}|\boldsymbol{B}|^{-\frac{1}{2}}|\boldsymbol{C}|^{\frac{1}{2}}\exp\left[-\frac{1}{2}\left(\boldsymbol{a}^T\boldsymbol{A}^{-1}\boldsymbol{a}+\boldsymbol{b}^T\boldsymbol{B}^{-1}\boldsymbol{b}-\boldsymbol{c}^T\boldsymbol{C}^{-1}\boldsymbol{c}\right)\right],$$

[1] The first part is taken from Sam Roweis' Gaussian Identities notes.

We can complete the square in C to obtain a gaussian in a (or, because of symmetry, b). We do so here for the scalar version:

$$\begin{aligned}
&(aA^{-1}a + bB^{-1}b - cC^{-1}c) \\
&= \frac{a^2}{A} + \frac{b^2}{B} - \frac{c^2}{C} \\
&= \frac{a^2}{A} + \frac{b^2}{B} - \frac{Ca^2}{A^2} - 2\,\frac{Cab}{AB} - \frac{Cb^2}{B^2} \\
&= \frac{a^2}{A} + \frac{b^2}{B} - \frac{a^2}{(A^{-1}+B^{-1})\,A^2} - 2\,\frac{ab}{(A^{-1}+B^{-1})\,AB} - \frac{b^2}{(A^{-1}+B^{-1})\,B^2} \\
&= \frac{a^2 + b^2 - 2\,ab}{B + A} \\
&= \frac{(a-b)^2}{B+A}
\end{aligned}$$

Thus z_c can be re-written as:

$$\begin{aligned}
z_c &= (2\pi)^{-1} A^{-\frac{1}{2}} B^{-\frac{1}{2}} C^{\frac{1}{2}} \exp\left[-\frac{1}{2}\left(aA^{-1}a + bB^{-1}b - cC^{-1}c\right)\right] \\
&= (2\pi)^{-1}\left(\frac{C}{AB}\right)^{\frac{1}{2}} \exp\left[-\frac{1}{2}\left(\frac{(a-b)^2}{B+A}\right)\right] \\
&= (2\pi)^{-1}\left(\frac{C}{AB}\right)^{\frac{1}{2}} \left(\sqrt{2\pi(B+A)}\right) \mathcal{N}(a|b,(B+A)) \\
&= \left(\frac{C(B+A)}{AB}\right)^{\frac{1}{2}} \mathcal{N}(a|b,(B+A)) \\
&= \mathcal{N}(a|b,(B+A))
\end{aligned}$$

F.3 Breaking apart a compound gaussian

$$\mathcal{N}(x|(z+d)u,S) = \frac{\sqrt{\pi S}}{u}\mathcal{N}\left(d|\frac{x-uz}{u},\frac{S}{u^2}\right)\mathcal{N}\left(x|uz,\frac{S}{2}\right)$$

which can be seen by completing the square in d and then in x:

$$
\begin{aligned}
&= \frac{(x-(z+d)\,u)^2}{S} \\
&= \frac{x^2}{S} - 2\,\frac{xuz}{S} - 2\,\frac{xud}{S} + \frac{u^2z^2}{S} + 2\,\frac{u^2zd}{S} + \frac{u^2d^2}{S} \\
&= \frac{u^2d^2}{S} + \left(-2\,\frac{xu}{S} + 2\,\frac{u^2z}{S}\right) d + \frac{x^2}{S} - 2\,\frac{xuz}{S} + \frac{u^2z^2}{S} \\
&= \frac{u^2}{S}\left(d - \frac{x-uz}{u}\right)^2 - \frac{x^2 - 2\,xuz + u^2z^2}{S} + \frac{x^2}{S} - 2\,\frac{xuz}{S} + \frac{u^2z^2}{S} \\
&= \frac{u^2}{S}\left(d - \frac{x-uz}{u}\right)^2 + \left(2\,\frac{x^2}{S} - 4\,\frac{xuz}{S} + 2\,\frac{u^2z^2}{S}\right) \\
&= \frac{u^2}{S}\left(d - \frac{x-uz}{u}\right)^2 + 2\,\frac{(x-uz)^2}{S}
\end{aligned}
$$

And then

$$
\begin{aligned}
\mathcal{N}(x|(z+d)u, S) &= \frac{1}{\sqrt{2\pi S}} \exp\left[-\frac{1}{2}\left(\frac{(x-(z+d)\,u)^2}{S}\right)\right] \\
&= \frac{1}{\sqrt{2\pi S}} \exp\left[-\frac{1}{2}\left(\frac{u^2}{S}\left(d - \frac{x-uz}{u}\right)^2\right)\right] \exp\left[-\frac{1}{2}\left(2\,\frac{(x-uz)^2}{S}\right)\right] \\
&= \frac{1}{\sqrt{2\pi S}} \sqrt{\frac{2\pi S}{u^2}} \sqrt{\frac{2\pi S}{2}} \mathcal{N}\left(d|\frac{x-uz}{u}, \frac{S}{u^2}\right) \mathcal{N}\left(x|uz, \frac{S}{2}\right) \\
&= \left(\sqrt{\frac{\pi S}{u^2}}\right) \mathcal{N}\left(d|\frac{x-uz}{u}, \frac{S}{u^2}\right) \mathcal{N}\left(x|uz, \frac{S}{2}\right).
\end{aligned}
$$

F.4 Multivariate gaussian conditionals

Given a two-dimensional gaussian[2] ,

$$
\mathcal{N}\left(\begin{bmatrix} x \\ y \end{bmatrix} | \begin{bmatrix} a \\ b \end{bmatrix}, \begin{bmatrix} A & C \\ C & B \end{bmatrix}\right),
$$

the conditional of one variable on the other is given by

$$
p(x|y) = \mathcal{N}\left(x|a + \frac{C}{B}(y-b), A - \frac{C^2}{B}\right).
$$

[2]From Sam Roweis' Gaussian Identities which contains the more general form.

If you have only the precision matrix, for a 2D Gaussian,

$$\mathcal{N}\left(\left[\begin{array}{c} x \\ y \end{array}\right] | \left[\begin{array}{c} a \\ b \end{array}\right], \left[\begin{array}{cc} A & C \\ C & B \end{array}\right]^{-1}\right),$$

then taking note of the fact that

$$\text{Inverse}\left(\left[\begin{array}{cc} A & C \\ C & B \end{array}\right]\right) = \frac{1}{AB - C^2}\left[\begin{array}{cc} B & -C \\ -C & A \end{array}\right]$$

then you can avoid computing the inverse, and use the following

$$p(x|y) = \mathcal{N}\left(x|a - \frac{C}{A}(y-b), \frac{B}{AB-C^2} - \frac{C^2}{A(AB-C^2)}\right).$$

F.5 Matrix version of completing the square

Using the scalar analogy,
$ax^2 - 2bx + c = (\sqrt{(}a)x - \frac{b}{\sqrt{a}})^2 - \frac{b^2}{a} + c = (\sqrt{a}(x - \frac{b}{a}))^2 - \frac{b^2}{a} + c = a(x - \frac{b}{a})^2 - \frac{b^2}{a} + c$,
it is reasonably straightforward to see that

$$\boldsymbol{x}^T\boldsymbol{A}\boldsymbol{x} - 2\boldsymbol{x}^T\boldsymbol{b} + c = (\boldsymbol{x} - \boldsymbol{A}^{-1}\boldsymbol{b})^T\boldsymbol{A}(\boldsymbol{x} - \boldsymbol{A}^{-1}\boldsymbol{b}) - \boldsymbol{b}^T\boldsymbol{A}^{-1}\boldsymbol{b} + c,$$

provided that A (and hence its inverse) is symmetric.

F.6 Unrolling unnormalized gaussian clique potentials

Suppose you have a product of clique potentials , each with potential
$\phi(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu_i})^T\boldsymbol{S_i}(\boldsymbol{x} - \boldsymbol{\mu_i})\right)$, which together form an energy function,

$$E(\boldsymbol{x}) = \prod_i \phi(\boldsymbol{x}) = \exp\left(-\frac{1}{2}\sum_i (\boldsymbol{x} - \boldsymbol{\mu_i})^T\boldsymbol{S_i}(\boldsymbol{x} - \boldsymbol{\mu_i})\right),$$

then one can write this in the following form[3] which may be more convenient,

$$E(\boldsymbol{x}) = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{\mu})^T\boldsymbol{S}(\boldsymbol{x} - \boldsymbol{\mu})\right) C$$

[3]Thanks to David Ross for first pointing this out to me.

where $\mathbf{S} = \sum_{\mathbf{i}} \mathbf{S_i}$, $\mu = \mathbf{S^{-1}}(\sum_{\mathbf{i}} \mathbf{S_i}\boldsymbol{\mu_i})$ and C is a constant which does not depend on $\boldsymbol{x}$, shown below to have the value $C = \exp\left(-\frac{1}{2}\left[\left(\sum_i \boldsymbol{\mu_i^T S_i \mu_i}\right) - \boldsymbol{\mu}^T S \boldsymbol{\mu}\right]\right)$. This is accomplished by expanding each sub-component and then 'completing the square' (Identity F.5), as follows

$$
\begin{aligned}
E(\boldsymbol{x}) &= \exp\left(-\frac{1}{2}\left[\sum_i \left(\boldsymbol{x}^T \boldsymbol{S_i x} - 2\boldsymbol{x}^T \boldsymbol{S_i \mu_i} + \boldsymbol{\mu_i^T S_i \mu_i}\right)\right]\right) \\
&= \exp\left(-\frac{1}{2}\left[\boldsymbol{x}^T \left(\sum_i \boldsymbol{S_i}\right)\boldsymbol{x} - 2\boldsymbol{x}^T\left(\sum_i \boldsymbol{S_i \mu_i}\right) + \left(\sum_i \boldsymbol{\mu_i^T S_i \mu_i}\right)\right]\right) \\
&= \exp\left(-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^T \boldsymbol{S}(\boldsymbol{x}-\boldsymbol{\mu})\right) \exp\left(-\frac{1}{2}\left[\left(\sum_i \boldsymbol{\mu_i^T S_i \mu_i}\right) - \left(\sum_i \boldsymbol{S_i \mu_i}\right)^T \boldsymbol{S}^{-1}\left(\sum_i \boldsymbol{S_i \mu_i}\right)\right]\right) \\
&= \exp\left(-\frac{1}{2}\left(\boldsymbol{x}-\boldsymbol{\mu}\right)^T \boldsymbol{S}(\boldsymbol{x}-\boldsymbol{\mu})\right) \exp\left(-\frac{1}{2}\left[\left(\sum_i \boldsymbol{\mu_i^T S_i \mu_i}\right) - \boldsymbol{\mu}^T \boldsymbol{S \mu}\right]\right).
\end{aligned}
$$

Bibliography

[1] Petricoin E 3rd and Liotta LA. Counterpoint: The vision for a new diagnostic paradigm. *Clin Chem*, 49:1276–1278, 2003.

[2] Yates JR 3rd. Mass spectral analysis in proteomics. *Annu Rev Biophys Biomol Struct*, 33:297–316, 2004.

[3] Ruedi Aebersold and Matthias Mann. Mass spectrometry-based proteomics. *Nature*, 422:198–207, 2003.

[4] Bruce Alberts, Alexander Johnson, Julian Lewis, Martin Raff, Keith Roberts, and Peter Walter. *Molecular Biology of the Cell*. Garland Science, 2002.

[5] A.H. America, J.H. Cordewener, M.H. van Geffen, A. Lommen, J.P. Vissers, R.J. Bino, and R.D. Hall. Alignment and statistical difference analysis of complex peptide data sets generated by multidimensional LC-MS. *Proteomics*, 2:641–53, 2006.

[6] Markus Anderle, Sushmita Roy, Hua Lin, Christopher Becker, and Keith Joho. Quantifying reproducibility for differential proteomics: noise analysis for protein liquid chromatography-mass spectrometry of human serum. *Bioinformatics Advance Access*, doi:10.1093, 2004.

[7] Thomas M. Annesley. Ion suppression in mass spectrometry. *Clinical Chemistry*, 49:1041–1044, 2003.

[8] Yeast Resource Center at the University of Washington. Introduction to mass spectrometry. Retrieved from web on November 3nd 2004 from `http://depts.washington.edu/yeastrc/ms_home.htm`.

[9] Ziv Bar-Joseph, Georg Gerber, David K. Gifford, Tommi Jaakkola, and Itamar Simon. A new approach to analyzing gene expression time series data. In *RECOMB*, pages 39–48, 2002.

[10] E. Bauer, D. Koller, and Y. Singer. Batch and on-line parameter estimation in Bayesian networks. 1997.

[11] Yoshua Bengio and Paolo Frasconi. An input output HMM architecture. In G. Tesauro, D. Touretzky, and T. Leen, editors, *Advances in Neural Information Processing Systems*, volume 7, pages 427–434. The MIT Press, 1995.

[12] J. Binder, K. Murphy, and S. Russell. Space-efficient inference in dynamic probabilistic networks. 1997.

[13] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

[14] H.F. Boelens, R.J. Dijkstra, P.H. Eilers, F. Fitzpatrick, and J.A. Westerhuis. New background correction method for liquid chromatography with diode array detection, infrared spectroscopic detection and raman spectroscopic detection. *Journal of Chromatography A*, 1057:21–30, 2004.

[15] D. Bylund and R. Danielssonand G Malmquistand K.E. Markides. Chromatographic alignment by warping and dynamic programming as a pre-processing tool for parafac modelling of liquid chromatography-mass spectrometry data. *J Chromatogr A*, 961:237–244, 2002.

[16] University of Newbraska Center for Biotechnology. Mass spectrometry for biologists. Retrieved from web on November 17th 2004 from `http://www.biotech.unl.edu/oldroot/MassSpec/msprimer.html`.

[17] Philip K. Chan and Matthew V. Mahoney. Modeling multiple time series for anomaly detection. In *ICDM*, 2005.

[18] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Proceedings of the Royal Statistical Society*, pages 1–38, 1976.

[19] Eleftherios P. Diamandis. Correspondence: Re: Serum proteomic patterns for detection of prostate cancer. *Journal of the National Cancer Institute,*, 95:489–491, 2003.

[20] EP Diamandis. Comment on: Use of proteomic patterns in serum to identify ovarian cancer. *The Lancet*, 360:170, 2002.

[21] EP Diamandis. Point: Proteomic patterns in biological fluids: do they represent the future of cancer diagnostics? *Clin Chem*, 49:1272–1275, 2003.

[22] EP Diamandis. Mass spectrometry as a diagnostic and a cancer biomarker discovery tool: Opportunities and potential limitations. *Mol Cell Proteomics*, 3:367–378, 2004.

[23] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge Univ. Press, 2000. Durbin.

[24] Petricoin EF and Liotta LA. SELDI-TOF-based serum proteomic pattern diagnostics for early detection of cancer. *Curr Opin Biotechnol*, 15:24–30, 2004.

[25] B. Efron, R. Tibshirani, J.D. Storey, and V. Tusher. Empirical bayes analysis of a microarray experiment. *Journal of the American Statistical Association*, 96:1151–1160, 2001.

[26] B. Ferrell and S. Santuro. NASA shuttle valve data. `http://www.cs.fit.edu/~pkc/nasa/data/`, 2005.

[27] Bernd Fischer, Jonas Grossmann, Volker Roth, Wilhelm Gruissem, Sacha Baginsky, and Joachim M. Buhmann. Semi-supervised LC/MS alignment for differential proteomics. In *ISMB (Supplement of Bioinformatics)*, pages 132–140, 2006.

[28] Scott J. Gaffney and Padhraic Smyth. Joint probabilistic curve clustering and alignment. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*, pages 473–480. MIT Press, Cambridge, MA, 2005.

[29] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald B. Rubin. *Bayesian Data Analysis, Second Edition*. Chapman & Hall/CRC, July 2003.

[30] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. Johns Hopkins University Press, Baltimore, MD, USA, 1996.

[31] Andrew Guzzetta. Reverse phase HPLC basics for LC/MS. Retrieved from web on November 3rd 2004 from `http://www.ionsource.com/tutorial/chromatography/rphplc.htm`.

[32] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer, 2001.

[33] Ramsay J.O. and Li X. Curve registration. *Journal of the Royal Statistical Society: Series B (Methodological)*, 60:351–363, 1998.

[34] Nebojsa Jojic and Brendan J. Frey. A comparison of algorithms for inference and learning in probabilistic graphical models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(9):1392–1416, 2005.

[35] Michael I. Jordan. An introduction to probabilistic graphical models. Forthcoming.

[36] Baggerly KA, Morris JS, Wang J, Gold D, Xiao LC, and Coombes KR. A comprehensive approach to the analysis of matrix-assisted laser desorption/ionization-time of flight proteomics spectra from serum samples. *Proteomics*, 3:1667–1672, 2003.

[37] Mohammed Waleed Kadous. *Temporal Classification: Extending the Classification Paradigm to Multivariate Time Series*. PhD thesis, School of Computer Science and Engineering, University of New South Wales, 2002.

[38] Robert E. Kass, Bradley P. Carlin, Andrew Gelman, and Radford M. Neal. Markov chain Monte Carlo in practice: A roundtable discussion. *The American Statistician*, 52(2):93–100, 1998.

[39] P. Kearney and P. Thibault. Bioinformatics meets proteomics - bridging the gap between mass spectrometry data analysis and cell biology. *Journal of Bioinformatics and Computational Biology*, 1:183–200, 2003.

[40] T. Kislinger and A. Emili. Going global: protein expression profiling using shotgun mass spectrometry. *Curr Opin Mol Ther.*, 5:285–293, 2003.

[41] Cartegni L, Chew SL, and Krainer AR. Listening to silence and understanding nonsense: exonic mutations that affect splicing. *Nature Reviews Genetics*, 3:285–298, 2002.

[42] L. Latecki, V. Megalooikonomou, Q. Wang, R. Lakaemper, C. Ratanamahatana, and E. Keogh. Elastic partial matching of time series, 2005.

[43] J. Lember and A. Koloydenko. Adjusted viterbi training, 2004.

[44] Benjamin Lewin. *Genes VIII*. Prentice Hall, 2004.

[45] Jennifer Listgarten and Andrew Emili. Practical proteomic biomarker discovery: taking a step back to leap forward. *Drug Discovery Today*, 10:1697–1702, 2005.

[46] Jennifer Listgarten and Andrew Emili. Statistical and computational methods for comparative proteomic profiling using liquid chromatography-tandem mass spectrometry. *Molecular and Cellular Proteomics*, 4:419–434, 2005.

[47] Jennifer Listgarten, Kathryn Graham, Sambasivarao Damaraju, Carol Cass, John Mackey, and Brent Zanke. Clinically validated benchmarking of normalisation techniques for two-colour oligonucleotide spotted microarray slides. *Applied Bioinformatics*, 2:219–228, 2003.

[48] Jennifer Listgarten, Radford M. Neal, Sam T. Roweis, and Andrew Emili. Multiple alignment of continuous time series. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems 17*. MIT Press, Cambridge, MA, 2005.

[49] Jennifer Listgarten, Radford M. Neal, Sam T. Roweis, Rachel Puckrin, and Sean Cutler. Bayesian detection of infrequent differences in sets of time series with shared structure. In B. Schölkopf, J.C. Platt, and T. Hofmann, editors, *Advances in Neural Information Processing Systems 19*. MIT Press, Cambridge, MA, 2007.

[50] Jennifer Listgarten, Radford M. Neal, Sam T. Roweis, Peter Wong, and Andrew Emili. Difference detection in LC-MS data for protein biomarker discovery. In *Bioinformatics (to appear)*, 2006.

[51] Proteometrics LLC. Tutorial on m/z. Retrieved from web on November 25th 2004 from `http://bioinformatics.genomicsolutions.com/moverz/tutorials/pages/peak.html`.

[52] I. Lonnstedt and T. Speed. Replicated microarray data. *Statistica Sinica*, 12:31–46, 2002.

[53] James Lyons-Weiler. Standards of excellence and open questions in cancer biomarker research: An informatics perspective. *Cancer Informatics*, 1:1–7, 2005.

[54] D. J. C. MacKay. Introduction to Monte Carlo methods. In M. I. Jordan, editor, *Learning in Graphical Models*, NATO Science Series, pages 175–204. Kluwer Academic Press, 1998.

[55] X. L. Meng and D. van Dyk. The EM algorithm — an old folk song sung to a fast new tune (with Discussion). *J. Royal Stat. Soc. B*, 59:511–567, 1997.

[56] Mukund Narasimhan, Paul Viola, and Michael Shilman. Online decoding of markov models under latency constraints. In *ICML '06: Proceedings of the 23rd international conference on Machine learning*, pages 657–664, New York, NY, USA, 2006. ACM Press.

[57] R. Neal and G. Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. In M. I. Jordan, editor, *Learning in Graphical Models*. Kluwer, 1998.

[58] R. M. Neal and J. Zhang. Classification with Bayesian neural networks and Dirichlet diffusion trees. In I. Guyon et al, editor, *Feature Extraction, Foundations and Applications*. Physica-Verlag, Springer, 2006.

[59] Radford M. Neal. Probabilistic inference using Markov chain Monte Carlo methods. Technical Report CRG-TR-93-1, University of Toronto, 1993.

[60] Radford M. Neal. Slice sampling. *Annals of Statistics*, 31:705–767, 2003.

[61] N.-P. V. Nielsen, J. M. Carstensen, and J. Smedsgaard. Aligning of single and multiple wavelength chromatographic profiles for chemometric data analysis using correlation optimised warping. *J Chromatogr A*, 805:17–35, 1998.

[62] Emanuel F. Petricoin, Ali M. Ardekani, Ben A. Hitt, Peter J. Levine, Vincent A. Fusaro, Seth M. Steinberg, Gordon B. Mills, Charles Simone, David A. Fishman, Elise C. Kohn, and Lance A. Liotta. Use of proteomic patterns in serum to identify ovarian cancer. *The Lancet*, 359:572–577, 2002.

[63] Emanuel F. Petricoin and Lance A. Liotta. Mass spectrometry-based diagnostics: The upcoming revolution in disease detection. *Clinical Chemistry*, 49:533–534, 2003.

[64] P.A. Pevzner. *Computational Molecular Biology: An Algorithmic Approach*. The MIT Press, 2000.

[65] Alan B. Poritz. Hidden markov models: A guided tour. In *Proceedings of the IEEE Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7–13. Morgan Kaufmann, 1988.

[66] A. Prakash, P. Mallick, J. Whiteaker, H. Zhang, A. Paulovich, M. Flory, H. Lee, R Aebersold, and B. Schwikowski. Signal maps for mass spectrometry-based comparative proteomics. *Molecular and Cellular Proteomics*, doi:10.1074/mcp.M500133-MCP200, 2006.

[67] John T. Prince and Edward M. Marcotte. Chromatographic alignment of ESI-LC-MS proteomics datasets by ordered bijective interpolated warping. *Analytical Chemistry*, 78:6140–6152, 2006.

[68] Dragan Radulovic, Salomeh Jelveh, Soyoung Ryu, T. Guy Hamilton, Eric Foss, Yongyi Mao, and Andrew Emili. Informatics platform for global proteomic profiling and biomarker discovery using liquid-chromatography-tandem mass spectrometry. *Mol Cell Proteomics*, 10:984–997, 2004.

[69] Lilien RH, Farid H, and Donald BR. Probabilistic disease classification of expression-dependent proteomic data from mass spectrometry of human serum. *J Comput Biol*, 10:925–946, 2003.

[70] H. Sakoe and S.Chiba. Dynamic programming algorithm for spoken word recognition. *Readings in Speech Recognition*, pages 159–165, 1990.

[71] Ruslan Salakhutdinov and Sam T. Roweis. Adaptive overrelaxed bound optimization methods. In *ICML*, pages 664–671, 2003.

[72] Ruslan Salakhutdinov, Sam T. Roweis, and Zoubin Ghahramani. Optimization with EM and Expectation-Conjugate-Gradient. In *International Conference on Machine Learning*, pages 672–679, 2003.

[73] Glen A. Satten, Somnath Datta, Hercules Moura, Adrian R. Woolfitt, Maria da G. Carvalho, George M. Carlone, Barun K. De, Antonis Pavlopoulos, and John R. Barr. Standardization and denoising algorithms for mass spectra to classify whole-organism bacterial specimens. *Bioinformatics Advance Access*, doi:10.1093, 2004.

[74] Anne Sauve and Terence Speed. Normalization, baseline correction and alignment of high-throughput mass spectrometry data. In *Gensips*, 2004.

[75] Jeffrey C. Silva, Richard Denny, Craig A. Dorschel, Marc Gorenstein, Ignatius J. Kass, Guo-Zhong Li, Therese McKenna, Michael J. Nold, Keith Richardson, Phillip Young, and Scott Geromanos. Quantitative proteomic analysis by accurate mass retention time pairs. *Analytical Chemistry*, 77:2187–2200, 2005.

[76] Gordon K. Smyth, Yee Hwa Yang, and Terry Speed. Statistical issues in cDNA microarray data analysis. In M J. Brownstein and A. B. Khodursky, editors, *Functional Genomics: Methods and Protocols*. Humana Press, 2001.

[77] J.M. Sorace and M. Zhan. A data review and re-assessment of ovarian cancer serum proteomic profiling. *BMC Bioinformatics*, 4, 2003.

[78] J.D. Storey and R. Tibshirani. Statistical significance for genome-wide studies. *Proceedings of the National Academy of Sciences*, 100:9440–9445, 2003.

[79] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, and Gilbert Chu. Diagnosis of multiple cancer types by shrunken centroids of gene expression. *PNAS*, 99:6567–6572, 2002.

[80] Robert Tibshirani, Trevor Hastie, Balasubramanian Narasimhan, Scott Soltys, Gongyi Shi, Albert Koong, and Quynh-Thu Le. Sample classification from protein mass spectrometry, by peak probability contrasts. *Bioinformatics Advance Access*, doi:10.1093, 2004.

[81] Giorgio Tomasi, Thomas Skov, and Frans van den Berg. Correlation optimized warping and dynamic time warping as preprocessing methods for chromatographic data. *Journal of Chemometrics*, 18:231–241, 2004.

[82] V. Tusher, R. Tibshirani, and G. Chu. Significance analysis of microarrays applied to the ionizing radiation response. *Proceedings of the National Academy of Sciences*, 98:5116–5124, 2001.

[83] Mike Tyers and Matthias Mann. From genomics to proteomics. *Nature*, 422:193–197, 2003.

[84] Wang W, Zhou H, Lin H, Roy S, Shaler TA, Hill LR, Norton S, Kumar P, Anderle M, and Becker CH. Quantification of proteins and metabolites by mass spectrometry without isotopic labeling or spiked standards. *Analytical Chemistry*, 75:4818–4826, 2003.

[85] Michael Wagner, Dayanand Naik, and Alex Pothen. Protocols for disease classification from mass spectrometry data. *Proteomics*, 3:1692–1698, 2003.

[86] Matthew C. Wiener, Jeffrey R. Sachs, Ekaterina G. Deyanova, and Nathan A. Yates. Differential mass spectrometry: A label-free LC-MS method for finding significant differences in complex peptide and protein mixtures. *Analytical Chemistry*, 76:6085–6096, 2004.

[87] Qu Y, Adam BL, Yasui Y, Ward MD, Cazares LH, Schellhammer PF, Feng Z, Semmes OJ, and Wright GL Jr. Boosted decision tree analysis of surface-enhanced laser desorption/ionization mass spectral serum profiles discriminates prostate cancer from noncancer patients. *Clinical Chemistry*, 48:1835–1843, 2002.

[88] Yasui Y, Pepe M, Thompson ML, Adam BL, Wright GL Jr, Qu Y, Potter JD, Winget M, Thornquist M, and Feng Z. A data-analytic strategy for protein biomarker discovery: profiling of high-dimensional proteomic data for cancer detection. *Biostatistics*, 4:449–463, 2003.