A Bayesian approach to initial model inference in cryo-electron microscopy

Dissertation

zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades

"Doctor rerum naturalium"

der Georg-August-Universität Göttingen

im Promotionsprogramm PCS

der Georg-August University School of Science (GAUSS)

vorgelegt von

Paul Joubert

aus Kapstadt, Südafrika

Göttingen, 2016

Referent: Dr. Michael Habeck
Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Axel Munk
Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission:

Prof. Dr. Timo Aspelmeier
Georg-August-Universität Göttingen

Prof. Dr. Winfried Kurth
Georg-August-Universität Göttingen

Prof. Dr. Holger Stark
Georg-August-Universität Göttingen

Prof. Dr. Stephan Waack
Georg-August-Universität Göttingen

Tag der Mündlichen Prüfung: 4.3.2016

Abstract

Single-particle cryo-electron microscopy (cryo-EM) is widely used to study the structure of macromolecular assemblies. Tens of thousands of noisy two-dimensional images of the macromolecular assembly viewed from different directions are used to infer its three-dimensional structure. The first step is to estimate a low-resolution initial model and initial image orientations. This is a challenging ill-posed inverse problem with many unknowns, including an unknown orientation for each two-dimensional image. Obtaining a good initial model is crucial for the success of the subsequent refinement step. In this thesis we introduce new algorithms for estimating an initial model in cryo-EM, based on a coarse representation of the electron density. The contribution of the thesis can be divided into these two parts: one relating to the model, and the other to the algorithms. The first main contribution of the thesis is using Gaussian mixture models to represent electron densities in reconstruction algorithms. We use spherical (isotropic) mixture components with unknown positions, size and weights. We show that using this representation offers many advantages over the traditional grid-based representation used by other reconstruction algorithms. There is for example a significant reduction in the number of parameters needed to represent the three-dimensional electron density, which leads to fast and robust algorithms. The second main contribution of the thesis is developing Markov Chain Monte Carlo (MCMC) algorithms within a Bayesian framework for estimating the parameters of the mixture models. The first algorithm is a Gibbs sampling algorithm. It is derived by starting with the standard Gibbs sampling algorithm for fitting Gaussian mixture models to point clouds, and extending it to work with images, to handle projections from three dimensions to two dimensions, and to account for unknown rotations and translations. The second algorithm takes a different approach. It modifies the forward model to work with Gaussian noise, and uses sampling algorithms such as Hamiltonian Monte Carlo (HMC) to sample the positions of the mixture components and the image orientations. We provide extensive tests of our algorithms using simulated and experimental data, and compare them to other initial model algorithms.

Zusammenfassung

Eine Hauptanwendung der Einzelpartikel-Analyse in der Kryo-Elektronenmikroskopie ist die Charakterisierung der dreidimensionalen Struktur makromolekularer Komplexe. Dazu werden zehntausende Bilder verwendet, die verrauschte zweidimensionale Projektionen des Partikels zeigen. Im ersten Schritt werden ein niedrig aufgelöstetes Anfangsmodell rekonstruiert sowie die unbekannten Bildorientierungen geschätzt. Dies ist ein schwieriges inverses Problem mit vielen Unbekannten, einschließlich einer unbekannten Orientierung für jedes Projektionsbild. Ein gutes Anfangsmodell ist entscheidend für den Erfolg des anschließenden Verfeinerungsschrittes. Meine Dissertation stellt zwei neue Algorithmen zur Rekonstruktion eines Anfangsmodells in der Kryo-Elektronenmikroskopie vor, welche auf einer groben Darstellung der Elektronendichte basieren. Die beiden wesentlichen Beiträge meiner Arbeit sind zum einen das Modell, welches die Elektronendichte darstellt, und zum anderen die neuen Rekonstruktionsalgorithmen. Der erste Hauptbeitrag liegt in der Verwendung Gaußscher Mischverteilungen zur Darstellung von Elektrondichten im Rekonstruktionsschritt. Ich verwende kugelförmige Mischungskomponenten mit unbekannten Positionen, Ausdehnungen und Gewichtungen. Diese Darstellung hat viele Vorteile im Vergleich zu einer gitterbasierten Elektronendichte, die andere Rekonstruktionsalgorithmen üblicherweise verwenden. Zum Beispiel benötigt sie wesentlich weniger Parameter, was zu schnelleren und robusteren Algorithmen führt. Der zweite Hauptbeitrag ist die Entwicklung von Markovketten-Monte-Carlo-Verfahren im Rahmen eines Bayes'schen Ansatzes zur Schätzung der Modellparameter. Der erste Algorithmus kann aus dem Gibbs-Sampling, welches Gaußsche Mischverteilungen an Punktwolken anpasst, abgeleitet werden. Dieser Algorithmus wird hier so erweitert, dass er auch mit Bildern, Projektionen sowie unbekannten Drehungen und Verschiebungen funktioniert. Der zweite Algorithmus wählt einen anderen Zugang. Das Vorwärtsmodell nimmt nun Gaußsche Fehler an. Sampling-Algorithmen wie Hamiltonian Monte Carlo (HMC) erlauben es, die Positionen der Mischungskomponenten und die Bildorientierungen zu schätzen. Meine Dissertation zeigt umfassende numerische Experimente mit simulierten und echten Daten, die die vorgestellten Algorithmen in der Praxis testen und mit anderen Rekonstruktionsverfahren vergleichen.

Acknowledgements

I'm grateful to have had the opportunity to spend the first part of my PhD in Tübingen. The machine learning department at the MPI is a very stimulating environment. I would like to thank everyone there, especially the other PhD students, for the many interesting talks and discussions.

For the second part of my PhD I moved to Göttingen. Thank you to everyone at the IMS for making me feel welcome there, and for being friendly and supportive.

I am very grateful to my main supervisor, Michael Habeck. I consider myself lucky to have had a very knowledgeable supervisor who generously gave his time and attention to discussing my work, made useful comments and asked good questions, and explained new concepts clearly. I learned a lot from his attention to detail, and his persistence in understanding the results of an experiment.

Thank you to my second supervisor, Axel Munk, for his advice and support. I would also like to thank the other members of the examination committee, Holger Stark, Timo Aspelmeier, Winfried Kurth and Stephan Waack, for agreeing to be on my committee.

Thank you very much to my girlfriend Tanja for her cheerful support and enthusiasm. Thank you also to my friends and family, especially my brother Jan, for their support and encouragement. Lastly, I am deeply indebted to my parents, for their love, and everything they taught me.

Contents

Chapter 1

Introduction

1.1 Cryo-electron microscopy

Many of the important processes taking place in cells are carried out by macromolecular assemblies. To understand how these large biological molecules perform their functions, it is important to know what they look like. The three-dimensional structure of macromolecular assemblies can be determined by structural biologists using various experimental techniques.

A relatively new technique for inferring the structure of macromolecular assemblies is cryo-electron microscopy (cryo-EM). In contrast to X-ray crystallography, it does not require the macromolecular assembly to be crystallised, thereby allowing it to retain its native structure. Compared to nuclear magnetic resonance (NMR), it can be used to determine the structure of much larger macromolecular assemblies.

To apply cryo-EM to a macromolecular assembly, many copies of the macromolecular assembly are isolated in a water solution, and spread over a two-dimensional carbon grid. The grid is rapidly frozen using liquid ethane to prevent the water from crystallising. This ensures that the native structure of the macromolecular assembly particles is preserved.

The ice layer with the frozen particles is imaged using a transmission electron microscope (TEM). The images contain two-dimensional projections of the three-dimensional particles: the intensity of each pixel is proportional to the integral of the electron density of the particle along the imaging direction (Fig. 1.1).

The electron microscope uses a very low electron dose to limit damage to the particles caused by the electron beam. As a result, the particle images have a very low signal-to-noise (SNR) ratio. To compensate for the low SNR, a large number of particles are imaged (around 10^4 to 10^5).

The particles are oriented randomly in the ice layer. This means that every particle image is the projection of a different particle along a random, unknown direction. An additional complication is that there could be variations in the structure of the particles. In this thesis we will ignore the variations.

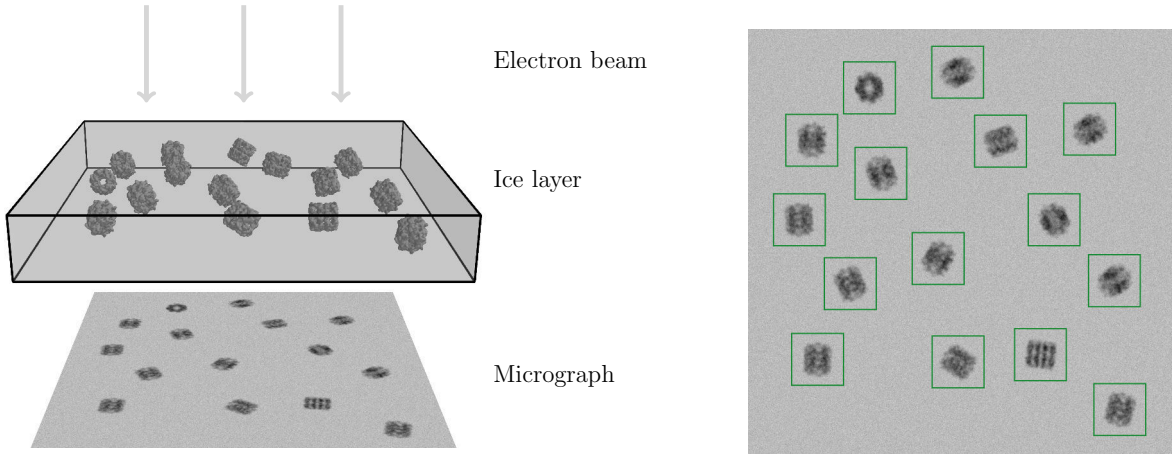

Figure 1.1: (*Left*) Multiple particles in random orientations are imaged using an electron microscope to produce an image called a micrograph. (*Right*) Individual particle images in the micrograph are identified during the particle picking step (*green squares*).

The challenge in cryo-EM is to infer the three-dimensional electron density from its two-dimensional projections. What makes this hard is that the orientations of the particles are unknown. The resulting electron density map typically has a resolution in the range of 3 (near-atomic) to 25 Å. By combining cryo-EM density maps with other sources of information, such as the atomic structure of protein subunits, an atomic model of the macromolecular assembly can sometimes be inferred.

The above procedure to infer the structure of a single macromolecular assembly is also known as single-particle cryo-EM. In contrast, electron tomography is another cryo-EM technique which is used to study larger structures, such as an entire cell. We will not be concerned with such alternative cryo-EM techniques, and will therefore refer to single-particle cryo-EM simply as cryo-EM.

1.2 Cryo-EM data processing pipeline

The data processing pipeline consists of several steps to reconstruct a high-resolution electron density from the images recorded by the electron microscope. Each microscope image, known as a micrograph, contains the projected images of many particles. Locating the particle images in each micrograph is known as particle picking, which is the first step in the data processing pipeline (Fig. 1.1).

The next step is to cluster the particle images into groups. To each particle image corresponds a specific three-dimensional orientation of the particle, which can be parametrised by the projection direction (two parameters) and an in-plane rotation (one parameter). Particles with similar projection directions are grouped together, and aligned relative to each other by in-plane

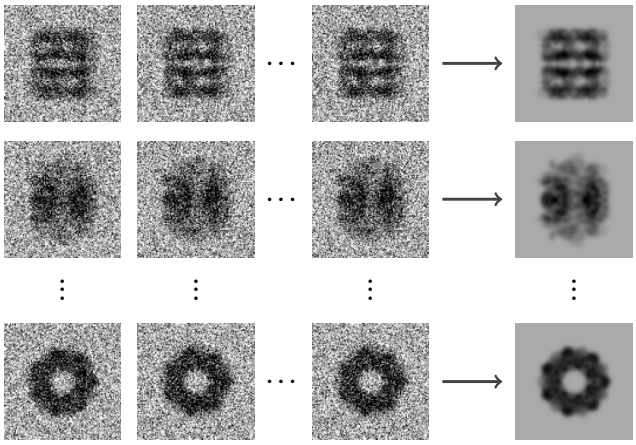

Figure 1.2: Individual particle images are clustered, aligned and averaged to obtain class averages. They have a higher signal-to-noise ratio, and reduce the number of orientation parameters.

rotations and translations. The aligned particles in each group, or class, are averaged together to produce a single class average (Fig. 1.2). This class averaging step summarises the original set of around 10^4 particle images by a much smaller set of around 10 to 100 class averages with a much higher SNR. In addition to reducing the noise level, class averaging also reduces the number of unknown particle orientation parameters.

The next step is to use the class averages to infer an initial, low-resolution model of the electron density (Fig. 1.3).

In the final step, the initial model is refined using the individual particle images to obtain a final, high-resolution model.

Obtaining a good initial model is crucial, as it strongly influences the result of the refinement step. If we view the reconstruction problem as trying to locate the global optimum of a cost function, then the refinement step is a local search starting from the initial model. An unsuitable initial model would lead the local search to converge to a local optimum instead of a global one.

In some cases, a good initial model might already be available. It could, for example, have been obtained from a previous reconstruction of the same or a similar macromolecular assembly. But in general, algorithms for inferring initial models from class averages form an important step in the data processing pipeline.

The focus of this thesis will be on developing algorithms for inferring initial models.

1.3 Reconstructions from images with known orientations

The previous section outlined the typical pipeline used in cryo-EM for obtaining a high-resolution reconstruction from particle images where the particle orientations are unknown. Algorithms for the initial model step and the refinement step are often based on solutions to the simpler

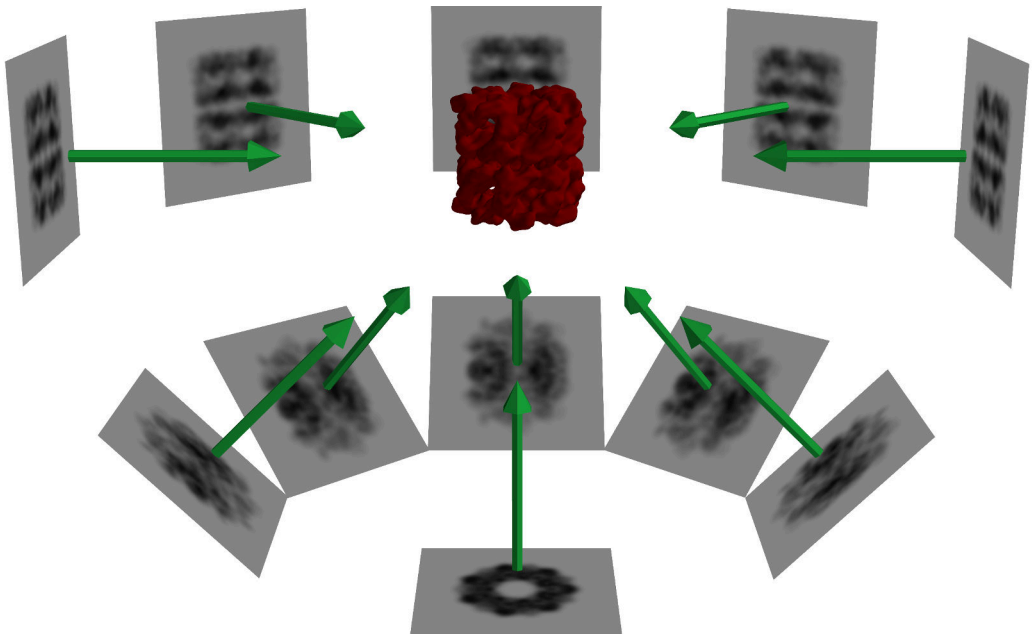

Figure 1.3: An initial three-dimensional reconstruction is computed from the class averages. This initial model is refined using the individual particle images.

problem where the particle orientations are known. If we assume that we know the orientation of the particle corresponding to each image, then there are several efficient and robust algorithms for reconstructing the electron density (see Penczek (2010a) and Frank (2006, Chapter 5) for more detailed overviews).

The reconstructed electron density is usually represented on a regular three-dimensional grid of dimension $L \times L \times L$, say with $L = 100$, and with equal grid spacing in all dimensions. Estimating the L^3 volume parameters given a number of $L \times L$ input images and their orientation parameters is the task of the reconstruction algorithm.

Fourier-based reconstruction algorithms estimate the Fourier representation of the electron density on a regular grid in Fourier space, and then apply the inverse Fourier transform to recover the desired electron density in real space.

The reconstruction in Fourier space is based on the projection-slice theorem (Frank 2006, Chapter 5). We project the electron density in an arbitrary direction, and compute the Fourier transform of the projected image. The theorem then states that this two-dimensional Fourier transform can also be obtained as a slice through the origin of the three-dimensional Fourier transform of the original electron density. The slice should be orthogonal to the projection direction.

To apply this theorem to our input images, we first compute their Fourier transforms, and then orient them in three-dimensional Fourier space orthogonal to their respective projection directions. Given enough images, the resulting irregular sampling of the electron density in

Fourier space can be interpolated to a regular grid. A widely-used Fourier-based reconstruction algorithm that uses this approach is known as direct Fourier inversion (Penczek et al. 2004).

A second reconstruction technique is weighted back-projection (Radermacher 1988). Each image is back-projected to three dimensions in real space by smearing it out along its projection direction. To prevent an over-emphasis on the lower frequencies, and to account for the uneven distribution of projection directions, each image is modified using a two-dimensional weighting function in Fourier space before being back-projected.

Direct Fourier inversion and weighted back-projected are examples of transform methods (Penczek 2010a), because they make use of the projection-slice theorem. The other class of methods are known as algebraic methods.

Algebraic methods (Gilbert 1972; Gordon et al. 1970) formulate the reconstruction problem as a system of linear equations relating the unknown density parameters to the observed images. To each pixel in every image corresponds an equation which discretises the line integral along the projection direction onto that pixel. The unknown density can be estimated by solving the overdetermined system of linear equations. The noise is assumed to be i.i.d. Gaussian noise, leading to a least-squares solution.

Algebraic methods have several advantages, such as formulating the reconstruction problem as a linear least squares problem, for which well-established solutions exist. In contrast to transform methods, they do not need to take into account the uneven distribution of projection directions. Furthermore, they can be modified to use different noise models and to impose other constraints. Their main disadvantage, however, is that they are computationally more expensive than the transform methods.

A popular variation of the algebraic method ART (Algebraic Reconstruction Technique, Gordon et al. (1970)) is ART-with-blobs (Marabini et al. 1998). Instead of the usual voxel-based representation, it uses smooth, spherically symmetric volume elements. These 'blobs' are still located along a regular grid, but because they are spherically symmetric, computing projections becomes faster and more accurate. Compared to weighted back-projection, they produce superior quality reconstructions at a lower computational cost (Marabini et al. 1998).

1.4 Algorithms for inferring initial models

There exist many algorithms for inferring initial models. The problem is still the subject of active research, with at least five new algorithms introduced in the last five years. The different algorithms approach the problem in many different ways. In this section we will identify common themes, and motivate our own approach.

1.4.1 Statistical models

The initial model problem is an inverse problem. We can define model parameters θ to describe the unknown electron density and other relevant parameters. The data $\mathcal{D}$ is the images, either

class averages, or individual particle images. And the forward model describes how to generate the images (the data) from the electron density (the model parameters). The goal is then to do the reverse, i.e. to estimate the model parameters from the data.

The data is corrupted by noise, which we can include in our model by using a statistical forward model. This means that for given values of the model parameters θ, our forward model is a probability distribution $p(\mathcal{D}|\theta)$ over possible data sets. Having specified θ, $\mathcal{D}$ and $p(\mathcal{D}|\theta)$, there are well-established statistical approaches for estimating θ given $\mathcal{D}$.

Defining a statistical model, and using statistical inference to estimate the model parameters is a widely used and natural approach for a problem such as this. Although some of the algorithms for inferring initial models that we will review here do follow this approach explicitly, many others use ad hoc approaches.

In considering the variety of different algorithms, the advantage of the statistical modelling point of view is that even ad hoc algorithms can often be understood as being based on an implicit statistical model, or at least containing elements of a statistical model. This is useful for drawing comparisons between algorithms, and for understanding their implicit assumptions. We will make such comparisons at the end of this chapter, and discuss some of the benefits of using statistical modelling over ad hoc algorithms.

1.4.2 Algorithms based on common lines

Initial model algorithms typically estimate two distinct sets of parameters. Most important is the electron density, represented on a grid of size $L \times L \times L$, for a total of L^3 parameters. In addition, for each of the P input images there is an unknown particle orientation (three parameters) and an unknown in-plane translation (two parameters), a total of $5P$ parameters.

Many algorithms estimate all these parameters simultaneously. Other algorithms take a two-step approach: they first estimate the orientations and translations of the images, and then estimate the electron density using one of the reconstruction algorithms from Section 1.3.

In the two-step approach, the image orientations can be estimated by using the principle of common lines. The principle of common lines follows from the projection-slice theorem introduced in Section 1.3. Consider two images created by projecting an electron density in two non-parallel directions. According to the projection-slice theorem, the Fourier transforms of the images will correspond to two slices through the center of the Fourier transform of the electron density. For non-parallel projection directions, the two slices will intersect in exactly one common line.

Now suppose that we do not know how the two images are oriented relative to each other, but that we have identified the common line in their Fourier transforms. This reduces the number of degrees of freedom of the relative orientation from three to one. By introducing a third image, and locating its common line with each of the first two images, it becomes possible to determine all three image orientations relative to each other.

Additional images are added one at a time, by exhaustively searching through a discrete grid

over all possible orientations. Once an image has been added, its orientation is not modified again, and it serves as a reference for subsequent images. Given enough images, the electron density can be reconstructed using the estimated orientations.

This algorithm is known as angular reconstitution (van Heel 1987). It relies on being able to accurately determine the common lines. It works best for symmetric structures, such as viruses, where three class averages are enough to obtain an initial model. When adding additional images for non-symmetric structures, errors in the initial common line estimates propagate and are amplified. The final reconstruction strongly depends on the choice of the first three images.

Instead of estimating orientations one at a time, Penczek et al. (1996) introduced an algorithm for estimating all orientations simultaneously. By discretising the space of orientations, the problem is formulated as a discrete optimisation problem. However, finding the global optimum is hard, given that the number of possible solutions grows exponentially with the number of images.

More robust optimisation methods such as simulated annealing enabled Elmlund and Elmlund (2012), Elmlund et al. (2010) and Elmlund et al. (2008) to estimate more orientations simultaneously.

Another class of common lines algorithms are based on mathematically sophisticated ideas from convex optimisation, semidefinite relaxation, and spectral methods (Singer and Shkolnisky 2011; Singer et al. 2010; Wang et al. 2013). In contrast to earlier approaches, they introduce a model for errors in common line estimates. A common line between two images is assumed to be either exactly correct, or completely random. Under this assumption, they show that only a small fraction of common lines need to be estimated correctly to determine all orientations correctly.

One drawback of common line algorithms is that it is not clear how to model the errors in common line estimates. These errors are influenced not only by the image noise, but also by the unknown electron density and the projection directions. Part of the difficulty comes from the separation between the parameters describing the electron density and the orientations.

1.4.3 Algorithms based on projection matching

An alternative to the above two-step approach is to estimate both the orientation parameters and the electron density parameters at the same time. Most algorithms that follow this approach are based on projection matching.

Projection matching (Penczek et al. 1994) is the standard algorithm used in the refinement step of the cryo-EM pipeline (Section 1.2). It improves the resolution of the reconstruction by refining the orientation parameters. It is an iterative algorithm which alternates between updating the electron density, and updating the orientations.

At every iteration, the current estimate of the electron density is projected along a discrete grid covering the range of possible projection directions. Every particle image is compared to every projection image by computing the cross-correlation coefficient. The orientation of the

projection image maximising the cross-correlation is used to update the orientation parameters of the particle image.

Using the updated orientation parameters, a new electron density is reconstructed with one of the reconstruction algorithms from Section 1.3.

These two steps are repeated, usually a predetermined number of times. After a few iterations, there are typically only small changes in the orientation parameters, and therefore the discrete grid over all orientations can be replaced by a smaller grid in the neighbourhood of the previous orientation.

Projection matching is widely used, and works well if the initial electron density does not differ too much from the 'true' electron density. Note that there is no explicit or implicit cost function that is being optimised, and therefore no guarantee that the algorithm will converge. Furthermore, there are many algorithmic settings which can influence the final result, such as the choice of discrete grid of orientations at every step, and the function to compare projection images to particle images. Determining appropriate settings requires an experienced user, and can lead to results that are biased.

The simplest initial model algorithm based on projection matching is the random model algorithm (Sanz-Garcia et al. 2010; Yan et al. 2007). A similar algorithm is used by the software suite EMAN2 (Tang et al. 2007). They start with a random model, and refine it using projection matching. Class averages are used instead of individual particle images.

Creating an initial random model can be done in different ways. Sanz-Garcia et al. (2010) assigned random orientations to the input images and used them to reconstruction an initial random model. In contrast, EMAN2 applies a low-pass filter to three-dimensional random noise.

Because it uses projection matching, the result of the random model algorithm will be biased by the initial random model. To make it more robust, the algorithm is usually repeated several times starting from different random models. The corresponding final models are then ranked using different strategies. One strategy is to compute the cross-correlations between the input images and the projections of the final model using the estimated orientations. Sanz-Garcia et al. (2010) evaluated this and several other strategies based on Fourier shell correlation (FSC, see page 53), principal component analysis (PCA) and map variance. They concluded that no single strategy is always reliable, and highlighted the importance of comparing different models by eye.

Similarly to the algorithms based on common lines, random model algorithms also work better if the structure can be assumed to be symmetric, as was done by Yan et al. (2007). Sanz-Garcia et al. (2010) explored asymmetric structures, and found that their method works if there are prominent structural features, but struggles with round and relatively featureless structures.

To summarise, random model algorithms only work on some structures, and lack a reliable way of comparing multiple resulting models. They may also require careful tuning of algorithmic parameters such as the angular step size in each projection matching iteration (Sanz-Garcia et

al. 2010).

Two recent algorithms (Elmlund et al. 2013; Sorzano et al. 2015) modify the projection matching strategy to try not to get trapped in local optima. In particular, they modify the orientation assignment step. In projection matching, each data image is assigned a single orientation, corresponding to the most similar projection image. But there are often several projection images that are similar to the data image, possibly corresponding to very different orientations. This can be due to the noise in the data image, or because the current estimate of the electron density still differs too much from the 'true' density. Instead of trying to choose between these equally worthy candidates, Elmlund et al. (2013) and Sorzano et al. (2015) assign multiple projection images to a single data image. A weight is attached to each projection image to quantify how similar it is to the data image. During the subsequent reconstruction step of projection matching, all the assigned projection images are used, possibly including copies of the same image, one for each data image to which it was assigned.

Replacing hard orientation assignments by soft assignments in this way makes the algorithm more robust. But to turn the above general description into a concrete algorithm, several details have to be specified, such as choosing which projection images to assign to a data image, and how to compute the weights.

Both Elmlund et al. (2013) and Sorzano et al. (2015) use cross-correlations to compare projections and data images. But Sorzano et al. (2015) then choose a predetermined percentage of images with the highest cross-correlations, letting the percentage threshold decrease from 15% to 0.01% over several iterations. In contrast, Elmlund et al. (2013) set the threshold to the highest cross-correlation from the previous iteration, and use only a random subset of the highest cross-correlations.

To determine the weights, Sorzano et al. (2015) directly normalise the cross-correlations themselves, while Elmlund et al. (2013) first apply two successive transformations to the cross-correlations, each involving the exponential function, before normalising the result.

These choices constitute two ad hoc approaches, with many algorithmic parameters that can be adjusted by the user, leading to biased results. We will see below that a similar algorithm follows naturally from a more principled approach, with fewer algorithmic parameters, and ones that are easier to interpret.

These two algorithms also differ in other ways. Instead of using class averages as data images, Elmlund et al. (2013) start with the individual particle images. In this way, the class averaging step is made part of the initial model inference algorithm. The resulting algorithm is computationally very demanding, requiring around 5000 to 10000 CPU hours. In comparison, the algorithm by Sorzano et al. (2015) is relatively fast, needing just a few hours starting from the class averages.

1.4.4 Algorithms with many initial models

The algorithms in the previous section modified projection matching to make it more likely that any starting model will converge to the global optimum. An orthogonal strategy is to use a simpler, faster algorithm instead of projection matching, and then repeat it with a large number of starting models in the hope that at least one will converge to the global optimum.

The algorithms introduced by Lyumkis et al. (2013) and Vargas et al. (2014) use far more starting models than the random model algorithms from the previous section.

Vargas et al. (2014) form about 20 to 50 class averages, and then select 380 random subsets of 4 to 9 class averages. They use non-linear dimensionality reduction to project the original class averages onto a two-dimensional subspace, and then use this representation to ensure that similar class averages are not in the same subset. A reconstruction is obtained from each subset using random orientation assignments. All the reconstructions obtained in this way are ranked according to how well their projections match the input class averages. The best 5 to 10 models are refined using projection matching, and the best one is the result of the algorithm.

Lyumkis et al. (2013) reconstruct up to 1000 different models using a common-lines based approach. The models are aligned, clustered and averaged, and the averages are refined using projection matching. The refined averages are also ranked by comparing their projections to the input class averages.

Compared to the algorithms from the previous section, these two algorithms also have many algorithmic parameters that have to be carefully tuned. As in the case of the random model algorithms, they also need a way to rank their final models, and each introduces its own ad hoc approach to do so.

Similarly to the random model algorithms and the initial common lines based algorithms, most of the examples by Vargas et al. (2014) are of symmetric structures. The only asymmetric structure is the 70S ribosome. On the other hand, their algorithm requires less than an hour on a laptop for the ribosome, which is much faster than most other algorithms. This shows that fast algorithms for inferring initial models are possible.

1.4.5 Algorithms based on a statistical model

In this section we describe algorithms that use a statistical forward model and optimise an explicit cost function. There are not many such algorithms for inferring initial models, and therefore we also include a few refinement algorithms.

All these algorithms use the same forward model, or extensions of it. We first describe the forward model without using grids. Let $\tilde{v} : \mathbb{R}^3 \mapsto \mathbb{R}$ be the electron density, and let $\tilde{y}_i : \mathbb{R}^2 \mapsto \mathbb{R}$ be the ith image. Correspond to the ith image is the ith rotation $R_i \in \mathrm{SO}(3)$, and the ith translation $t_i \in \mathbb{R}^3$.

The ith transformation $\tau_i(x) := R_i x + t_i$ is defined as the composition of the rotation and the translation. The corresponding linear operator $\mathcal{T}(R_i, t_i)$ acts on $\tilde{v}$ to give $\mathcal{T}(R_i, t_i)\tilde{v} : \mathbb{R}^3 \mapsto \mathbb{R}$,

where for $x \in \mathbb{R}^3$:

$$(\mathcal{T}(R_i, t_i)\tilde{v})(x) = \tilde{v}(\tau_i^{-1}(x)) = \tilde{v}(R_i^T(x - t_i)). \tag{1.1}$$

In other words, $\mathcal{T}(R_i, t_i)$ transforms the density by first rotating it by R_i and then translating it by t_i.

Also define a projection operator $\mathcal{P}$ which acts on the density $\tilde{v}$ to give $\mathcal{P}\tilde{v} : \mathbb{R}^2 \mapsto \mathbb{R}$, where for $x = (x_1, x_2, x_3) \in \mathbb{R}^3$:

$$(\mathcal{P}\tilde{v})(x_1, x_2) = \int_{\mathbb{R}} v(x_1, x_2, x_3) dx_3. \tag{1.2}$$

In other words, $\mathcal{P}$ projects the density along the z-axis.

Then according to the forward model (without noise), the ith image is obtained from the density by rotating, translating, and then projecting it:

$$\tilde{y}_i = \mathcal{P}\mathcal{T}(R_i, t_i)\tilde{v}. \tag{1.3}$$

Next we fix a three-dimensional $L \times L \times L$ grid for the density map, and a two-dimensional $L \times L$ grid for the images. The length L^3 density vector v is obtained by evaluating $\tilde{v}$ on the three-dimensional grid, and each length L^2 image vector y_i is obtained from $\tilde{y}_i$ by evaluation on the two-dimensional grid. Corresponding to the linear operator $\mathcal{P}\mathcal{T}(R_i, t_i)$ is the $L^2 \times L^3$ matrix $PT(R_i, t_i)$, which describes how to interpolate the three-dimensional grid to project it in the ith direction.

Discretising the forward model in Eqn. 1.3 and adding i.i.d. Gaussian noise with variance σ^2, the forward model for a single image becomes:

$$y_i = PT(R_i, t_i)v + \epsilon_i. \tag{1.4}$$

If we define the data $\mathcal{D}$ as consisting of all the images y_i, and the model parameters θ as the density v, the rotations R_i, and the translations t_i, then the entire forward model can be written as:

$$p(\mathcal{D}|\theta) = \prod_{i=1}^{P} \mathcal{N}(y_i | PT(R_i, t_i)v, \sigma^2 I), \tag{1.5}$$

where $\mathcal{N}$ denotes the normal distribution, and I is the $L^2 \times L^2$ identity matrix.

Yang et al. (2005) described a refinement algorithm which aims to find the corresponding maximum likelihood estimate:

$$\theta_{\mathrm{ML}} = \arg\max_{\theta} p(\mathcal{D}|\theta) \tag{1.6}$$

$$= \arg\min_{\theta} \sum_{i=1}^{P} \|PT(R_i, t_i)v - y_i\|^2, \tag{1.7}$$

where $\|\cdot\|$ is the Euclidean norm. They approximate the gradients of the cost function in Eqn. 1.7 using finite differences, and use the quasi-Newton optimisation algorithm LBFGS (Nocedal 1980) to find a local optimum.

This algorithm is similar to projection matching. If we fix the rotations and translations in Eqn. 1.7, then we obtain the cost function being optimised by the algebraic reconstruction techniques (Section 1.3), which is the reconstruction step of projection matching. Conversely, if we fix the electron density, then the local rotation updates correspond to the orientation assignment step in projection matching where only a neighbourhood of the current best orientation for each image is explored.

One advantage over projection matching is that several algorithmic parameters are no longer necessary, such as the angular step size and the size of the orientation neighbourhood. Another advantage is that instead of alternating between the two sets of parameters, both are optimised simultaneously. Furthermore, the cost function is guaranteed to decrease at every step.

The cost function of Eqn. 1.7 is also related to the cross-correlation function used to compare projection images and data images in several of the algorithms discussed so far. Let $\hat{y}_i = PT(R_i, t_i)v$ be the projected image. Then

$$\|PT(R_i, t_i)v - y_i\|^2 = \|\hat{y}_i - y_i\|^2 \tag{1.8}$$

$$= \|\hat{y}_i\|^2 - 2\langle \hat{y}_i, y_i \rangle + \|y_i\|^2, \tag{1.9}$$

where $\langle \cdot, \cdot \rangle$ denotes the dot product, which is the unnormalised cross-correlation. Note that the last term $\|y_i\|^2$ is constant. Thus if we assume that the first term $\|\hat{y}_i\|^2$ does not vary too much, then minimising the cost function of Eqn. 1.7 is equivalent to maximising the average cross-correlation between projection images and data images.

Several algorithms discussed so far (Lyumkis et al. 2013; Sanz-Garcia et al. 2010; Vargas et al. 2014) compare multiple reconstructed electron densities based on the average cross-correlation between their projections and the data images. The above argument shows that this is equivalent to comparing their likelihoods (Eqn. 1.6).

Interpreting the algorithms in terms of the forward model and the cost function also gives insight into how they may be changed. For example, Sorzano et al. (2015) suggest using a different distance measure between projection images and data images, one which takes into account local neighbourhoods of pixels. But this can be shown to be equivalent to modifying the noise model to use correlated Gaussian noise instead of i.i.d. noise.

The refinement algorithm by Yang et al. (2005) was presented as a proof of concept, and only applied to simulated data. Another refinement algorithm based on a statistical forward model was introduced by Scheres et al. (2007). It was named ML3D in reference to the maximum likelihood framework.

Scheres et al. (2007) estimate multiple conformations of the electron density simultaneously. This is done by extending the above forward model: every particle image is generated as the projection of one of K distinct electron densities.

All the algorithms presented so far estimate both the electron density and the rotations. Ultimately, however, we are only interested in the electron density. Scheres et al. (2007) take this into account by integrating out the rotations. The integration is approximated by a summation over a discrete grid of rotations.

Using the resulting forward model they again define the cost function as the ML estimate of the model parameters. To optimise the cost function they use expectation maximisation (EM, see Section 2.2.2).

The resulting algorithm is similar to several of the algorithms based on projection matching from Section 1.4.3. The E-step of the EM algorithm is similar to the orientation assignment step of projection matching, while the M-step of the EM algorithm corresponds to the reconstruction step of projection matching.

Integrating out the rotations by computing a sum over a discrete grid of rotations is similar to the way in which soft orientation assignments are computed by Elmlund et al. (2013) and Sorzano et al. (2015).

Subsequent improvements to ML3D include the introduction of explicit prior distributions on the model parameters as needed for a Bayesian approach, and the use of maximum a posteriori (MAP) estimates instead of ML estimates (Scheres 2012a,b).

The ML3D algorithm and its later versions (Scheres 2012b; Scheres et al. 2007) are all refinement algorithms, requiring an initial model. A Bayesian approach to initial model inference was proposed by Jaitly et al. (2010). They estimate only a single electron density, but also integrate out the rotations and derive a cost function based on a MAP estimate.

To make their algorithm robust enough to find the global optimum, they use a quasi-Newton optimisation algorithm combined with simulated annealing. In addition, they downsample the class averages to 32×32.

The resulting algorithm is computationally intensive, requiring about a week for asymmetric structures.

The algorithms in this section are similar in many ways to algorithms presented earlier. The difference is that all our assumptions about the problem are explicitly encoded by the statistical forward model, instead of being hidden in the algorithmic details of ad hoc algorithms. This reduces the potential bias that can be introduced by a user through the tweaking of algorithmic parameters.

A statistical model also allows for a clear distinction between the optimisation algorithm and the forward model. The forward model determines the cost function to optimise, while the optimisation algorithm dictates how that cost function should be optimised. This is useful for understanding why a method does not work, and for suggesting ways of improving it.

The parameters used by the algorithm can also be divided into those defining the forward model, and those needed by the optimisation algorithm. With an ad hoc algorithm, all the parameters could influence both the forward model and the optimisation.

Although there are many advantages to using a statistical approach, one disadvantage is that

the algorithms introduced so far tend to be computationally intensive, and are mostly refinement algorithms that require an initial model.

In this thesis we will introduce algorithms for inferring initial models that are both fast, and use a statistical framework.

Our approach is based on the idea of using a coarse-grained, grid-independent representation of the electron density, in contrast to the grid-based representations used by the algorithms reviewed in this chapter. The coarse-grained representation will use Gaussian mixture models with isotropic components. These have many benefits, such as allowing structures to be represented efficiently with a small number of model parameters and allowing projections to be computed efficiently without grid interpolation.

To estimate the model parameters, we will follow a Bayesian approach, as was done by some of the algorithms reviewed above (Jaitly et al. 2010; Scheres 2012b). We will also introduce priors on the parameters to encode our prior knowledge. But instead of computing just a single MAP (maximum a posteriori) estimate of the mode of the posterior distribution, we will use Markov chain Monte Carlo (MCMC) algorithms for sampling from the full posterior.

The representation using Gaussian mixture models and the MCMC algorithms for sampling the model parameters will be introduced in Chapter 2. We will also show how the same approach can be used for related problems, such as computing a coarse-grained representation from a three-dimensional density, or reconstructing a mixture model representation from class averages with known orientations.

The algorithms will be applied to simulated data in Chapter 3. Different structures will be used, and the effect of the different model parameters and algorithmic parameters will be studied in depth.

In Chapter 4 we will apply the algorithms to real data, in particular to the 70S ribosome, GroEL, and APC/C. We will also introduce a CTF-correction algorithm for preparing the class averages.

Chapter 5 will introduce a variation on the previous algorithms by changing the noise model. The resulting algorithm will be applied to simulated and real data, and compared to the earlier approach.

Conclusions and an outlook for future work is presented in Chapter 6.

Chapter 2

Reconstruction algorithms using Gaussian mixture models

All the initial model algorithms reviewed in the previous chapter use a regular grid to represent the electron density. Such a grid typically has a large number of parameters (the number of voxels), allowing it to represent electron densities at a high resolution.

But initial models do not need to be have a high resolution. They only have to be accurate enough for the subsequent refinement step to converge to a good solution. One way to limit the resolution of initial models is by removing high-resolution models from the search space. This would simplify the optimisation problem, leading to more robust algorithms. It would also reduce the computation time.

Some algorithms remove high-resolution models by using a coarser grid, with $64 \times 64 \times 64$ being a common choice (Scheres et al. 2007; Sorzano et al. 2015). Other algorithms suppress high-resolution models by applying low-pass filters, either to the input class averages (Vargas et al. 2014), or to the three-dimensional density at every iterative step of the reconstruction algorithm (Scheres 2012a).

In this thesis, we address this problem by using a completely different model to represent the electron density, namely a Gaussian mixture model. We will show that this significantly reduces the number of parameters used by previous approaches, even ones using coarser grids. It also excludes high-resolution densities from the parameter space, thereby removing the need for ad hoc low-pass filters.

The first part of this chapter will be about the alternative density model. In the rest of the chapter we will define an appropriate statistical forward model, and introduce algorithms for estimating the model parameters.

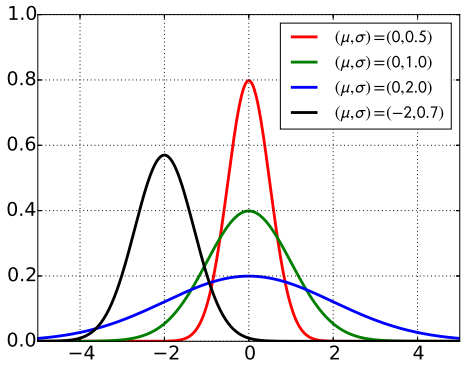

Figure 2.1: Examples of one-dimensional Gaussian distributions with different parameters. The mean μ determines the center of the distribution, and the standard deviation σ determines its width.

2.1 Gaussian mixture models

2.1.1 One-dimensional GMMs

Our building block is the Gaussian distribution, also known as the normal distribution. The one-dimensional Gaussian distribution is parameterised by its mean μ and variance σ^2, where σ is referred to as the standard deviation. The distribution has the following probability density function:

$$\mathcal{N}(x|\mu,\sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left\{\frac{-(x-\mu)^2}{2\sigma^2}\right\}. \tag{2.1}$$

The mean μ determines the position of the distribution, and the standard deviation σ determines its width (Fig. 2.1).

Individual Gaussian distributions cannot accurately approximate complicated distributions. A wider class of distributions can be obtained by combining multiple Gaussian distributions in a weighted sum, known as a Gaussian mixture model (GMM):

$$h(x) = \sum_{k=1}^{K} w_k \mathcal{N}(x|\mu_k,\sigma_k^2). \tag{2.2}$$

The individual Gaussian distributions in Eqn. 2.2 are referred to as the components of the mixture. For the mixture model to be a valid probability density function, the weights w_k must be non-negative and sum to one ($\sum_k w_k = 1$).

Fig. 2.2 shows two examples of one-dimensional Gaussian mixture models. They represent two possible ways of restricting the family of one-dimensional mixture models: by requiring all weights to be the same ($w_k = 1/K$ for all k), or all standard deviations to be the same ($\sigma_k = \sigma$ for all k). We will make use of such restrictions for two reasons. One reason is that they

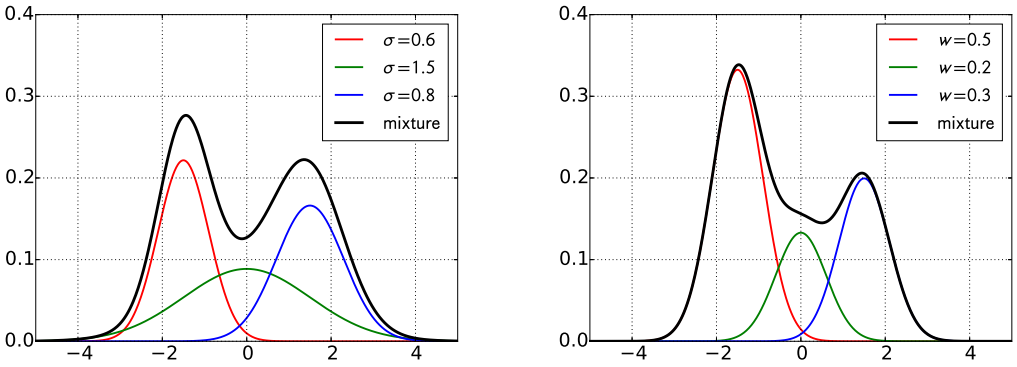

Figure 2.2: Examples of one-dimensional Gaussian mixture models (*black*) with three components (*red, green, blue*). (*Left*) Each Gaussian component has a different size σ, but the same weight $w = 1/3$. (*Right*) Each component has a different weight w, but the same size $\sigma = 0.6$. Despite having significantly different components, the two mixtures (*black*) look similar. In this thesis our mixture models will be similar to the one on the right.

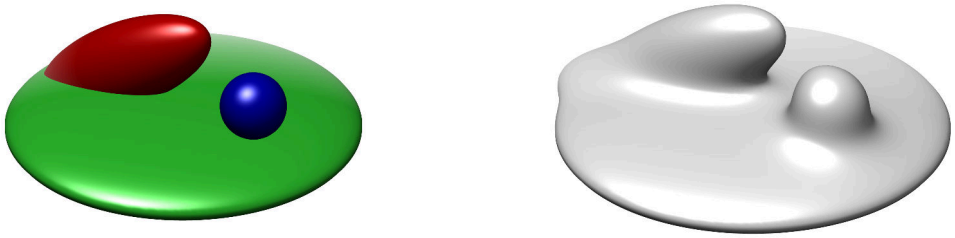

Figure 2.3: Three individual Gaussians (*left*) form the components of a Gaussian mixture model (*right*). The spherical blue component is an example of an isotropic Gaussian.

reduce the number of model parameters to be estimated while still leaving enough flexibility to approximate fairly complicated functions. The second reason is that such simplifications will allow us to implement significantly more efficient algorithms.

2.1.2 Higher-dimensional GMMs

We will mainly work with two- and three-dimensional probability distributions. The d-dimensional analog of the one-dimensional Gaussian distribution is the multivariate Gaussian distribution:

$$\mathcal{N}(x|\mu,\Sigma) = \frac{1}{(2\pi)^{d/2}|\Sigma|^{1/2}} \exp\left\{-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu)\right\}, \tag{2.3}$$

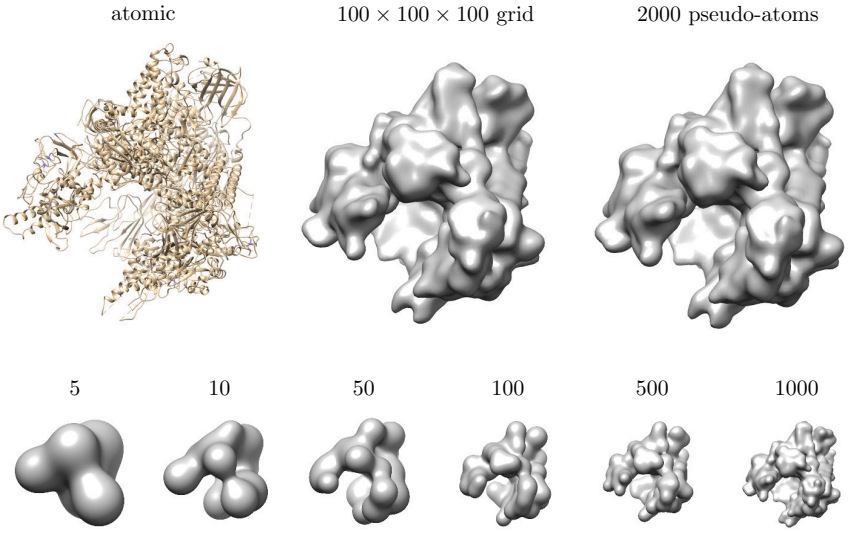

Figure 2.4: Example of a cryo-EM density represented using a Gaussian mixture model. The atomic structure of RNA polymerase II (*top left*) is used to simulate a cryo-EM density on a regular grid (*top middle*). The electron density is almost indistinguishable from a mixture model representation with 2000 isotropic Gaussian components (*top right*). Different levels of accuracy can be achieved by varying the number of components from 5 to 1000 (*bottom row*).

which is parameterised by its mean $\mu \in \mathbb{R}^d$ and covariance matrix $\Sigma \in \mathbb{R}^{d\times d}$. The corresponding d-dimensional Gaussian mixture model is:

$$h(x) = \sum_{k=1}^{K} w_k \mathcal{N}(x|\mu_k, \Sigma_k). \tag{2.4}$$

Fig. 2.3 shows examples of three-dimensional Gaussians and Gaussian mixture models. The level sets of a three-dimensional Gaussian is an ellipsoid, whose shape and orientation is determined by its covariance matrix Σ, a positive-definite 3×3 matrix. By restricting the covariance matrix to be a multiple of the identity matrix

$$\Sigma = \sigma^2 I = \begin{bmatrix} \sigma^2 & 0 & 0 \\ 0 & \sigma^2 & 0 \\ 0 & 0 & \sigma^2 \end{bmatrix}, \tag{2.5}$$

the level sets become spherical, and the Gaussian is said to be *isotropic*. The expression for its probability density function also simplifies (cf. Eqn. 2.1):

$$\mathcal{N}(x|\mu, \sigma^2) = \frac{1}{(2\pi)^{d/2}\sigma^d} \exp\left\{\frac{-\|x-\mu\|^2}{2\sigma^2}\right\}. \tag{2.6}$$

In this thesis we will use only isotropic Gaussian distributions. In addition, all the (isotropic) components of a Gaussian mixture model will share a common variance. Instead of the $6K$ para-

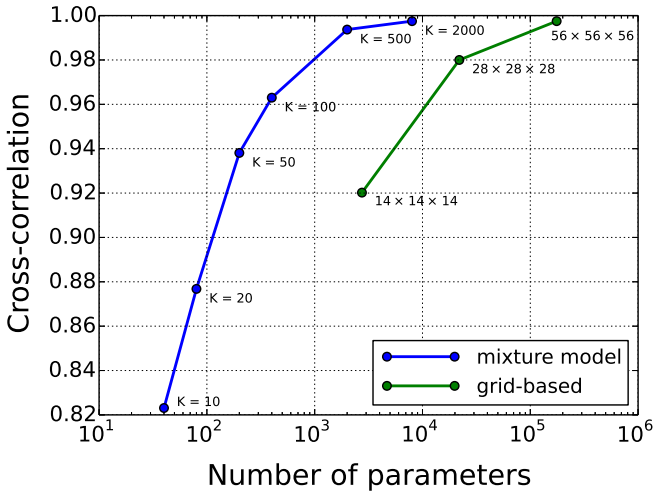

Figure 2.5: The number of parameters required for the mixture model representation as compared to the standard grid-based representation of Pol II, using the models from Fig. 2.4. The accuracy of every model is given by its cross-correlation relative to the original reference structure on a $112 \times 112 \times 112$ grid. The grid-based representations were obtained by downsampling the reference structure by factors of 2, 4 and 8. The accuracy of the mixture models increases with the number of components (K). Similarly, the accuracy of the grid-based representation increases with the number of voxels. But for any specified level of accuracy, the mixture model representation needs fewer than 10% of the number of parameters needed by the grid-based representation.

meters needed to describe all the covariance matrices of a general three-dimensional Gaussian mixture model, we will just have a single parameter.

The resulting isotropic mixture models can still approximate complicated distributions, as shown in Fig. 2.4. Using isotropic mixture models also simplifies and speeds up the algorithms for estimating the model parameters. Nevertheless, it is possible to extend the algorithms that we will introduce to work on mixture models with full covariance matrices. But in our experience the gain in expressiveness does not justify the loss in computational efficiency. Furthermore, using a single component size ensures that the density has a relatively uniform resolution.

Instead of using the variance σ^2 we will use the precision $s = 1/\sigma^2$. This simplifies the analysis. Combining Eqns. 2.4 and 2.6, we obtain the final form of the probability density function that we will use to describe electron densities (in $d = 3$ dimensions):

$$h(x) = \sum_{k=1}^{K} w_k \left(\frac{s}{2\pi}\right)^{d/2} \exp\left\{\frac{-s}{2}\|x - \mu_k\|^2\right\}. \tag{2.7}$$

Fig. 2.4 shows an example of using the isotropic mixture model to represent an electron density map. By increasing the number of components (K), the mixture model representation can achieve the same resolution as the traditional grid-based approach.

The advantage of the mixture model representation is that it requires far fewer parameters. In contrast to the grid-based representation, which needs one parameter for each voxel, the mixture model has only $4K$ parameters. In the Pol II example shown in Fig. 2.5, the grid-based representation needs at least an order of magnitude more parameters to represent the density map at a level of accuracy similar to that of the mixture model representation.

The parameters of the mixture model are the means μ_k, the K weights w_k, and the component precision s. Algorithms for estimating these parameters are described starting from Section 2.2.

2.1.3 Additional uses of the mixture model representation

Once a density map has been reconstructed using cryo-EM, it can be used in various ways to help determine the atomic structure and dynamics of the macromolecular assembly. Some of these approaches benefit from using a coarser representation of the density map, such as a Gaussian mixture model.

One example is fitting known subunits into the density map. Suppose that the macromolecular assembly consists of several subunits whose atomic structures are known, but where it is not known how they fit together to form the entire assembly. By rotating and translating them to fit into the density map without overlapping each other, it is possible to estimate the atomic structure of the whole assembly.

Kawabata (2008) approaches the subunit fitting problem by representing both the subunits and the electron density of the entire structure as Gaussian mixture models. This significantly reduces the computational cost of evaluating a given configuration of subunits.

Another example is studying the dynamics of a macromolecular assembly: how it adopts different conformations in performing its function. Nogales-Cadenas and Jonic (2013) and Jin and Sorzano (2014) explore these conformational changes by doing a normal mode analysis of the atomic structure of the assembly. They extend their algorithm to work with density maps instead of atomic structures by using a Gaussian mixture model representation of the density map, and treating the mixture components as large atoms.

Both these algorithms take a standard electron density obtained using any cryo-EM reconstruction algorithm, convert it into a mixture model, and then analyse this mixture model. In constrast, our goal is to introduce new reconstruction algorithms which directly produce a mixture model, without the intermediate density map. The mixture model that we obtain could then be used as input to either of the above algorithms. As far as we are aware, our approach is the first to use a mixture model representation of electron densities for the cryo-EM reconstruction problem.

The two algorithms mentioned above use different families of Gaussian mixture models. Kawabata (2008) uses mixture components with full covariance matrices. In contrast, Jin and Sorzano (2014) use the same isotropic mixture model representation that we use, also with a single variance for all components. They refer to the mixture components as *pseudo-atoms* to emphasise the similarity to an atomic model, and we will occasionally use the same terminology.

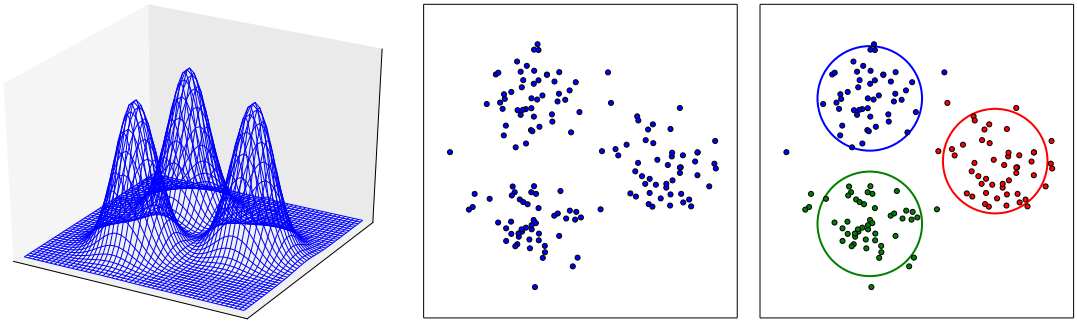

Figure 2.6: Sampling a point cloud from a mixture model. (*Left*) The mixture model has three components. (*Middle*) The points sampled from the mixture model form the data $\mathcal{D}$. (*Right*) The points are coloured according to the component from which they were sampled. This extra information is the missing data, or latent variables $\mathcal{Z}$.

Note that a pseudo-atom is typically much larger than a real atom, and that it does not represent a specific set of atoms, or a subunit. We will sometimes refer to a Gaussian mixture model as a *pseudo-atom model*.

2.2 Estimating GMM parameters

The Gaussian mixture model was introduced in the previous section as a representation for an electron density:

$$h(x) = \sum_{k=1}^{K} w_k \left(\frac{s}{2\pi}\right)^{d/2} \exp\left\{\frac{-s}{2}\|x-\mu_k\|^2\right\} \tag{2.8}$$

$$= \sum_{k=1}^{K} w_k \mathcal{N}(x|\mu, s^{-1}I). \tag{2.9}$$

The parameters of the mixture model are denoted by θ, and consist of the means μ_k, the weights w_k and the precision s. The rest of this chapter will be devoted to algorithms for estimating θ. The final algorithm will estimate θ for a three-dimensional mixture model given two-dimensional images, which corresponds to the initial model inference problem. But the different aspects of the algorithm will be introduced incrementally, starting with the simplest case: fitting a d-dimensional mixture model to a d-dimensional point cloud.

Two well-known algorithms for fitting mixture models to point clouds are expectation maximisation (EM) and Gibbs sampling (Bishop 2006). They will form the foundation for the extensions to follow, and are reviewed below.

2.2.1 Statistical forward model

Before applying either EM or Gibbs sampling, the relation between the model parameters and the data must first be expressed as a statistical forward model.

The data $\mathcal{D}$ is a set of N d-dimensional points $x_1, x_2, \ldots, x_N \in \mathbb{R}^d$ (Fig. 2.6). Each data point x_i is assumed to have been sampled independently from an isotropic Gaussian mixture model:

$$p(x_i|\theta) = \sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu, s^{-1}I) \tag{2.10}$$

$$p(\mathcal{D}|\theta) = \prod_{i=1}^{N} p(x_i|\theta) \tag{2.11}$$

$$= \prod_{i=1}^{N} \sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1}I). \tag{2.12}$$

The forward model $p(\mathcal{D}|\theta)$ is known as the likelihood, and describes how the data $\mathcal{D}$ is obtained given the model parameters θ.

Each data point x_i is sampled by first choosing a mixture component, and then sampling x_i from the Gaussian distribution for that component.

A mixture component is chosen with probability proportional to its weight w_k, i.e. its index is sampled from a categorical distribution with the weights w_k as parameters. Let z_i denote the component assigned to x_i, using the 1-of-K notation. This means that z_i is a length K vector $(\ldots, z_{ik}, \ldots)$ where $z_{ik} \in \{0, 1\}$ and $\sum_k z_{ik} = 1$. The index of the only non-zero entry of the vector is the index of the assigned component. The set of all component assignments z_i is denoted by $\mathcal{Z}$.

The component assignments $\mathcal{Z}$ are part of the description of the forward model, but are not observed in the final data set (Fig. 2.6). They are known as *missing data*, or *latent variables*. Counterintuitively, the parameter estimation problem becomes simpler when the model parameters θ are augmented to include the latent variables $\mathcal{Z}$. This requires a reformulation of the forward model to include the latent variables.

The extended likelihood $p(\mathcal{D}, \mathcal{Z}|\theta)$ describes how both the data $\mathcal{D}$ and the latent variables $\mathcal{Z}$ are obtained given the model parameters θ. It can be factorised:

$$p(\mathcal{D}, \mathcal{Z}|\theta) = p(\mathcal{D}|\mathcal{Z}, \theta)p(\mathcal{Z}|\theta). \tag{2.13}$$

The factorisation corresponds to the two steps in sampling a point x_i as described above, by first choosing the component z_i according to

$$p(\mathcal{Z}|\theta) = \prod_{i=1}^{N} p(z_i|\theta) = \prod_{i=1}^{N} \prod_{k=1}^{K} w_k^{z_{ik}}, \tag{2.14}$$

and then sampling x_i from the chosen component:

$$p(\mathcal{D}|\mathcal{Z},\theta) = \prod_{i=1}^{N} p(x_i|z_i,\theta) = \prod_{i=1}^{N}\prod_{k=1}^{K} \mathcal{N}(x_i|\mu_k, s^{-1}I)^{z_{ik}}. \tag{2.15}$$

Note that the product over k in Eqns. 2.14 and 2.15 consists of only a single term, the one for which z_{ik} is non-zero. Combining Eqns. 2.14 and 2.15 yields the extended likelihood

$$p(\mathcal{D},\mathcal{Z}|\theta) = \prod_{i=1}^{N}\prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)\right]^{z_{ik}}. \tag{2.16}$$

The original likelihood $p(\mathcal{D}|\theta)$ can be recovered as the marginal distribution of the extended likelihood:

$$\sum_{\mathcal{Z}} p(\mathcal{D},\mathcal{Z}|\theta) = \sum_{\mathcal{Z}} \prod_{i=1}^{N}\prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)\right]^{z_{ik}} \tag{2.17}$$

$$= \prod_{i=1}^{N}\sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1}I) \tag{2.18}$$

$$= p(\mathcal{D}|\theta). \tag{2.19}$$

Computing the marginal distribution corresponds to discarding the component assignments after sampling them along with the data.

The forward model introduced above (Eqn. 2.12) specifies how to generate data $\mathcal{D}$ given known model parameters θ. Algorithms for estimating mixture model parameters need to do the reverse: they must estimate θ given known data $\mathcal{D}$.

A common approach to this inverse problem is to compute the maximum likelihood estimate:

$$\theta_{\mathrm{ML}} = \arg\max_{\theta} p(\mathcal{D}|\theta) \tag{2.20}$$

For some forward models $p(\mathcal{D}|\theta)$ the maximum likelihood estimate θ_{ML} can be calculated directly. For example, given a set of points sampled from a single Gaussian distribution, there are simple expressions for the maximum likelihood estimates of its mean and (co)variance.

But for mixture models, finding the global optimum of the likelihood function is in general not possible. Instead, algorithms such as EM (to be discussed below) are used to find a local optimum.

The Bayesian approach extends the statistical framework consisting of the data $\mathcal{D}$, model parameters θ and likelihood $p(\mathcal{D}|\theta)$ by including a *prior* distribution $p(\theta)$ over the parameters. The prior encodes what we know about the model parameters before having seen the data. An example of prior knowledge could be constraints that some parameters must satisfy, such as non-negativity constraints.

For a given form of the likelihood function, there is often a natural parametric form of the prior, chosen for its mathematical convenience. The natural prior for Gaussian mixture models

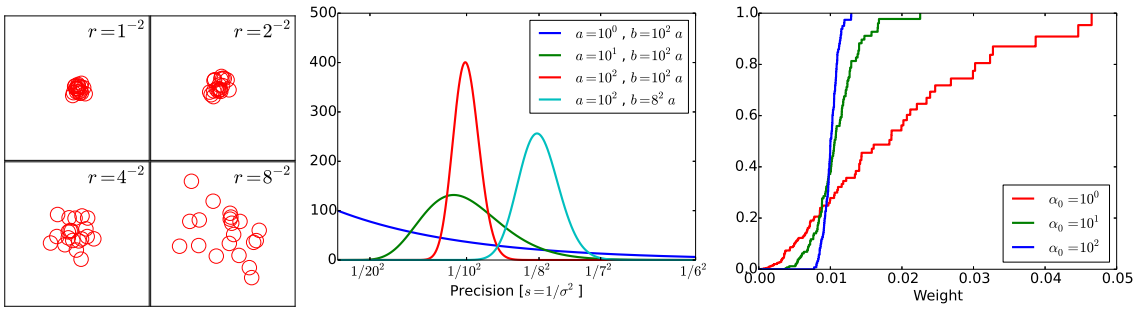

Figure 2.7: The effect of the four hyperparameters of the prior. (*Left*) The mean hyperparameter r specifies how much the components overlap, and the extent of the region that they cover. (*Middle*) The precision hyperparameters a and b indicate the range of plausible component sizes. (*Right*) As the weight hyperparameter α_0 increases, its cumulative distribution function approaches a step function, indicating less variance in the weights.

is

$$p(\theta) = p(\mu|s)p(s)p(w), \tag{2.21}$$

where

$$p(\mu|s) = \prod_{k=1}^{K} p(\mu_k|s) = \prod_{k=1}^{K} \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \tag{2.22}$$

$$p(s) = \text{Gamma}(s|a,b) \propto s^{a-1}e^{-bs} \tag{2.23}$$

$$p(w) = \text{Dirichlet}(w|\alpha_0) \propto \prod_{k=1}^{K} w_k^{\alpha_0-1}. \tag{2.24}$$

The prior can be adjusted using four hyperparameters (Fig. 2.7).

The prior on the component precision is a Gamma distribution with parameters a and b (Eqn. 2.23). Choosing these parameters determines a range of values considered plausible for the component precision, and thus indirectly for the component variance (s^{-1}) (Fig. 2.7).

The prior on each component mean is a Gaussian distribution (Eqn. 2.22). Its hyperparameter r determines the size of the region around the origin where the components should be located (Fig. 2.7). For cryo-EM, the size of this region corresponds to the expected size of the electron density map.

In the mean prior (Eqn. 2.22), the variance of the Gaussian depends on the component precision s. This means that the size of the region covered by the components is also affected by the hyperparameters a and b. The dependency on the component precision s can be removed to obtain an alternative prior on the means:

$$p(\mu) = \prod_{k=1}^{K} p(\mu_k) = \prod_{k=1}^{K} \mathcal{N}(\mu_k|0, r^{-1}). \tag{2.25}$$

In the alternative prior, the hyperparameter r is simpler to interpret. The derivation of the EM algorithm is not possible with this alternative prior, but it can be used for Gibbs sampling.

The prior on the weights is a Dirichlet distribution (Eqn. 2.24). Its hyperparameter α_0 determines how uniformly the weights are distributed (Fig. 2.7).

The hyperparameters of the prior are assumed to be known, and are kept fixed during the execution of any of the algorithms to be introduced. Guidelines for choosing the hyperparameters and their effect on the results will be explored in Section 3.1.5.

Given the prior, the *posterior* distribution $p(\theta|\mathcal{D})$ is computed using Bayes' rule:

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})} \propto p(\mathcal{D}|\theta)p(\theta). \tag{2.26}$$

The posterior distribution $p(\theta|\mathcal{D})$ captures more information than is given by the maximum likelihood estimate θ_{ML}. Its mode is known as the maximum a-posteriori (MAP) estimate

$$\theta_{\mathrm{MAP}} = \arg\max_{\theta} p(\theta|\mathcal{D}) = \arg\max_{\theta} p(\mathcal{D}|\theta)p(\theta). \tag{2.27}$$

The MAP estimate is the same as the ML estimate (Eqn. 2.20) except for the introduction of the prior. It represents a compromise between fitting the data (by increasing the likelihood $p(\mathcal{D}|\theta)$) and satisfying the prior constraints (by increasing the prior $p(\theta)$).

The same EM algorithm that is used to compute the ML estimate for fitting Gaussian mixture models to point clouds can be used for computing the MAP estimate, with only slight modifications to take the prior into account. It is the latter version that is described below (Section 2.2.2).

The full posterior $p(\theta|\mathcal{D})$ contains more information than just its mode, the MAP estimate. For instance, the width of the posterior around the mode conveys the precision of the MAP parameter estimates.

There is no tractable analytic solution for the full posterior distribution for Gaussian mixture models. One approach to exploring the posterior is to generate samples from it. A common sampling algorithm for Gaussian mixture models is Gibbs sampling, which will be described below (Section 2.2.3).

2.2.2 Expectation maximisation

Expectation maximisation (Dempster et al. 1977) is a general algorithm for use in situations where there is some form of missing data. In the mixture model application, the missing data is the component assignments $\mathcal{Z}$. This section will describe the classical application of EM to Gaussian mixture models, adapted to the restricted family of isotropic mixture models. The same general EM approach will be applied in subsequent sections with additional missing data.

For the application to cryo-EM, Gibbs sampling will be used instead of expectation maximisation. The two approaches are very similar, however, and EM might be more familiar due

to being used by other reconstruction algorithms (Scheres et al. 2007), and in other cryo-EM applications (Kawabata 2008).

The goal is to find a local maximum θ_{MAP} of the posterior distribution $p(\theta|\mathcal{D})$:

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\theta|\mathcal{D}). \tag{2.28}$$

Using Bayes' rule (Eqn. 2.26) and noting that the evidence $p(\mathcal{D})$ is independent of θ:

$$\theta_{\text{MAP}} = \arg\max_{\theta} p(\mathcal{D}|\theta)p(\theta). \tag{2.29}$$

Finally, it is computationally convenient to take the logarithm of the argument. Since the logarithm is a strictly increasing function, this does not affect the value of θ at a posterior mode, and thus

$$\theta_{\text{MAP}} = \arg\max_{\theta} \left[\log p(\mathcal{D}|\theta) + \log p(\theta)\right]. \tag{2.30}$$

The first term of the argument in Eqn. 2.30 is known as the log-likelihood

$$\log p(\mathcal{D}|\theta) = \sum_{i=1}^{N} \log p(x_i|\theta) \tag{2.31}$$

$$= \sum_{i=1}^{N} \log \sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1}I). \tag{2.32}$$

The second term of the argument in Eqn. 2.30 is known as the log-prior, and their sum is the log-posterior.

Direct optimisation of the log-posterior is intractable because the logarithm in Eqn. 2.32 cannot be brought inside the summation over k. EM provides a way of changing the sum to a product, which then allows the logarithm to be brought inside the expression. This is done by introducing the component assignments as latent variables. We will give an informal description of the algorithm (following Bishop (2006)), and refer to Appendix A for a detailed derivation.

A naive approach is to try to optimise the log-posterior by setting its derivatives w.r.t. the model parameters to zero. Recall from Eqn. 2.30, that up to a constant, the log-posterior is

$$\log p(\mathcal{D}|\theta) + \log p(\theta). \tag{2.33}$$

Setting the derivative of the log-posterior w.r.t. μ_k to zero, and solving for μ_k gives:

$$\mu_k = \frac{1}{N_k + r} \sum_{i=1}^{N} r_{ik} x_i, \tag{2.34}$$

where

$$r_{ik} = \frac{w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)}{\sum_l w_l \mathcal{N}(x_i|\mu_l, s^{-1}I)} \tag{2.35}$$

$$N_k = \sum_{i=1}^{N} r_{ik}. \tag{2.36}$$

The variables r_{ik} are called the soft assignments. To distinguish them from the assignments z_{ik} defined in Section 2.2.1, the latter will be refer to as the hard assignments.

The definition of the soft assignments in Eqn. 2.35 is equivalent to

$$r_{ik} \propto w_k \mathcal{N}(x_i|\mu_k, s^{-1}I), \quad \sum_{k=1}^{K} r_{ik} = 1. \tag{2.37}$$

Note the similarity with the hard assignments, for which $\sum_k z_{ik} = 1$ also holds. But instead of assigning a data point x_i to a single component, the vector $(r_{i1}, r_{i1}, \ldots, r_{iK})$ can be seen as partitioning the assignment between several different components, each receiving just a fraction of the data point. Nearby components, or components with large weights will receive a larger fraction.

Continuing with setting derivatives of Eqn. 2.33 to zero, the corresponding expressions for the precision and the weights are

$$s = \frac{(N+K)d + 2(a-1)}{\sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|x_i - \mu_k\|^2 + r\sum_{k=1}^{K}\|\mu_k\|^2 + 2b} \tag{2.38}$$

$$w_k = \frac{N_k + (\alpha_0 - 1)}{N + K(\alpha_0 - 1)}. \tag{2.39}$$

These three expressions for the parameters (Eqns. 2.34, 2.38, 2.39) do not define a global optimum, because they depend on r_{ik}, which itself depends on all the parameters (Eqn. 2.35). But it does suggest an iterative algorithm which alternates between updating the assignments r_{ik} and the model parameters μ, s and w.

This iterative algorithm is the EM algorithm. First the initial values for the parameters are assigned, for instance by sampling them from the prior. Then the parameter values are used to compute the soft assignments using Eqn. 2.35 in what is called the E-step. Next these soft assignments are used to compute new parameter values using Eqns. 2.34, 2.38, 2.39 during what is known as the M-step. The algorithm continues to alternate between the two steps until convergence.

Fig. 2.8 shows an example of the EM algorithm in two dimensions.

2.2.3 Gibbs sampling

Gibbs sampling (Geman and Geman 1984) is a Markov-chain Monte Carlo (MCMC) algorithm for sampling from the posterior $p(\theta|\mathcal{D})$. Whereas the EM algorithm was used to obtain a single estimate $\hat{\theta}$ of the model parameters, a Gibbs sampler produces multiple parameters $\{\theta_1, \theta_2, \ldots, \theta_T\}$. The EM estimate $\hat{\theta}$ approximates the posterior mode θ_{MAP}, while the Gibbs samples typically explore the region around the mode. The variation in the parameter values θ_t around the mode convey the precision with which the model parameters can be determined given the data.

As in the case of the EM algorithm, Gibbs sampling also makes use of the component assignments as latent variables $\mathcal{Z}$. The idea is to use a Gibbs sampler to draw samples from the

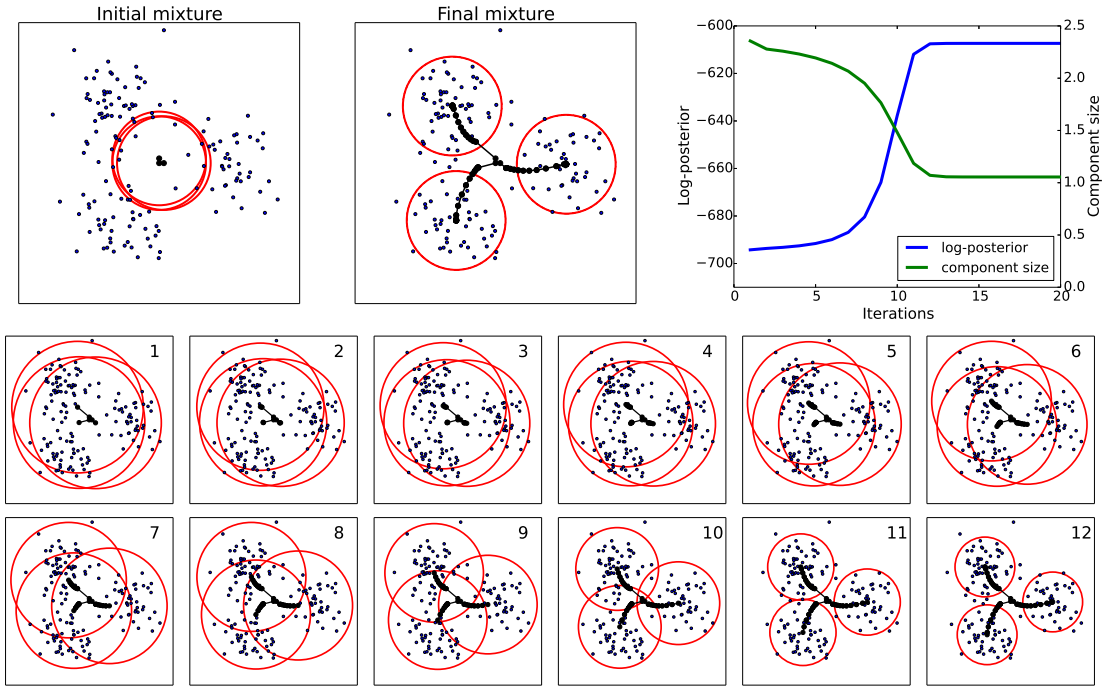

Figure 2.8: Using the EM algorithm to estimate the parameters of a two-dimensional isotropic Gaussian mixture with three components. The data (*blue dots*) was sampled from a three-component mixture. (*Top left*) The initial components (*red circles*) are centered around the origin, but after 20 EM iterations (*top middle*) the components match the data well. The means of the components are shown as *black dots* with *black lines* linking means from successive iterations. (*Middle and bottom row*) The algorithm has converged after about 12 iterations. (*Top right*) The log-posterior increases after every iteration, and the component size decreases to match the value of 1 used to generate the original data.

extended posterior $p(\theta, \mathcal{Z}|\mathcal{D})$:

$$\{(\theta_1, z_1), (\theta_2, z_2), \ldots, (\theta_T, z_T)\}, \tag{2.40}$$

and discard the component assignments to obtain samples from the marginal distribution $p(\theta|\mathcal{D})$:

$$\{\theta_1, \theta_2, \ldots, \theta_T\}. \tag{2.41}$$

The idea behind Gibbs sampling is to partition the parameters that constitute θ and $\mathcal{Z}$ into sets, and iteratively sample each set of parameters conditioned on the other parameters. In our case the parameters are partitioned into four sets: the means μ, the precision s, the weights w, and the component assignments z.

Given the current parameter values (μ_t, s_t, w_t, z_t), the values for the next step are sampled

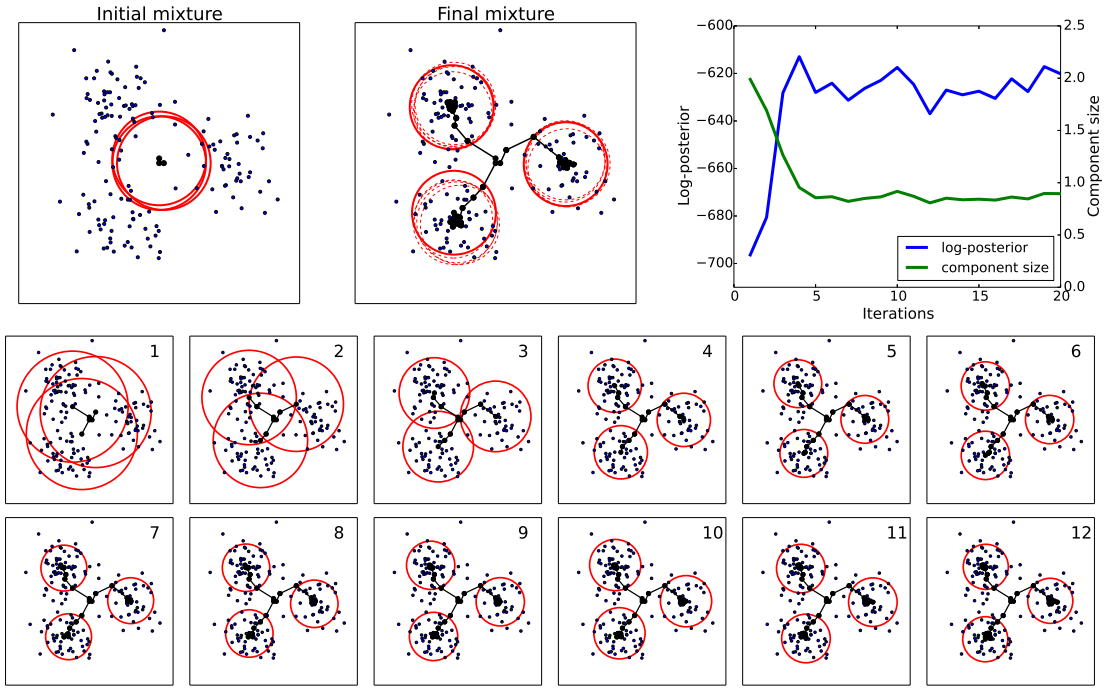

Figure 2.9: Using Gibbs sampling to estimate the parameters of the same two-dimensional mixture model using the same data as in Fig. 2.8. (*Bottom two rows*) Compared to the EM algorithm, the Gibbs sampler converges faster to the correct solution. (*Top middle*) The Gibbs samples (*dotted red circles*) are all close to the optimal solution found by the EM algorithm. Increasing the number of data points would decrease the amount of variation. (*Top right*) The log-posterior is no longer monotonically increasing, but instead oscillates just below the optimal value found by the EM algorithm.

from the following conditional distributions, one after the other:

$$p(z_{t+1}|\mu_t, s_t, w_t, \mathcal{D}) \tag{2.42}$$

$$p(\mu_{t+1}|s_t, w_t, z_{t+1}, \mathcal{D}) \tag{2.43}$$

$$p(s_{t+1}|\mu_{t+1}, w_t, z_{t+1}, \mathcal{D}) \tag{2.44}$$

$$p(w_{t+1}|\mu_{t+1}, s_{t+1}, z_{t+1}, \mathcal{D}) \tag{2.45}$$

The Gibbs sampler is initialised in the same way that the EM algorithm was initialised, for example by sampling $\theta_0 = (\mu_0, s_0, w_0)$ from the prior, and z_0 from its conditional distribution.

The expressions for the conditional distributions are (see Appendix B for the derivation):

$$p(z_i|\mu, s, w, \mathcal{D}) = \prod_{k=1}^{K} r_{ik}^{z_{ik}} \tag{2.46}$$

$$p(\mu_k|s, w, z, \mathcal{D}) = \mathcal{N}\left(\mu_k \Big| \frac{\sum_{i=1}^{N} z_{ik} x_i}{N_k + r}, \frac{1}{s(N_k + r)} I\right) \tag{2.47}$$

$$p(s|\mu, w, z, \mathcal{D}) = \text{Gamma}(s|\tilde{a}, \tilde{b}) \tag{2.48}$$

$$2\tilde{a} = 2a + (N + K)d \tag{2.49}$$

$$2\tilde{b} = 2b + \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|x_i - \mu_k\|^2 + r \sum_{k=1}^{K} \|\mu_k\|^2 \tag{2.50}$$

$$p(w|\mu, s, z, \mathcal{D}) \propto \prod_{k=1}^{K} w_k^{N_k + \alpha_0 - 1}, \tag{2.51}$$

where r_{ik} is defined as for the EM algorithm (Eqn. 2.35), $N_k = \sum_{i=1}^{N} z_{ik}$ and $N = \sum_{k=1}^{K} N_k$.

There are strong similarities between the Gibbs sampling algorithm and the EM algorithm. Both algorithms produce a sequence of parameter values. While the EM algorithm discards all but the last term of the sequence, the Gibbs sampling algorithm typically discards a number of samples at the start of the sequence (the *burn-in* period), and then picks every nth sample (with $n = 50$ for example) to reduce the dependency between successive samples.

The division of each step of the EM algorithm into an E-step and an M-step is also reflected by the conditional distributions of the Gibbs sampler. The soft assignments r_{ik} that are computed during the E-step also appear in the conditional distribution for the hard assignments z_i (Eqn. 2.46). The other conditional distributions are for sampling the model parameters, which corresponds to the M-step.

Furthermore, each of the four EM update equations is exactly the same as either the mean or the mode of the corresponding Gibbs sampling conditional distribution. For the assignments, z_i is sampled from a categorical distribution where the expected value for each z_{ik} is r_{ik}. The mean (and mode) of the Gaussian distribution for μ_k (Eqn. 2.47) coincides with the update for μ_k (Eqn. 2.34). The conditional distribution for s is the Gamma distribution (Eqn. 2.48), whose mode $(\tilde{a} - 1)/\tilde{b}$ is exactly the update for s (Eqn. 2.38). Finally, the conditional distribution for the weights is a Dirichlet distribution (Eqn. 2.51, with its mode given by the corresponding EM update (Eqn. 2.39).

One consequence of the similarity between the two algorithms is that they require a similar amount of computation per step. The computationally most intensive part is computing the soft assignments r_{ik}, which are needed in both cases. The additional computations needed for Gibbs sampling are insignificant in comparison.

Fig. 2.9 shows how Gibbs sampling can be used instead of EM for the example from Fig. 2.8. It suggests that one advantage of Gibbs sampling is that it converges faster than EM. Another advantage is that it explores the posterior distribution, instead of providing just a single estimate.

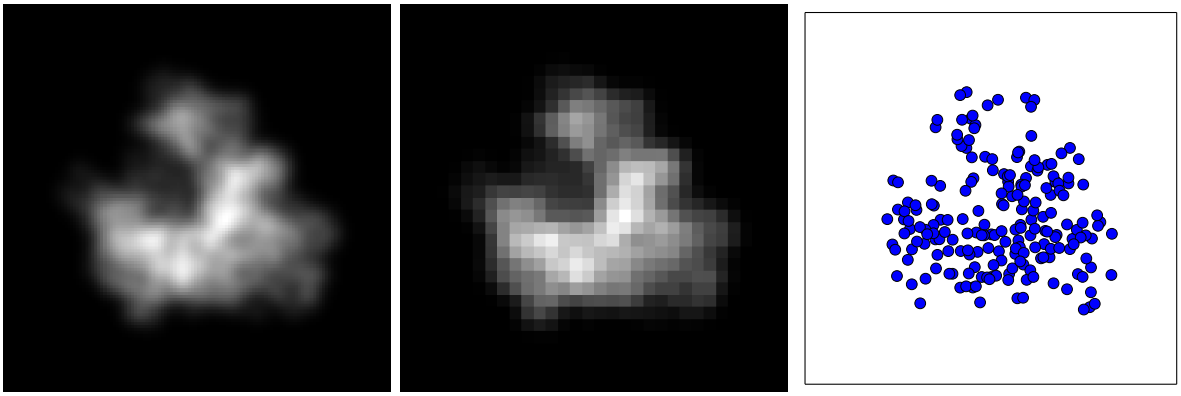

Figure 2.10: (*Left*) The Gibbs sampling and EM algorithms need to be modified to work with images as input for application to cryo-EM. (*Middle*) The same image at a lower resolution. (*Right*) The two algorithms introduced above use point clouds as input.

EM can be viewed as a special case of Gibbs sampling obtained by increasing the amount of data, which decreases the width of the Gibbs sampling conditional distributions. As shown above, the update equations for EM can (at least in this case) be derived from the conditional distributions of Gibbs sampling. For these reasons, we will focus on Gibbs sampling in the rest of this chapter.

2.3 Binned data

The cryo-EM data to which our reconstruction algorithm will be applied, will be non-negative, real-valued, two-dimensional images. The Gibbs sampling and EM algorithms presented above work with point clouds instead of images. Fig. 2.10 shows the difference between the type of data that we can currently work with (point clouds) and the type of data that we would like to work with (images). This section describes how to bridge this gap by discretising the images and modifying the above algorithms.

The strategy is to convert the images to point clouds. The first step is to discretise the real-valued images by scaling and rounding the values to the nearest integer. The scaling factor is chosen such that the sum of the integer values across all pixels in a given image approximately equals a predetermined constant N_0.

Consider an image with M pixels indexed by j. Let v_j denote the coordinates of the center of the jth pixel, and λ_j its intensity. Choose N_0 as the desired sum of discrete values across all pixels, also known as the desired number of counts. Then the discrete value y_j for each pixel is defined as

$$y_j := \left\lceil \frac{\lambda_j}{\sum_{j=1}^{M} \lambda_j} N_0 \right\rceil \tag{2.52}$$

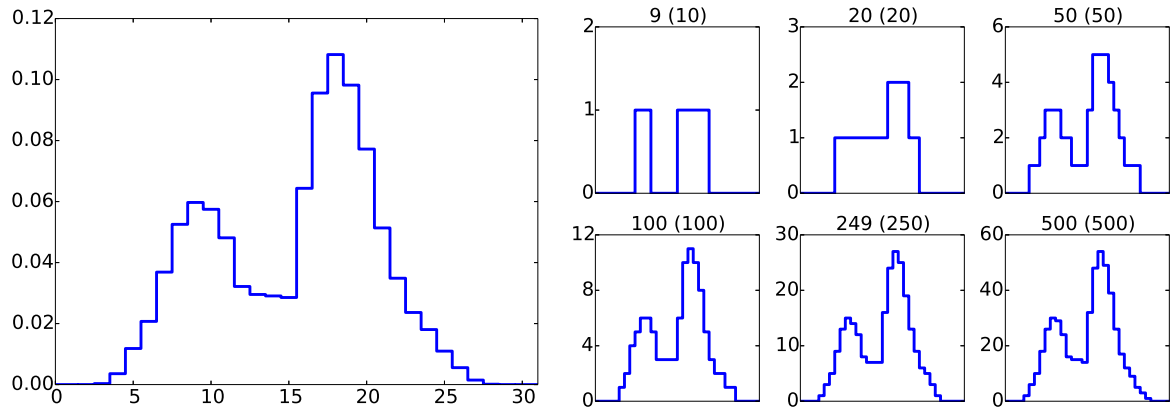

Figure 2.11: Discretising a one-dimensional real-valued function. (*Left*) A cross-section through the image in the middle of Fig. 2.10. (*Right*) Multiple discretised versions of the cross-section using different values for the desired number of counts N_0 (*in parenthesis*). The sum N (*outside the parenthesis*) of the discrete values is sometimes slightly different from N_0. For large values of N, the discretised version of the cross-section is a very good approximation.

where $[x]$ denotes x rounded to the nearest integer. Letting $N = \sum_{j=1}^{M} y_j$ denote the sum of the discrete values, we would expect that $N \approx N_0$ for relatively large N_0.

Fig. 2.11 shows the effect of discretisation on a cross-section of an image for different values of N_0. Although the description in this section uses images and pixels, the same procedure can be applied to one-dimensional data with intervals (Fig. 2.11), or three-dimensional data with voxels.

The next step is to view the discrete image as a point cloud, with y_j points located at the center of the jth pixel. In other words, the data is:

$$\mathcal{D} = \{\underbrace{v_1, v_1, \ldots, v_1}_{y_1}, \underbrace{v_2, v_2, \ldots, v_2}_{y_2}, v_3, \ldots, \underbrace{v_M, v_M, \ldots, v_M}_{y_M}\} \tag{2.53}$$

with y_j copies of v_j, and N points in total.

The data is now of the same type as earlier in the chapter, and the same Gibbs sampling and EM algorithms can be applied to fit a Gaussian mixture model to the data. But blindly applying those algorithms to the new point clouds would involve many unnecessary computations. For instance, when computing the soft assignments, every component should be evaluated at every data point. But data points can only be located at the center v_j of a pixel, and therefore every component needs to be evaluated at most once at every v_j, and not multiple times if $y_j > 1$. Avoiding the unnecessary evaluations is important, given that the number N of data points will usually be much larger than the number of pixels M. For implementing an efficient algorithm, it is useful to adapt the notation to the special structure of the data.

First, data points are indexed using two indices instead of one. The first index j denotes the pixel ($1 \leq j \leq M$), and the second index l ($1 \leq l \leq y_j$) is used to enumerate the y_j data points

at pixel j. The data is then

$$\mathcal{D} = \{x_{jl} \mid 1 \leq j \leq M, 1 \leq l \leq y_j\}, \tag{2.54}$$

where $x_{jl} = v_j$ for all j and l.

The soft assignments will be denoted by r_{jk} instead of r_{ik}, and their definition (Eqn. 2.35) is modified to denote that the components are evaluated at the grid points instead of the individual data points:

$$r_{jk} = \frac{w_k \mathcal{N}(v_j|\mu_k, s^{-1}I)}{\sum_l w_l \mathcal{N}(v_j|\mu_l, s^{-1}I)}. \tag{2.55}$$

For the hard assignments, z_{jl} denotes the component assigned to the data point x_{jl}, again using the 1-of-K notation. In other words, z_{jl} is a vector of length K, with entries $z_{jlk} \in \{0, 1\}$. A new vector of length K is introduced by summing over the data points at pixel j:

$$z_j = \sum_{l=1}^{y_j} z_{jl}. \tag{2.56}$$

The kth component z_{jk} of z_j is now any non-negative integer, and indicates the number of data points at the jth pixel assigned to the kth component:

$$z_{jk} = \sum_{l=1}^{y_j} z_{jlk}. \tag{2.57}$$

In the Gibbs sampling algorithm, z_j is now obtained via multiple samples from the same categorical distribution, i.e. via a single sample from a multinomial distribution. The corresponding conditional distribution (Eqn. 2.46) becomes

$$p(z_j|\mu, s, w, \mathcal{D}) \propto \prod_{k=1}^{K} r_{jk}^{z_{jk}} \tag{2.58}$$

where $\sum_{k=1}^{K} z_{jk} = y_j$.

The other conditional distributions for Gibbs sampling also undergo small modifications:

$$p(\mu_k|s, w, z, \mathcal{D}) = \mathcal{N}\left(\mu_k \Big| \frac{\sum_{j=1}^{M} z_{jk} y_j v_j}{N_k + r}, \frac{1}{s(N_k + r)} I\right) \tag{2.59}$$

$$p(s|\mu, w, z, \mathcal{D}) = \text{Gamma}(s|\tilde{a}, \tilde{b}) \tag{2.60}$$

$$2\tilde{a} = 2a + (M + K)d \tag{2.61}$$

$$2\tilde{b} = 2b + \sum_{j=1}^{M} \sum_{k=1}^{K} z_{jk} y_j \|v_j - \mu_k\|^2 + r \sum_{k=1}^{K} \|\mu_k\|^2 \tag{2.62}$$

$$p(w|\mu, s, z, \mathcal{D}) = \prod_{k=1}^{K} w_k^{N_k + \alpha_0 - 1}, \tag{2.63}$$

where now $N_k = \sum_{j=1}^{M} z_{jk} y_j$, but still $N = \sum_{k=1}^{K} N_k$.

The above changes to the Gibbs sampler are only for computational efficiency. They do not reflect a change in the underlying forward model. According to the forward model, the data points are still sampled from a Gaussian mixture model, and just happen to lie on a regular grid. Ideally, the forward model should be modified to include the effect of the grid.

We will now describe the required modifications to the forward model to take the grid into account, and consider what the corresponding changes to the Gibbs sampling and EM algorithms would be. Then we will argue that under certain reasonable assumptions, the effect of the grid can be ignored, and the original algorithms are recovered.

Intuitively, the modified forward model is to first sample a point cloud with N points, and then simply bin the data by letting y_j be the number of points in the jth pixel.

More precisely, the forward model is a Poisson point process, with its intensity function λ being the scaled probability density function of the mixture model

$$\lambda(x) = N \sum_{k=1}^{K} w_k \mathcal{N}(x|\mu_k, s^{-1}I). \tag{2.64}$$

The space $\mathbb{R}^d$ is partitioned into a square region R_j for each pixel j, and a single region R_0 to cover the area outside the grid. The observations are the number of points y_j in region R_j $(0 \leq j \leq M)$.

There are multiple equivalent ways of describing the point process. One way is to sample an integer N' from a Poisson distribution with rate N, and then sample N' points from the mixture, letting y_j be the number of points in region R_j. A second way is to also start by sampling N', but then sample the y_j's from a multinomial distribution with parameters $(N'; r_0, r_1, \ldots, r_M)$, where

$$r_j = \int_{R_j} \left(\sum_{k=1}^{K} w_k \mathcal{N}(x|\mu_k, s^{-1}I) \right) dx \tag{2.65}$$

And finally, this is also equivalent to sampling each y_j from a Poisson distribution with rate r_j:

$$p(y_j|\theta) = \frac{r_j^{y_j} e^{-r_j}}{y_j!}. \tag{2.66}$$

In the latter case we would define $N' = \sum_{j=0}^{M} y_j$, which also holds for the first two formulations. The forward model that has been adapted to take the grid into account is thus:

$$p(\mathcal{D}|\theta) = \prod_{j=1}^{M} p(y_j|\theta), \tag{2.67}$$

where $p(y_j|\theta)$ is defined in Eqn. 2.66, and r_j is defined in Eqn. 2.65.

There is a natural extension of the EM algorithm to find the MAP estimate for this forward model. The exact locations of the sampled points are unobserved, and are therefore considered

as latent variables, in addition to the component assignments that made up the latent variables thus far.

The resulting equations for the E-step and M-step become more complicated: they now involve expectations over regions R_j w.r.t. the mixture model, instead of simple function evaluations at pixel centers. The EM equations are derived by McLachlan and Jones (1988) for the one-dimensional case, and by Cadez et al. (2002) for higher dimensions. In one dimension, the solution can still be obtained analytically, in terms of the error function (erf). But in higher dimensions the solution requires the expectation integrals to be evaluated numerically, which is significantly more computationally intensive.

It would be possible to extend Gibbs sampling in a similar way, by augmenting the latent variables with the location of the sampled points. An extra sampling step would be required for sampling the locations of each of the y_j points from the mixture model restricted to region R_j. Aside from the fact that this is not a straighforward distribution to sample from, the computational complexity would then be linear in the number of sampled points, and not just in the number of pixels. In some of our applications, the number of sampled points will greatly exceed the number of pixels, making this approach computationally infeasible.

Fortunately, we can greatly simplify the equations with only a small loss in accuracy. This is based on two assumptions. The first is that the value of the Gaussian mixture model is relatively constant across a single pixel. This will be the case if the Gaussian components are large relative to the pixels. The second assumption is that the effective support of the mixture model is contained in the grid. Even though the support of the mixture model is unbounded, the part that lies outside the grid will have negligible mass as long as all the component means are well within the grid boundary.

Given these assumptions, the integrals over pixels can be approximated by simpling evaluating the integrands at the pixel centers. And the integrals over the complement of the grid are approximated by 0. Applying these approximations to the EM update equations in Cadez et al. (2002) yields the desired simplified version.

In conclusion, the extension of Gibbs sampling to binned data presented in this section is valid under the assumption that the mixture model components are large relative to the pixel size, and that the grids are sufficiently large to contain all the data.

The algorithm presented in this section can be used to fit three-dimensional mixture models to three-dimensional binned data. Although this does not address the cryo-EM reconstruction problem, it is nonetheless useful for another cryo-EM application: fitting mixture models to electron densities (Section 3.1). Such coarse-grained representations of the density can be used in different applications, two of which were described in Section 2.1.3.

2.4 Projected data

The algorithms introduced thus far are for fitting d-dimensional mixture models to d-dimensional data. But inferring initial models in cryo-EM requires three-dimensional mixture models to be estimated from two-dimensional data. This section describes an extension to the forward model by including a projection step to account for the lower dimensional data. The Gibbs sampling algorithm is extended to work with the new forward model.

2.4.1 Forward model

The model parameters θ will consist of the same d-dimensional mixture model as before, but in addition they will include the P directions along which the mixture is projected to create the $(d-1)$-dimensional data. As before, the mixture parameters consist of the d-dimensional means μ_k, the precision s and the weights w_k. The P projection directions are modelled as P transformations of the mixture model indexed by i. Each transformation is a d-dimensional affine transformation parametrised as a rotation $R_i \in \mathrm{SO}(d)$ followed by a translation $t_i \in \mathbb{R}^d$. The group of all rotations is the special orthogonal group:

$$\mathrm{SO}(d) = \{R \in \mathbb{R}^{d\times d} \mid RR^T = I, \det(R) = 1\}. \tag{2.68}$$

The data consists of P different $(d-1)$-dimensional histograms, i.e. binned data as described in Section 2.3. The forward model for generating one such histogram starts by sampling N_0 d-dimensional points from the mixture model. Every point x is transformed by a rotation and translation to obtain $R_i x + t_i$. This transformed point is then projected to $\mathbb{R}^{d-1}$ by discarding the last coordinate to obtain $P_o(R_i x + t_i)$, where P_o is a projection matrix given by

$$P_o = \begin{cases} \begin{bmatrix} 1 & 0 \end{bmatrix} & \text{if } d = 2, \\ \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \text{if } d = 3. \end{cases} \tag{2.69}$$

The projection which selects the last coordinate will be denoted by P_m. So for $d = 3$:

$$P_o = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.70}$$

$$P_m = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \tag{2.71}$$

and in general:

$$\begin{bmatrix} P_o \\ P_m \end{bmatrix} = I. \tag{2.72}$$

The subscripts o and m denote *observed* and *missing* respectively.

Fig. 2.12 shows an example of the forward model for generating a one-dimensional histogram from a two-dimensional mixture model. The figure also shows an equivalent way to view the

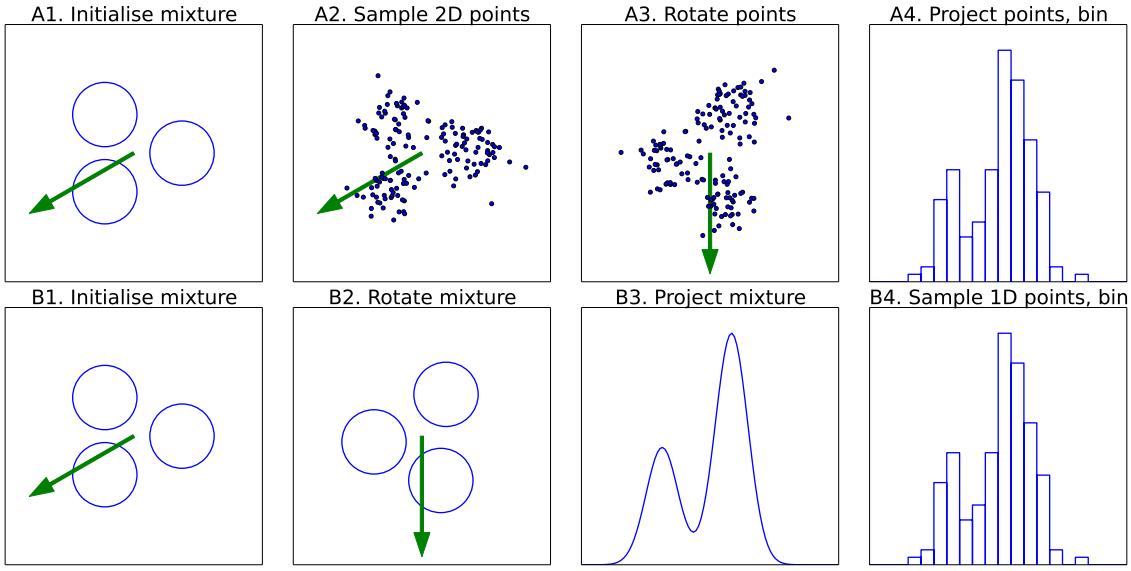

Figure 2.12: The forward model for generating a one-dimensional histogram from a two-dimensional mixture model. There are two equivalent ways of describing the forward model: (*Top row*) The points are first sampled, and then transformed and projected; (*Bottom row*) The mixture is first transformed and projected, and only then the points are sampled. The full forward model involves repeating this procedure P times with different rotations to generate P histograms.

forward model by reordering the steps: the transformations and projections can be applied to the mixture itself before sampling, instead of to the sampled points.

The advantage of using Gaussian mixture models is that they are closed under linear transformations. This means that applying a linear transformation to a mixture model yields another mixture model, only with different parameters. To apply the ith transformation, it is therefore only necessary to compute the new parameters.

Every mean μ_k is transformed to $R_i\mu_k + t_i$, and then projected to $P_o(R_i\mu_k + t_i)$. The N_0 points are then sampled from the projected $(d-1)$-dimensional mixture model, and binned to form a histogram. This process corresponds to the bottom row of Fig. 2.12.

The observed data is

$$\mathcal{D} = \{x^o_{ijl} \mid 1 \leq i \leq P,\ 1 \leq j \leq M,\ 1 \leq l \leq y_{ij}\}, \tag{2.73}$$

where M is the number of bins in each histogram. The superscript o denotes that this is the $(d-1)$-dimensional observed data. The missing component is denoted by x^m_{ijl}, and the full d-dimensional data point is

$$x_{ijl} = \begin{bmatrix} x^o_{ijl} \\ x^m_{ijl} \end{bmatrix}. \tag{2.74}$$

The forward model is

$$p(\mathcal{D}|\theta) = \prod_{ijl} p(x^o_{ijl}|\theta) \tag{2.75}$$

$$p(x^o_{ijl}|\theta) = \sum_k w_k \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I). \tag{2.76}$$

The forward model for mixture models (Eqn. 2.76) is much simpler than for the grid-based representations used by the algorithms reviewed in Section 1.4. Applying rotations and projections to grid-based density maps requires grid interpolation schemes and low-pass filters to avoid aliasing. The computational complexity scales with the number of voxels. In contrast, transforming and projecting a mixture model does not require any approximations, and the complexity scales with the number of mixture components.

The blobs introduced by Marabini et al. (1998) (Section 1.3) are an improvement over grid-based representations. One reason is that they are also spherically symmetric, and thus look the same when projected in any direction. But they are still grid bound, which means that many more of them are needed compared to the number of Gaussian mixture model components.

To simplify the notation for the remainder of this chapter, a single product symbol will be used to denote the product over multiple variables. Thus:

$$\prod_{ijlk} = \prod_{i=1}^{P} \prod_{j=1}^{M} \prod_{l=1}^{y_{ij}} \prod_{k=1}^{K}. \tag{2.77}$$

The range of the variables is always as given in Eqn. 2.77.

The prior for Gaussian mixture models is augmented with a prior on the rotations and the translations. The uniform prior is used for the rotations. It is defined as the unique distribution invariant under the action of the group of rotations on itself (the Haar measure). For $d = 3$, the space of rotations is SO(3). Miles (1965) derives expressions for the uniform distribution on SO(3) for different coordinate systems.

For the translations we use a Gaussian prior centered at the origin.

The new prior is

$$p(\theta) = p(\mu|s)p(s)p(w)p(R)p(t), \tag{2.78}$$

where $p(\mu|s)$, $p(s)$ and $p(w)$ are the same as before (Eqns. 2.22, 2.23 and 2.24), and

$$p(R) \propto 1 \tag{2.79}$$

$$p(t) = \prod_{i=1}^{P} p(t_i) = \prod_{i=1}^{P} \mathcal{N}(t_i|0, r_t^{-1}). \tag{2.80}$$

2.4.2 Algorithm

The idea for extending the EM and Gibbs sampling algorithms is to treat the missing components as latent variables. Thus the latent variables are

$$\mathcal{Z} = \{z, x^m\}, \tag{2.81}$$

where

$$z = \{z_{ijl} \mid 1 \leq i \leq P,\ 1 \leq j \leq M,\ 1 \leq l \leq y_{ij}\} \tag{2.82}$$

are the component assignments as before, and

$$x^m = \{x^m_{ijl} \mid 1 \leq i \leq P,\ 1 \leq j \leq M,\ 1 \leq l \leq y_{ij}\} \tag{2.83}$$

are one-dimensional missing components.

To derive the corresponding Gibbs sampling algorithm, the first step is to have the full (or extended) data likelihood:

$$p(\mathcal{D}, \mathcal{Z}|\theta) = \prod_{ijlk} \left[w_k \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)\right]^{z_{ijlk}}. \tag{2.84}$$

Using the prior (Eqn. 2.78) and the extended likelihood (Eqn. 2.84), the conditional distributions for Gibbs sampling can be computed (see Appendix C):

$$p(z_{ijl}|\mu, s, w, R, t, \mathcal{D}) = \prod_k \left[w_k \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I)\right]^{z_{ijlk}} \tag{2.85}$$

$$p(x^m_{ijl}|z_{ijl}, \mu, s, R, t, \mathcal{D}) = \prod_k \mathcal{N}(x^m_{ijl}|P_m(R_i\mu_k + t_i), s^{-1}I)^{z_{ijlk}} \tag{2.86}$$

$$p(w|x^m, z, \mu, s, R, t, \mathcal{D}) \propto \prod_k w_k^{N_k + \alpha_0 - 1} \tag{2.87}$$

$$p(\mu_k|x^m, z, s, w, R, t, \mathcal{D}) = \mathcal{N}(\mu_k|\frac{1}{N_k + r}\sum_{ijl} z_{ijlk} R_i^T(x_{ijl} - t_i), \frac{1}{s(N_k + r)}I) \tag{2.88}$$

$$p(s|x^m, z, \mu, R, t, \mathcal{D}) = \text{Gamma}(s|\tilde{\alpha}, \tilde{\beta}) \tag{2.89}$$

$$2\tilde{\alpha} = 2a + dM + K \tag{2.90}$$

$$2\tilde{\beta} = 2b + \sum_{ijkl} z_{ijkl}\|x_{ijl} - (R_i\mu_k + t_i)\|^2 + r\sum_k \|\mu_k\|^2 \tag{2.91}$$

$$p(R_i|t_i, x^m, \mu, s, \mathcal{D}) \propto \exp\left[\text{tr}(A_i^T R_i)\right] \tag{2.92}$$

$$A_i = s\sum_{jlk} z_{ijlk}(x_{ijl} - t_i)\mu_k^T \tag{2.93}$$

$$p(t_i|x^m, z, \mu, s, R, \mathcal{D}) = \mathcal{N}(\frac{1}{N_i}\sum_{jlk} z_{ijlk}(x_{ijl} - R_i\mu_k), \frac{1}{N_i s}I) \tag{2.94}$$

$$N_k = \sum_{ijl} z_{ijlk}. \tag{2.95}$$

Almost all the conditional distributions are standard distributions with implemented sampling algorithms that are widely available. The exception is the conditional distribution for the rotations (Eqn. 2.92). It is known as the matrix Fisher distribution, and can be sampled from using the algorithm by Habeck (2009).

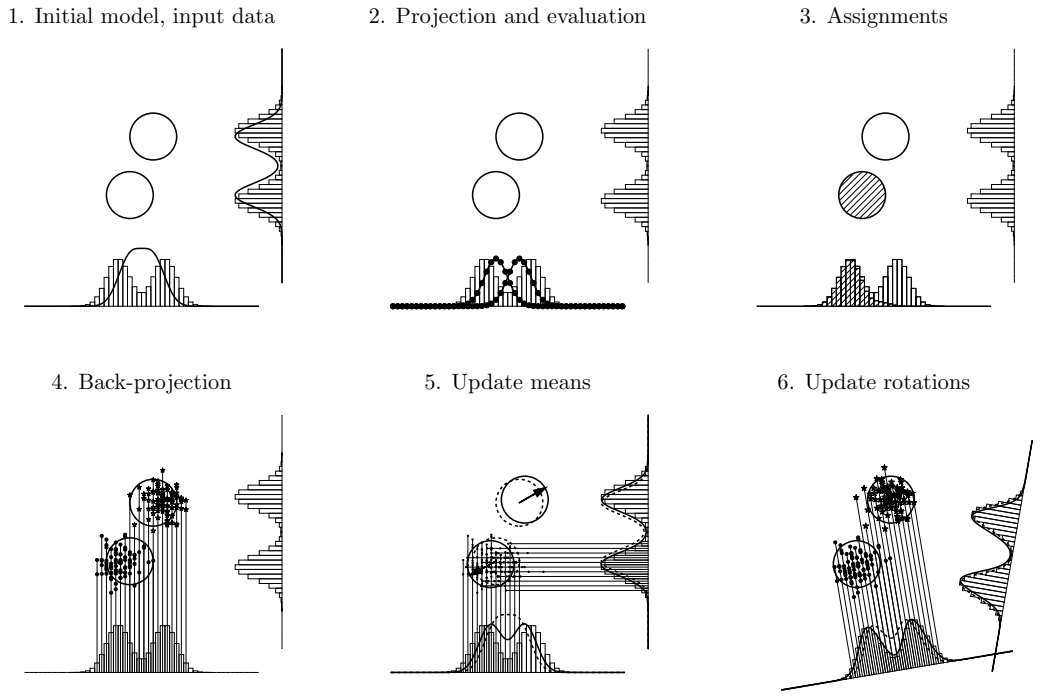

Figure 2.13: Simple two-dimensional reconstruction example to explain a single Gibbs sampling iteration. (*Solid lines*) One-dimensional projections of the current model. (*Dashed lines*) One-dimensional projections of the previous model. Initially (*1*), the one-dimensional projections differ significantly from the one-dimensional data. They improve after updating the component means (*5*), and once again after updating the rotations (*6*). The precision and weights are kept fixed in this example. The final projections approximate the data quite well.

Fig. 2.13 explains the above Gibbs sampling algorithm using a simple example that reconstructs a two-dimensional mixture model from one-dimensional histograms. Sampling the assignments (Eqn. 2.85) corresponds to panels *2* and *3* of the figure. The parameters of the multinomial distribution are computed by projecting and evaluating the Gaussian mixture model (panel *2*). Individual points are assigned to components by sampling from the multinomial (panel *3*). Sampling the missing components (Eqn. 2.86) can be interpreted as back-projecting the one-dimensional points to two dimensions (panel *4*), where the missing y-coordinate of the point is sampled to be near the y-coordinate of the mixture component to which it was assigned. The y-coordinate is the mean of the Gaussian in Eqn. 2.86. This back-projection step is repeated for each projection direction.

Having sampled the assignments and the missing components, the points now have the same dimension as the mixture. This is similar to the setting in Section 2.2.3. The mixture parameters are sampled in a similar way (panel *5*, Eqns. 2.87 to 2.89). The points from all the projection directions are combined in computing the mixture parameters. In Fig. 2.13 only the component

means are updated.

The last step is to sample the rotations and translations (panel *6*, Eqns. 2.92 and 2.94). Sampling the rotations can be viewed as keeping the mixture components fixed, and rotating the corresponding back-projected point cloud about the origin to better match the mixture components. Sampling the translations is similar. Only the rotation updates are shown in Fig. 2.13.

Although the derivation is completely different, there are several similarities between the Gibbs sampling algorithm presented here and the algorithms reviewed in Section 1.4, especially the algorithms based on projection matching.

To start with, the first part of the assignment sampling step is to project the current mixture model in P directions, and evaluate it at each grid point to obtain P different projection images. This corresponds to the projection step of projection matching, where the current estimate of the density map is also projected in many different directions to create projection images. The difference is that projection matching uses more projection directions. The Gibbs sampling algorithm corresponds to the later projection matching steps, where the projection directions are concentrated in a neighbourhood of the current estimate of the image orientation, instead of covering the entire space of orientations.

The next step for projection matching is to update the projection direction for each image. This corresponds to the last part of the Gibbs sampler, where the rotations and translations are sampled. For the Gibbs sampler it is not necessary to compute multiple projections in a neighbourhood of the current one. From the current projection the Gibbs sampler estimates how the likelihood will change for small changes in the rotation. The Gibbs sampler therefore corresponds to a local rotation optimiser. It will be extended in the next chapter to do global rotation sampling, which will make the algorithm more robust.

The remaining Gibbs sampling steps correspond to the reconstruction step in projection matching. Sampling the assignments and the missing components has the effect of back-projecting the two-dimensional points to a three dimensional point cloud, with points from every image. This is similar to the back-projection algorithm described in Section 1.3. The sampling steps that update the mixture parameters (mean, precision and weights) effectively convert the point cloud back into a mixture model.

Part of the Gibbs sampling approach presented here is related to work done by Ghahramani and Jordan (1995). They used EM to estimate Gaussian mixture models given data with some missing components. They also modelled the missing components as missing data in the EM framework. Compared to our approach, they did not have rotations or binned data, and used EM instead of Gibbs sampling. Each data point could have multiple missing components.

The Gibbs sampling algorithm will be applied to cryo-EM data in the following two chapters.

Chapter 3

Experiments with simulated data

The Gibbs sampling algorithms developed in Chapter 2 will now be applied to simulated data. There are two cases to consider: fitting a mixture model to a three-dimensional electron density map (Section 3.1), and inferring a mixture model from two-dimensional class averages (Section 3.2 onwards).

For the second case there are two classes of model parameters to estimate: the mixture model and the image orientations. Before applying the full algorithm for estimating both (Section 3.4), we will first consider the two simpler settings where one of the two is known.

3.1 Fitting mixtures to electron densities

The Gibbs sampling algorithm described in Section 2.3 can be used to approximate an electron density by a mixture model. In Section 2.3 the algorithm was used to fit two-dimensional mixture models to images. Electron densities are usually represented by voxels on a three-dimensional regular grid. In other words, they are the three-dimensional equivalent of images, and the same Gibbs sampling algorithm can be applied.

The same algorithm can also be used to approximate an atomic model of a structure by a mixture model, by first converting the atomic model to an electron density.

Since the data is three-dimensional, this is not a reconstruction algorithm. It is nevertheless useful to be able to convert density maps to mixture models. Two applications of the mixture model representation were discussed in Section 2.1.3.

Another reason for focusing on this algorithm is that it contains many of the components of the full reconstruction algorithm, but is simpler. Therefore, studying it helps in understanding the effect of the various parameters on the result.

3.1.1 Generating simulated data

The algorithm will be tested on density maps of different structures at multiple resolutions. The atomic model for each of the following structures is available from the Protein Data Bank

(PDB):

- RNA polymerase II (PDB: 1I3Q)
- GroEL (PDB: 1OEL)
- 50S ribosome subunit (PDB: 1VOR)

An atomic model can be converted to a density map by placing a Gaussian at each atom position with the atomic number as its weight, and evaluating the resulting mixture of Gaussians on a regular three-dimensional grid. This procedure is available in CHIMERA as the MOLMAP command.

The resulting density map is influenced by two parameters: the size (i.e. standard deviation) of the Gaussians, and the voxel size of the regular grid.

Specifying the size of the Gaussians amounts to choosing the resolution of the density map. The default approach used by CHIMERA is to assign the size σ as

$$\sigma = \frac{\text{res}}{\pi\sqrt{2}} \tag{3.1}$$

$$\approx 0.225 \times \text{res} \tag{3.2}$$

where **res** is the desired resolution in angstrom (Å). The motivation for the choice of coefficient in Eqn. 3.2 is that the Fourier transform of the Gaussian distribution falls to $1/e$ of its maximum value at wavenumber 1/res. The CHIMERA documentation suggests several other plausible choices for the coefficient, ranging from 0.187 to 0.425. We will use the coefficient from Eqn. 3.2. The resolutions of the density maps will range from 10 Å to 25 Å (Fig. 3.1).

The second important parameter is the voxel size. By the Nyquist criterion, a voxel size of d Å is sufficient for faithfully representing densities that have no information at resolutions beyond $2d$ Å. Although density maps created from Gaussian mixture models are not band-limited, this motivates using a voxel size no larger than 5 Å, given that the simulated density maps will have a resolution of at most 10 Å.

The voxel size also has implications for the mixture model fitting algorithm. A larger voxel size will generally require fewer counts, and the algorithm will run faster. These effects will be explored in detail below.

Resolution is usually expressed as a spatial frequency, with units 1/Å. We follow the common practice in cryo-EM of referring to the inverse of the spatial frequency as the resolution, with units Å. This is why a resolution of 10 Å is *higher* than a resolution of 15 Å.

Having obtained a real-valued density map from an atomic model, the next step is to convert the real values to integer values. This is done using the approach described in Section 2.3 to scale and round the values.

For the conversion to an integer-valued density, we have to select the desired sum N_0 of the discrete values (or *counts*) across all voxels. Below in Section 3.1.4 we investigate the effect of N_0 on the result, and give guidelines for choosing it.

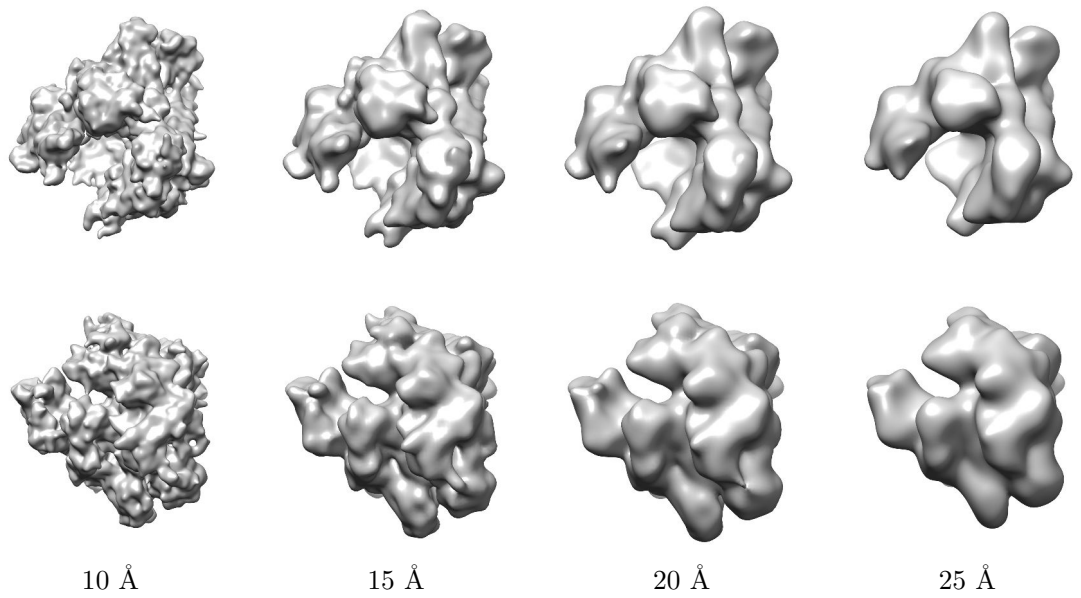

Figure 3.1: Simulated density maps of RNA polymerase II for testing the Gibbs sampling algorithm. The maps were created from the same atomic structure at four different resolutions using CHIMERA. Each *column* shows two views of the same map.

3.1.2 Examples

The first structure used to test the algorithm is a simulated density map of RNA polymerase II (Pol II) at 15 Å, with a voxel size of 2 Å. The number of counts is set to $N_0 = 10^6$.

The prior distribution over the model parameters was described in Section 2.2.1. There are four hyperparameters to be specified. We will test the sensitivity of the reconstruction result to each of these parameters below in Section 3.1.5; here we only state the default values used in this experiment.

The hyperparameter r for the prior on the means is set to the precision of a single isotropic Gaussian fitted to the data. The prior on the precision has hyperparameters $a = 10$ and $b = 1000$, corresponding to a mean precision of $1/10^2$, i.e. a component size of 10 Å. Finally, the hyperparameter for the weight prior is $\alpha_0 = 10$.

The results of the algorithm for Pol II are shown in Fig. 3.2. The algorithm is initialised by sampling a 500-component mixture model from the prior. This is followed by several Gibbs sampling steps. The only algorithmic parameter is the number of steps, which should be high enough for the Gibbs sampler to converge. Convergence can be monitored by looking at the log-posterior, which is computed after each step.

The mixture improves a lot during the first few steps, but then converges more slowly. This is typical behaviour for an EM-like algorithm with overlapping components. After about 250 steps the Gibbs sampler has converged to a stationary distribution, where the components continue

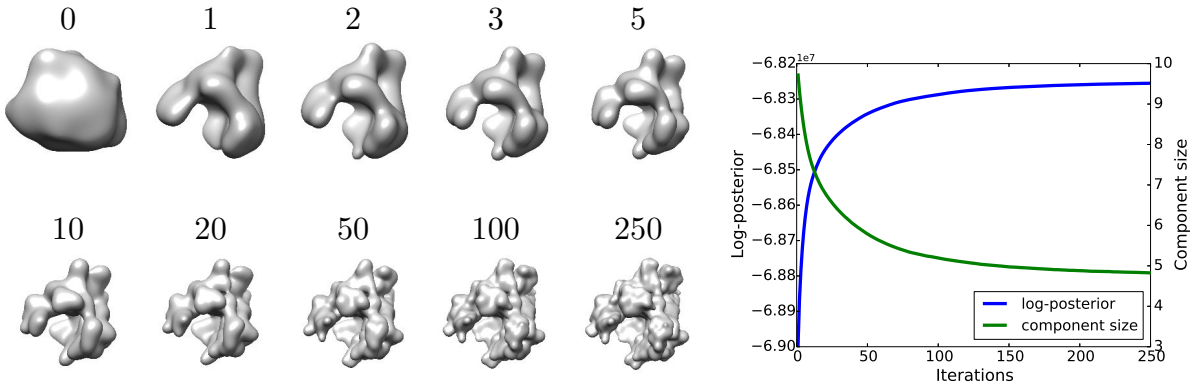

Figure 3.2: Fitting a 500-component mixture to a 15 Å density map of RNA polymerase II. Intermediate mixture models (*left*) converge after about 250 steps, as monitored by the log-posterior (*right*). The initial model, sampled from the prior, has a large component size, which quickly decreases and converges to about 5 Å.

to wiggle slightly from one step to the next. For now, the result of the algorithm is the final mixture generated by the Gibbs sampler, although it is possible to combine multiple samples to obtain a better result (Section 3.2.2).

The resulting Pol II mixture looks very similar to the reference density (Fig. 3.3). One way of quantifying the similarity is to compute the cross-correlation coefficient ρ between the reference and the reconstruction. Let x be the vector of reference density values, and let y be the vector of the inferred mixture model evaluated on the same three-dimensional grid as the reference density. Then the normalised cross-correlation coefficient ρ is defined as

$$\rho = \frac{\langle x, y \rangle}{\|x\| \|y\|} = \frac{\sum_i x_i y_i}{\sqrt{\sum_i x_i^2 \sum_i y_i^2}}. \tag{3.3}$$

From the Cauchy-Schwarz inequality and the non-negativity of x and y it follows that $0 \leq \rho \leq 1$.

In addition to Pol II, the algorithm was applied to GroEL and the 50S ribosome. Fig. 3.3 compares the results to the reference models. In all three cases the mixture model was fit to a reference density at 15 Å, but for comparing the resulting mixture to the reference it is more appropriate to use a lower resolution reference model (20 Å, or 25 Å for the ribosome).

In all three cases the algorithm converges to a result that is very similar to the reference, as measured by the cross-correlation. Repeating the algorithm multiple times with the same input gives very similar results, i.e. it is very robust. The running time of the algorithm can range from a few seconds to a few minutes, depending on the input data and various model and algorithmic parameters.

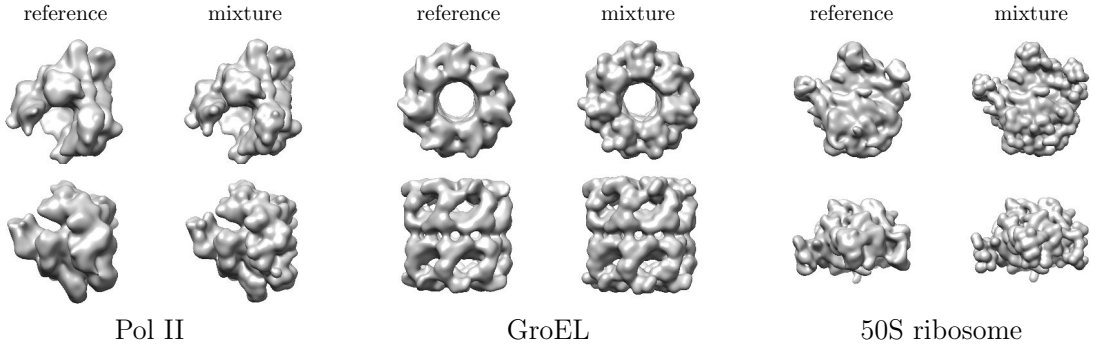

Figure 3.3: Fitting mixture models to different structures. The Pol II result is the 500-component mixture from Fig. 3.2, while the GroEL and 50S ribosome mixtures both have 1000 components. The references structures have resolutions 20 Å, 20 Å and 25 Å respectively. The cross-correlations between the references and mixture models are 0.9786, 0.989 and 0.9904 respectively.

3.1.3 The number of components

Before running the algorithm, the user must specify the number of mixture components, K. This determines the total number of model parameters: $3K$ for the means, 1 for the precision and $K-1$ for the weights, a total of $4K$ parameters. Therefore, increasing K increases the total number of model parameters, which allows the mixture to approximate density maps to a higher resolution (Figs. 2.4 and 2.5).

It is only necessary to specify the number of components K, not the component size σ. The component size is estimated by the algorithm, via the precision $s = 1/\sigma^2$. Contrast this with the approach used by Nogales-Cadenas and Jonic (2013), where the user specifies σ, and the algorithm estimates K.

There is a strong relation between K and σ: as the number of components increases, the size of each component decreases. If we imagine a component with size σ as a hard sphere with radius σ, then the total volume covered by K tightly packed spheres would be proportional to the total volume V of the spheres:

$$V = \frac{4\pi}{3} K\sigma^3. \tag{3.4}$$

Although the Gaussian components are not hard spheres, and the electron density is not binary, this does suggest the following relation between K and σ:

$$K\sigma^3 = c, \tag{3.5}$$

where c is a constant that depends on the density map. This relation is confirmed experimentally in Fig. 3.4 for three different structures.

In case the user would like to specify σ instead of K, Eqn. 3.5 can be used to estimate the appropriate value of K. For example, one approach would be to run the Gibbs sampling

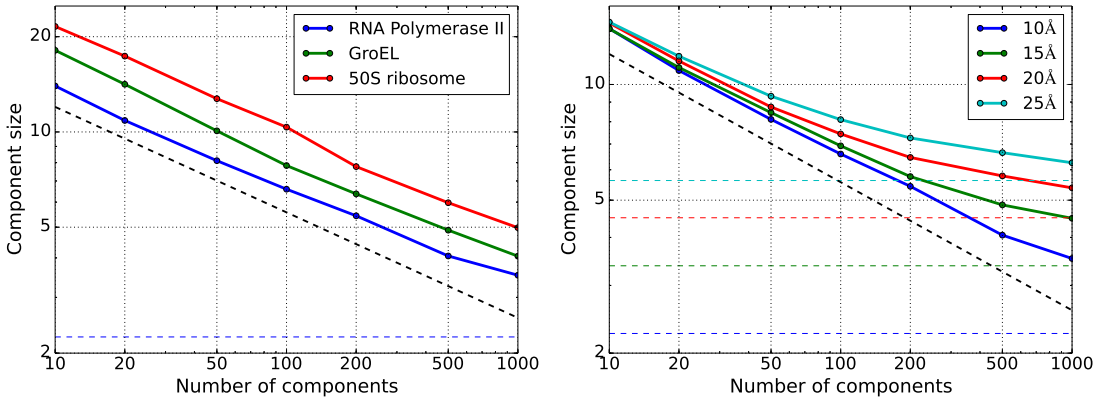

Figure 3.4: Relation between number of components and component size. The number of components is chosen in advance, while the component size is taken from the final model once the Gibbs sampler has converged. In all cases the reference grid spacing is 5 Å and the number of counts is 10^5. (*Left*) The resolution of the reference volume is 10 Å, and the results are compared for three different structures. (*Right*) A single structure (Pol II) is used, but the resolution of the reference volume is varied from 10 to 25 Å. The *dashed black line* shows the relation $K\sigma^3 = c$ for constant c. The results show that if the number of components is not too high, and the resolution of the reference volume is high enough, then this relation determines the component size σ for a given K. The constant c depends on the structure. The *dashed horizontal lines* denote the component size used to create the corresponding reference model.

algorithm for some value of K, note the final value of σ, and use Eqn. 3.5 to estimate a new value of K that would give the desired value of σ.

As shown in Fig. 3.4, the relation given by Eqn. 3.5 breaks down for large values of K. As K increases, the components do not become as small as predicted by Eqn 3.5. Instead, their size is influenced by the resolution of the density map: given an equal number of components, a higher resolution density map will have smaller components than a lower resolution one (Fig. 3.4). The component size adapts to the resolution of the density map.

A possible explanation for the deviation from Eqn. 3.5 is that if K was set to the number of atoms in the original atomic model, then the optimal component size would be the size of the Gaussians used to convert the atomic model to a density map. And therefore the component size converges to this value as the number of components increases (Fig. 3.4).

There are various guidelines for choosing K. As shown above, larger values of K should be used for larger structures, and to represent structures at higher resolutions. Furthermore, due to the strong relation between K and σ, it is possible to specify σ instead of K.

The choice also depends on the purpose for which the mixture model representation is to be used. For example, for subunit fitting (Section 2.1.3) a very low resolution representation might be sufficient, whereas higher resolution representations are needed for estimating normal modes

(Section 2.1.3).

Nogales-Cadenas and Jonic (2013) recommend choosing σ to be similar to the voxel size, but not larger than some threshold (9999 in their case). They also recommend downsampling the volume before applying their coarse-graining algorithm. The same guidelines can be applied in our case.

Note that the algorithm is fast enough to fit multiple mixture models with different values of K, and choose the one most suitable for the application.

3.1.4 Voxel size and number of counts

Before running the algorithm to fit the mixture, the real-valued data must be discretised, as explained in Section 2.3. The level of discretisation is determined by the number of counts N_0. If N_0 is too low, there will be too few discretisation levels to faithfully represent the data (Fig. 2.11). In contrast, setting N_0 too high will increase the running time of the algorithm without improving the accuracy of the result.

For choosing N_0, the relevant quantity is the average number of counts per voxel. As shown in Fig. 3.5 for Pol II, there should be at least one count per voxel on average. The voxels should also not be too large. Note that because the structure is not rectangular, there will be many voxels with zero counts, increasing the count density in the remaining voxels. Also note that the results given in Fig. 3.5 are for $K = 50$; larger K might require a somewhat larger count density for optimal results.

3.1.5 Effect of prior hyperparameters

In general, the Gibbs sampling results are very robust to variations in the four hyperparameters of the prior. Here we investigate the range over which the hyperparameters can vary.

Figs. 3.6, 3.7 and 3.8 show the effect of the hyperparameters for the priors on the means, precision, and weights respectively.

From the figures it can be seen that each of the hyperparameters can be varied over several orders of magnitude without influencing the results. It is only when we set them to extreme values, that their influence starts to outweigh that of the data.

We typically use the following default values: for the prior on the mean we fit a single isotropic Gaussian to the data, and use its precision as the value for the hyperparameter r.

For the precision s we choose $\alpha = 10$ and $\beta = 1000$, such that the mean precision corresponds to a component size of 10 Å. The distribution for $\alpha = 10$ at the bottom in Fig. 3.7 shows that these parameters allow for a wide range of optimal component sizes, as confirmed by the contour plot at the top left.

Finally, for the weights we choose $\alpha_0 = 10$. It is interesting that increasing α_0 does not have a strong effect on the result (Fig. 3.8). In the limit, this means that we could also set all the weights equal to $1/K$. This would reduce the number of model parameters without leading to significantly worse results.

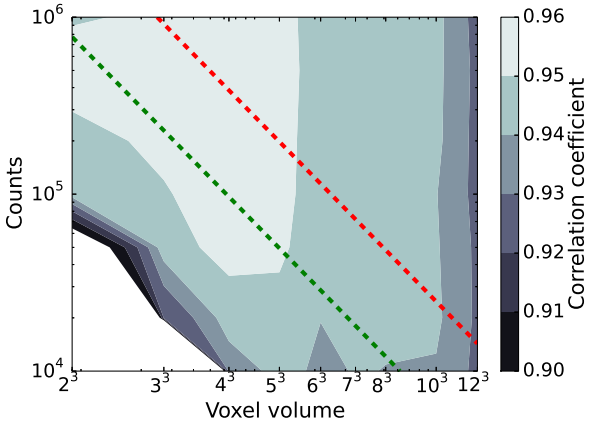

Figure 3.5: Effect of voxel size and number of counts. For each combination of the voxel size and the total number of counts, we fit a 50-component mixture to a 15 Å density map of RNA polymerase II. The quality of each mixture is measured by its cross-correlation relative to the reference density map on a grid with voxel size 2 Å. (*Lower left corner*) For small voxels with few counts, the cross-correlation drops below 0.9. Each *dashed line* denotes a constant number of counts per voxel: 0.5 (*green*) and 2 (*red*). The figure shows that to represent Pol II with 50 components the average number of counts per voxel should be at least 0.5 to ensure a good representation.

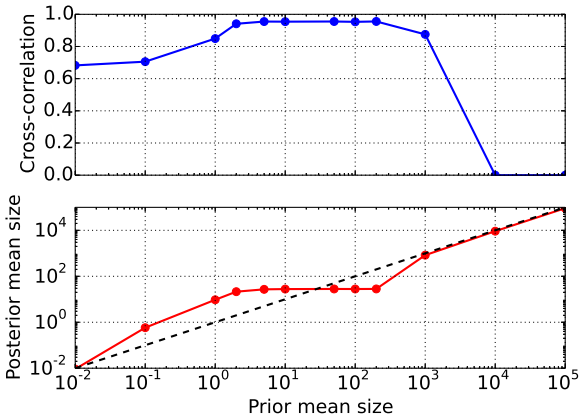

Figure 3.6: Varying the prior on the means. The mean prior is a normal distribution centered at the origin. Its hyperparameter determines the width of the distribution. The Gibbs sampling algorithm was repeated with the same Pol II density map for a wide range of possible values of the width. (*Top*) The cross-correlation of the result relative to the reference map is high and relatively constant from a width of 2 Å up to 200 Å. (*Bottom*) The component width of the final mixture model is also relatively constant across the same range, showing that the data dominates the prior. Outside this range, the prior becomes more important, ensuring that the final component width is similar to the prior component width (*dashed black line* is the identity function).

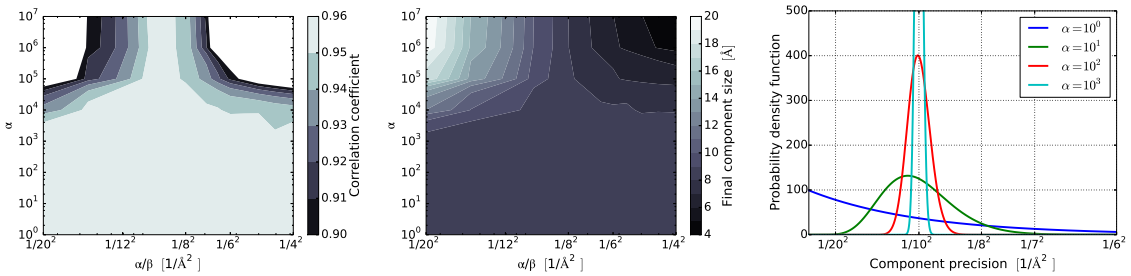

Figure 3.7: Varying the prior on the component size. The Gamma distribution prior on the component precision is parametrised by its mean α/β, which determines the typical component size, and α, which determines how narrow the distribution is. (*Right*) Examples of the Gamma distribution with typical component size $\alpha/\beta = 10\,\text{Å}$ and different values of α. The Gibbs sampling algorithm was repeated for many combinations of these parameters, using the same Pol II density map as in Fig. 3.6. (*Left*) The quality of the results measured by the cross-correlation with a reference structure. (*Middle*) The final component sizes. The figure shows that for a wide range of α values (from 10^0 to 10^3) the results are good and do not depend on the prior. But increasing α even further forces the final component size to be closer to the value determined by the prior. The figure in the *middle* shows that for $\alpha = 10^7$, the final component size is almost exclusively determined by the prior.

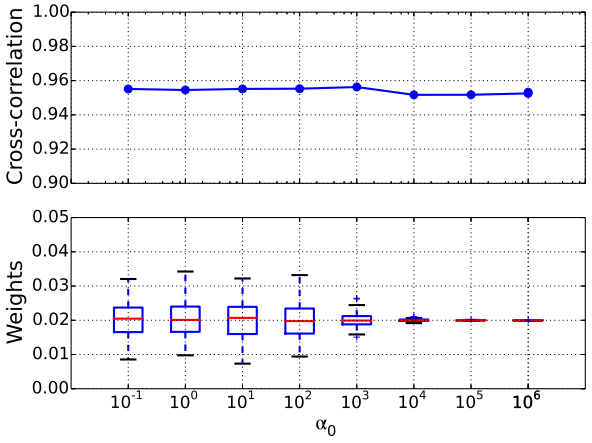

Figure 3.8: Varying the prior on the weights. The prior on the weights is a Dirichlet distribution with a single parameter, α_0. Good results are obtained for a very wide range of this parameter. For high values of the parameter, the final weights all become roughly equal to each other. This suggests that we could also fix the weights to be equal from the start.

3.2 Inferring mixtures from class averages with known orientations

For the rest of this chapter, the data will be two-dimensional class averages instead of three-dimensional density maps. In this section, the orientations of the class averages are assumed to be known.

Although the data is different, the goal is the same as in the previous section: to infer a three-dimensional mixture model representation of the electron density. There are many similarities with the algorithm from the previous section, which will be pointed out along the way.

3.2.1 Simulated class averages

The input data to the algorithm is non-negative class averages, simulated from the same three structures used in Section 3.1: Pol II, GroEL and the 50S ribosome. For each structure, the atomic model from the PDB is converted to a real-valued density map as before. The density map is projected along different directions to obtain two-dimensional real-valued images. The images are then discretised following the approach from Section 2.3.

As in Section 3.1, when creating the density maps we have to specify their resolution and voxel size. In addition, for the class averages we now have to choose the pixel size, and the number of counts N_0 per image.

Each image orientation is chosen randomly. The rotation R_i describing the ith image orientation is sampled uniformly from SO(3). The translations are also assumed to be known. By translating the images if necessary, we can assume that $t_i = 0$ without loss of generality.

The model parameters and the prior distribution are exactly the same as in Section 3.1. The effect of the prior hyperparameters on the result of the algorithm is very similar to Section 3.1, and the same default choices for the hyperparameters are used here.

Given the simulated class averages, the mixture model parameters are estimated using the Gibbs sampling algorithm from Section 2.4. The conditional distributions for the rotations and translations are removed from the algorithm, and the rotations and translations are kept fixed throughout the algorithm.

As in Section 3.1, the algorithm is initialised by sampling a mixture model from the prior. This is followed by several Gibbs sampling steps, and the log-posterior is monitored for convergence. The only parameter is the number of sampling steps.

3.2.2 Example

The algorithm was tested on Pol II. The input data (Fig. 3.9, *left*) consists of 25 images of size 50×50, with a pixel size of 4 Å. The images were created as described above, using an electron density at 15 Å, and with $N_0 = 10000$ counts per image.

The progress of the algorithm is shown in Fig. 3.9. After about 500 Gibbs sampling steps the algorithm has converged. The projections of the final mixture agree well with the original

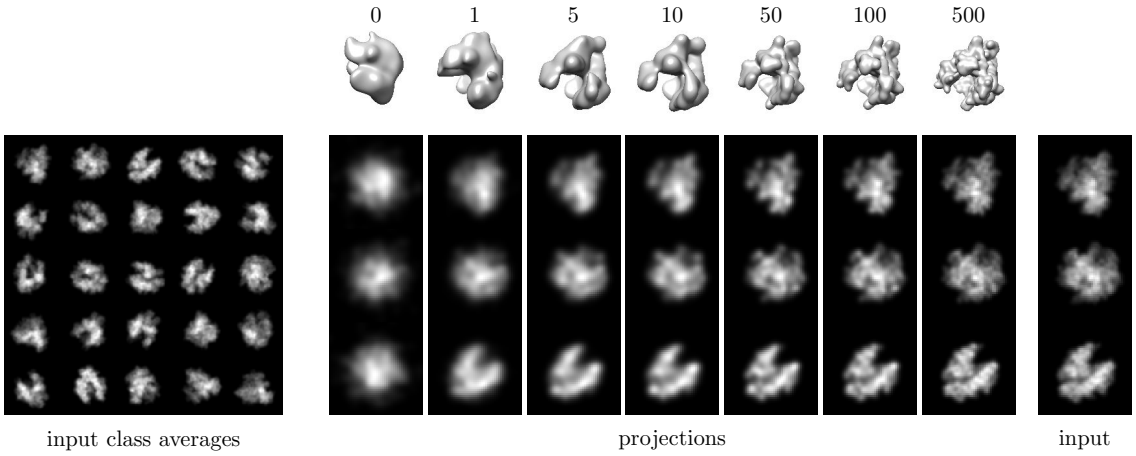

Figure 3.9: Pol II reconstruction from class averages with known orientations. A 500-component mixture model is inferred from 25 class averages. Along the top are the mixture models obtained at various stages of the Gibbs sampler. Below each three-dimensional model is its projections along the three projection directions corresponding to the first three class average orientations. At the right are the first three input class averages. Comparing the last two columns shows that the final projections match the input data very well.

input data.

Thus far in this chapter, the result of the algorithm was taken to be the final mixture model generated by the Gibbs sampler. This does not, however, take advantage of all the information provided by the algorithm. Once the Gibbs sampler has converged, the mixture models generated by subsequent sampling steps are all samples from the posterior. The simplest way in which to combine these mixture models, is to compute their mean. This is done by evaluating each mixture model on the same grid, and computing the mean of the three-dimensional arrays.

To estimate the posterior mean for Pol II, every 50th mixture model generated by the Gibbs sampler is collected, starting from the 1000th mixture. After collecting 50 mixture models, their mean is computed. Fig. 3.10 compares the resulting mean to a reference density at 20 Å. The posterior mean is more similar to the reference density than any of the individual posterior samples, as shown by the FSC curves.

FSC curves (Frank 2006, Chapter 3) are an alternative to cross-correlations for evaluating the similarity between two density maps. The Fourier Shell Correlation (FSC) curve between two density maps is obtained by computing the normalised cross-correlation coefficient between corresponding shells in Fourier space. The point where the curve crosses the 0.5 threshold is often used as a proxy for the resolution.

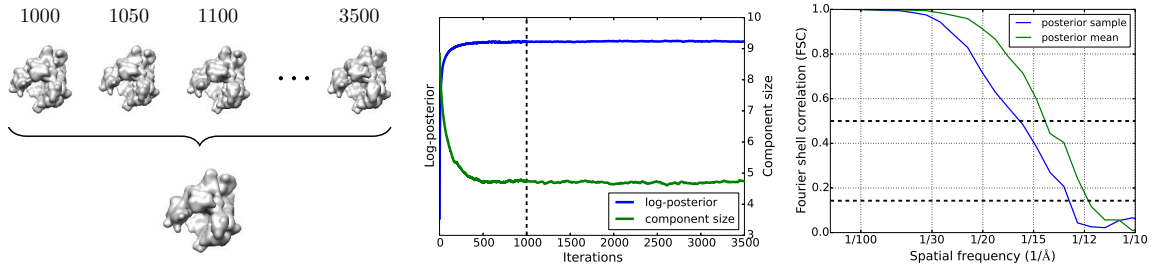

Figure 3.10: Computing the posterior mean for Pol II. (*Left*) Multiple mixture models produced by the Gibbs sampler are averaged together to yield an estimate of the posterior mean. (*Middle*) The first 1000 Gibbs sampling steps are discarded as belonging to the burn-in period. (*Right*) Replacing the final mixture model with the estimate of the posterior mean leads to a significant improvement in the FSC curve.

3.2.3 Comparison to direct Fourier inversion

As described in Section 1.3, there are several widely used algorithms for solving the known-orientation reconstruction problem. These algorithms differ from the one proposed here in several ways. To be concrete, we will compare our algorithm to direct Fourier inversion.

The first difference is that the result of our algorithm is a mixture model which can be evaluated on an arbitrarily fine three-dimensional grid. This can be useful when the pixel-size of the input images is quite large. In contrast, direct Fourier inversion produces three-dimensional maps whose voxel sizes are the same as the pixel size of the input images.

Another difference is that our input images need to be non-negative, and we do not use a Gaussian noise model, as is typically assumed by other algorithms. For non-negative class averages with very little noise, this does not make such a big difference, but for raw particle images, a Gaussian noise model would be more suitable. As a result, our reconstruction algorithm cannot be used directly on raw particle images. Later in the thesis (Chapter 5) we will introduce an alternative approach to address these shortcomings.

Our algorithm appears to be just as robust as other algorithms. For those other algorithms which are also iterative, the cost function is convex, and thus they always converge to the same solution. In our case the negative log-likelihood is not convex. Nevertheless, in our experience the Gibbs sampler never gets stuck in local optima.

Fig. 3.11 shows the results of comparing our reconstruction algorithm to direct Fourier inversion on the same 25 ribosome class averages. Our algorithm gives better results at low resolutions (below 20 Å), but the direct Fourier inversion results are better at higher resolutions. A possible reason for the improvement at low resolutions is that both the reference density and our reconstruction are non-negative, while the Fourier reconstruction allows negative values. This can be clearly seen in the intensity histograms in Fig. 3.11. Our reconstruction looks similar to the reference at all thresholds, while for the Fourier reconstruction the threshold must

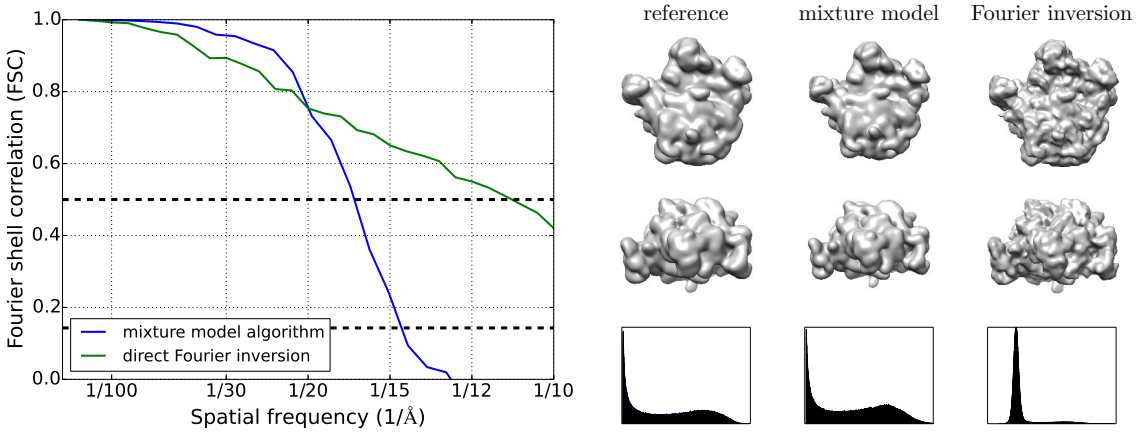

Figure 3.11: The mixture model reconstruction algorithm compared to direct Fourier inversion using the 50S ribosome subunit. Both reconstructions use the same 25 images, and are compared to the same 25 Å reference. The final mixture model has 2000 components. The mixture model result is more similar to the reference at frequencies below 20 Å (*left*), and has a more similar histogram of intensity values (*bottom right*).

be chosen carefully. At higher resolutions (above 20 Å) our algorithm performs worse, because the components are too large to respresent high-resolution detail.

3.2.4 The number of class averages

With only 5 ribosome class averages instead of 25, the mixture model algorithm performs significantly better than direct Fourier inversion at low frequencies (Fig. 3.12).

A possible reason for this is that the mixture model algorithm has far fewer parameters, and is not able to represent high frequency information. This helps to prevent over-fitting on limited data.

3.2.5 Missing cone

Up to now, the rotations for the simulated images were sampled uniformly from SO(3). But in practice, the image orientations are often not distributed uniformly.

One example is structures that adopt a preferred orientation when imaged using cryo-EM. These could be elongated structures such as the 26S proteasome, which tend to lie horizontally in the ice layer. As a result, only a few of the images depict the view along the main axis of the structure; most are side views.

Another example is Random Conical Tilt (RCT) (Frank 2006; Radermacher 1988). According to this data collection scheme, the ice layer containing the particles is imaged twice: first tilted at a random angle between 0° and an upper bound such as 60°, and then in the usual horizontal position. Particle images in the horizontal micrograph are aligned and clustered. For

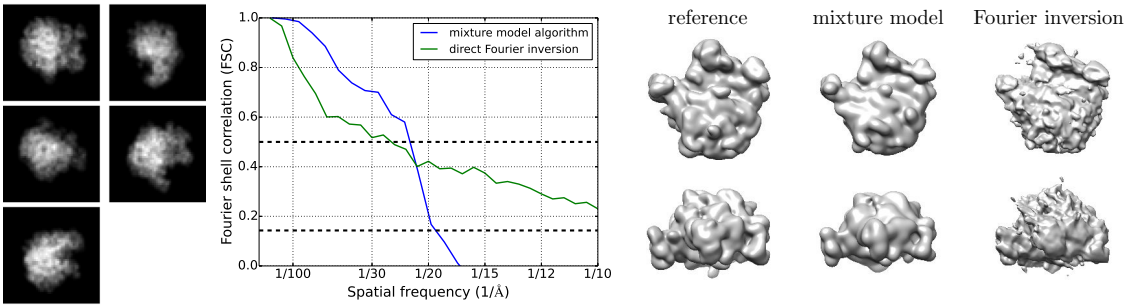

Figure 3.12: Reconstructions of the 50S ribosome subunit using less input data. (*Left*) The input consists of only 5 class averages. (*Middle*) The mixture model algorithm compares favourably to direct Fourier inversion at low frequencies. (*Right*) Compared to the reference, the mixture model looks better than the Fourier inversion result.

every cluster of untilted images, the correponding group of tilted images are used for a reconstruction from known orientations. Their relative orientations are derived from the alignment parameters of the untilted images, and the tilt angles.

Because the sample cannot be tilted more than say 60°, many views of the structure are not represented. It follows from the projection-slice theorem (Section 1.3) that the missing views correspond to a missing cone in Fourier space. The missing cone is known to have an adverse effect on the resolution of the reconstructed density map (Frank 2006).

The robustness to a missing cone of the mixture model reconstruction algorithm is tested using sets of 25 GroEL images simulated for different missing cone sizes (Fig. 3.13). The size of the missing cone ranges from 0° to 60°, corresponding to a maximum tilt angle ranging from 90° down to 30°. The untilted view is a top view of GroEL. A missing cone angle of 0° means that all views are possible, i.e. that there is no missing cone.

The mixture model reconstruction algorithm is compared to direct Fourier inversion (Fig. 3.13). As in the previous experiments, the mixture model algorithm gives better results at low resolutions (below 20 Å in this case), while the direct Fourier inversion result better represents the high frequency data. Furthermore, the quality of the mixture model reconstruction does not deteriorate as fast with a larger missing cone.

Both the last two experiments show that the mixture model algorithm is more robust to having only limited data.

3.3 Inferring class average orientations

The previous section showed how to infer a mixture model given images with known orientations. This section considers the opposite situation: given a mixture model, how to infer the orientation of each image.

As explained in Chapter 2, the ith image orientation is modeled as a rotation R_i. The rotation

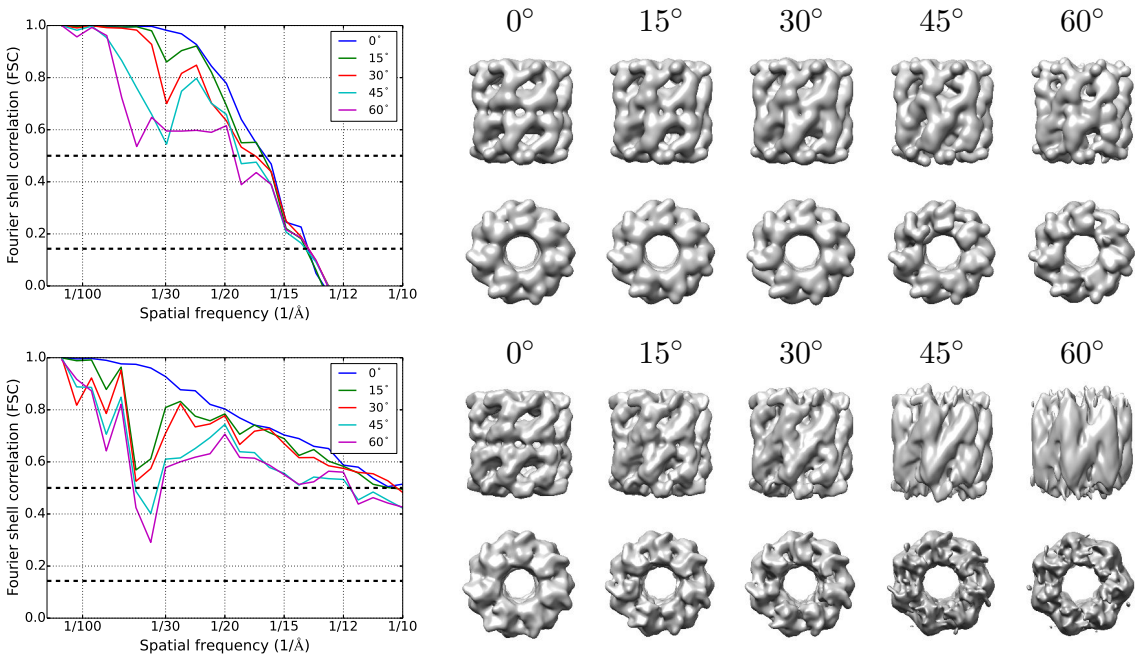

Figure 3.13: Comparing the effect of the missing cone on the mixture model reconstruction algorithm and on direct Fourier inversion. (*Top*) Results of the mixture model reconstruction algorithm. (*Bottom*) Results using direct Fourier inversion. (*Right*) Each *angle* denotes the size of the missing cone in Fourier space. A missing cone of 30° corresponds to a maximum tilt angle of 60°. (*Left*) Each reconstruction is compared to a reference GroEL structure at 15 Å by computing an FSC curve. Comparing the FSC curves shows that at low resolution (below 20 Å) the mixture model algorithm performs better, while at higher resolution direct Fourier inversion is better. At low resolutions, larger missing cones have a stronger effect on the direct Fourier inversion result than the mixture model result. Visually, the mixture model results also appear to be less affected by the missing cone.

describes both the direction in which the mixture model is projected (two parameters), and the in-plane rotation of the projected image (one parameter). Other reconstruction algorithms often treat the projection direction and the in-plane rotation separately.

As in the previous section, the translations t_i are fixed: $t_i = 0$ for each i.

Estimating the rotations forms part of many of the reconstruction algorithms described in Chapter 1. For instance, one of the two steps of every projection matching iteration is to update the orientation parameters for each image based on the current estimate of the density map. This can be either a global or a local update. During the first projection matching iterations, the density map is projected along a grid covering all possible directions, to estimate the globally best rotation for each image. During later iterations, the global grid is replaced by a local one surrounding the current best estimate of the orientation parameters for each image.

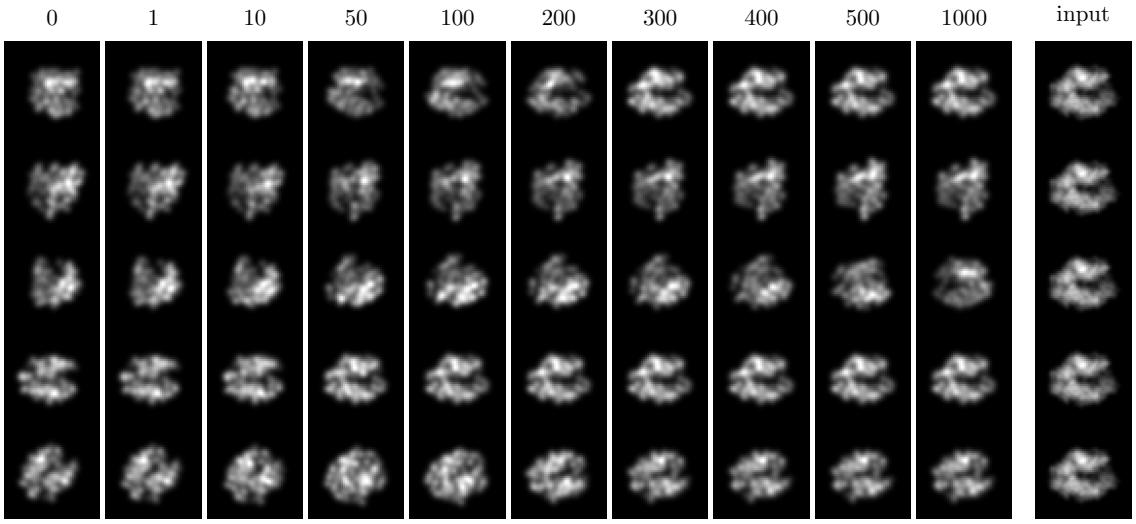

Figure 3.14: Estimating Pol II image orientation using local Gibbs sampling. The goal is to estimate the orientation of a single image (*right-most column*). The initial images (*left-most column*) are projections of the mixture model after applying different random rotations. Even after 1000 Gibbs sampling steps (*second-last column*), only two of the rotations have converged to the correct solution. The others are stuck in local optima. This shows that the local Gibbs sampler can only find the correct rotation if it is initialised with a nearby rotation.

Similarly, the rotation sampler for mixture models can be either local or global. The local rotation sampler is derived from the Gibbs sampler for the full model (Section 2.4) in the same way that the mixture model sampler (Section 3.2) was derived: by fixing the known parameters and removing their updating steps from the algorithm.

The parameters that remain are the assignments z, the missing components x^m, and the rotations R. These are sampled in turn from the same conditional distributions as before (Eqns. 2.85, 2.86, 2.92).

Each sampling iteration of z, x^m and R forms one step of the Gibbs sampler for rotations. The algorithm is initialised by sampling each rotation from the uniform prior over SO(3). Although the Gibbs sampler does not converge to a single rotation, it usually only changes by small amounts once the burn-in period has passed. The algorithm can therefore be evaluated by testing if the last sampled rotation is in the vicinity of the true rotation or not.

Testing the algorithm with Pol II images shows that it does not always converge to the correct rotation (Fig. 3.14). It converges to a local optimum, which coincides with the global optimum only if the initial random rotation happens to be near the true rotation.

One reason why the Gibbs sampler often gets stuck in a local optimum is the following: After sampling the assignments z, each count x_{ijl} in the image has been assigned to a specific component, given by z_{ijl}. The conditional distribution for R_i depends on this assignment, i.e.

it assumes that it is correct. It therefore assigns a very low probability to rotations where the count would be very far away from the projection of its assigned component. But if the current estimate of the rotation R_i is far from the true rotation, then many of the sampled assignments are also wrong, and consequently the true rotation may have a very low likelihood under the conditional distribution for R_i.

This effect of the Gibbs sampler getting stuck in local optima becomes more severe as the number of components increases, and the component size decreases.

To motivate a different approach to rotation sampling, note that as before, the Gibbs sampler is being used to sample from the extended posterior $p(\theta, \mathcal{Z}|\mathcal{D})$, where the model parameters θ are now just the rotations. As an alternative, consider the posterior that does not involve any latent variables:

$$p(\theta|\mathcal{D}) \propto p(\mathcal{D}|R)p(R) \tag{3.6}$$

$$\propto \prod_{i=1}^{P} p(R_i|\mathcal{D}), \tag{3.7}$$

where

$$p(R_i|\mathcal{D}) \propto \prod_{j=1}^{M} \prod_{l=1}^{y_i} \sum_{k=1}^{K} w_k \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I). \tag{3.8}$$

The posterior for R_i is not a standard distribution from which samples can easily be drawn; this was the motivation for introducing latent variables in Section 1.4.1. But we can nevertheless sample from Eqn. 3.8 by approximating it with a discrete distribution. The approximation is formed by sampling a large number N_R of rotations R_m uniformly from SO(3), say $N_R = 10000$. The values of the posterior of R_i at these rotations form the weights of the discrete approximation:

$$p(R_i|\theta, \mathcal{D}) \approx \sum_{m=1}^{N_R} w_m \delta(R_i - R_m), \tag{3.9}$$

where the weights are

$$w_m \propto p(R_m|\theta, \mathcal{D}) \tag{3.10}$$

$$\sum_{m=1}^{N_R} w_m = 1. \tag{3.11}$$

To sample a rotation R_i from Eqn. 3.9, just sample m from the categorical distribution with weights w_m, and let $R_i = R_m$ be the desired sample.

This approach using the posterior distribution for R_i (Eqn. 3.8) and its discrete approximation (Eqn. 3.9), will be referred to as the global rotation sampler. It is very similar to the global rotation update step in projection matching.

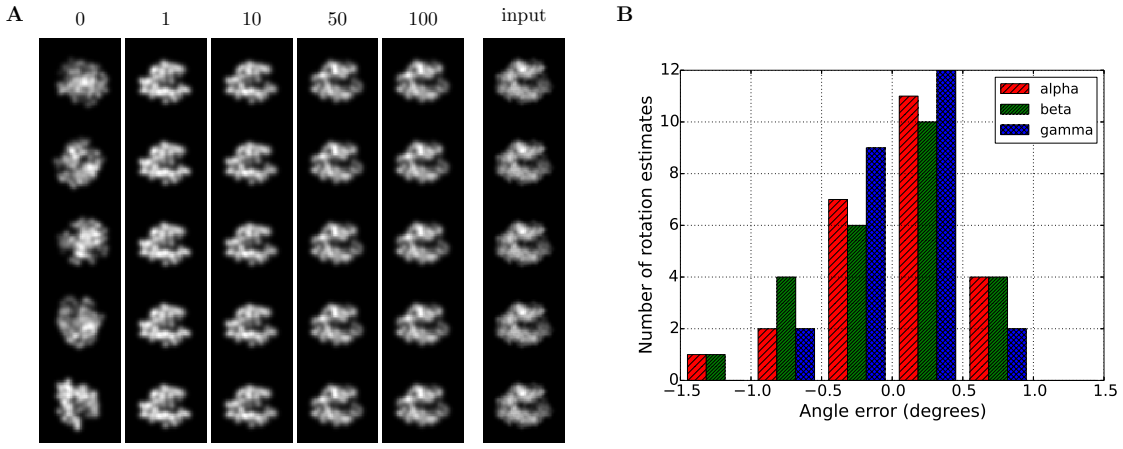

Figure 3.15: Combining global and local rotation sampling to estimate the Pol II image orientation from Fig. 3.14. (*A*) The initial rotations are again sampled randomly (*first column*). The first step (*second column*) is a global rotation sampling step, which finds an rotation close to the true rotation. During the remaining local Gibbs sampling steps, each rotation converges to the vicinity of the true rotation. (*B*) For each of the 25 images, we compare the Euler angles of the true rotation and the estimated rotation. All rotation estimates are very accurate, with most of the angular errors $< 1°$.

One difference between the local and global approaches to rotation sampling is that global rotation sampling is much more computationally intensive. Fortunately, one global rotation sampling step is usually enough to find a rotation in the vicinity of the true rotation. The proposed algorithm for estimating rotations is to start with a single global rotation sampling step, followed by multiple local rotation sampling steps to converge to the solution.

Fig. 3.15 shows that this algorithm successfully converges to the correct rotation in all tested examples.

3.4 Inferring mixtures from class averages with unknown orientations

This section introduces the complete algorithm for inferring initial models from class averages. The algorithm estimates both the mixture model and the image orientations simultaneously by combining the algorithms from the previous two sections.

3.4.1 Initial and refinement stages

The algorithm is divided into two parts: an initial stage, and a refinement stage. A very low resolution structure using only a few mixture components is constructed during the initial stage, and then refined with more components during the refinement stage.

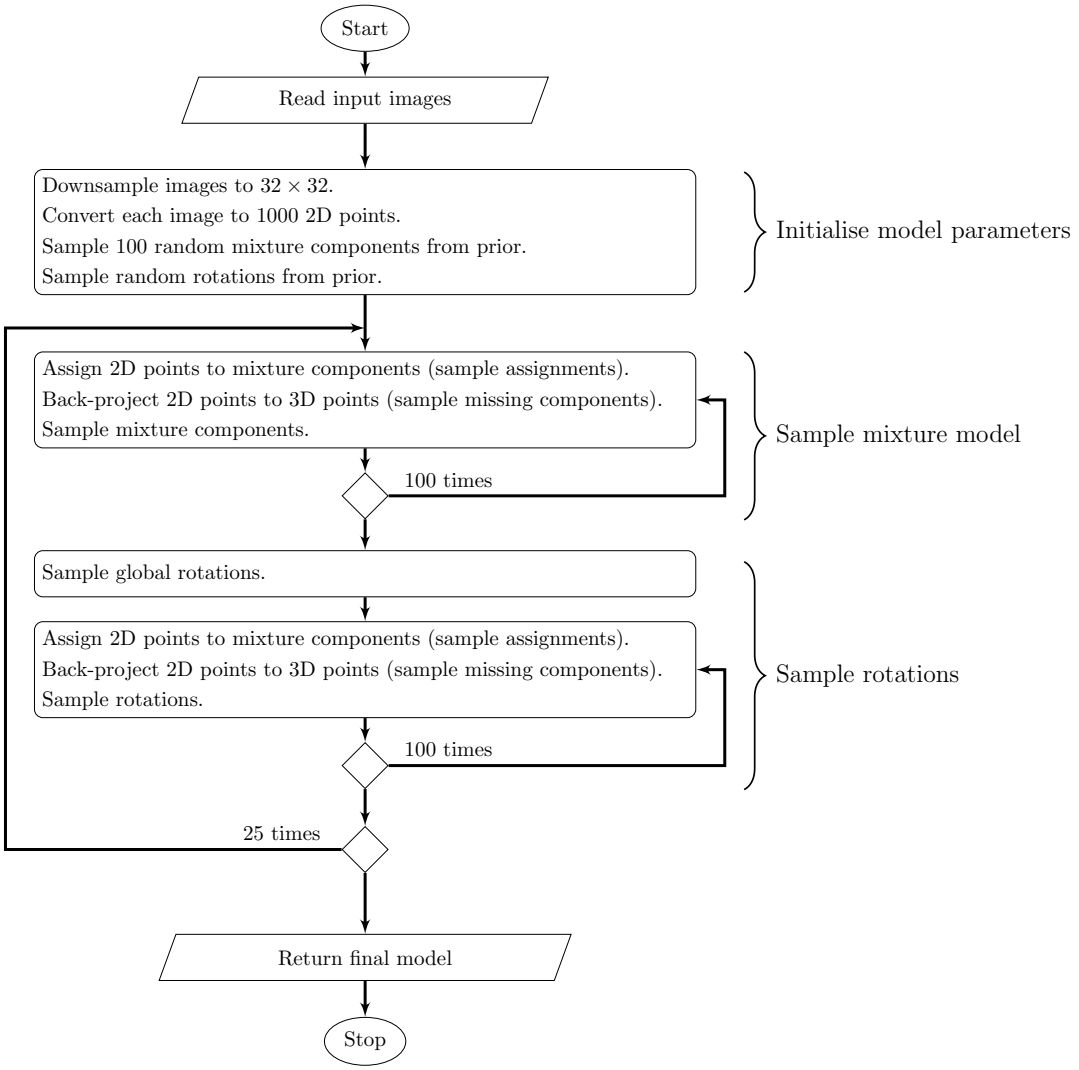

Figure 3.16: Initial stage of complete algorithm for inferring initial model from class averages.

The initial stage (Fig. 3.16) alternates between mixture model updates and rotation updates. Each of these updates consists of several Gibbs sampling steps during which the other model parameters are kept fixed.

The algorithm is initialised by sampling both a mixture model and rotations from the prior. The translations are set to zero, and held fixed during the initial stage.

After initialisation, the rotations are fixed, and the mixture model parameters are updated with multiple Gibbs sampling steps. This is the same algorithm as in Section 3.2, although fewer Gibbs sampling steps are used.

The mixture model parameters are then fixed, and the rotations are sampled using the algorithm proprosed at the end of Section 3.3. That is, a single global rotation sampling step is followed by multiple local rotation sampling steps.

The mixture model parameters are then sampled again with the rotations fixed, and so on. The complete algorithm typically converges within a few iterations of alternating between mixture model parameters and rotations.

The important parameters that have to be specified by the user for the initial stage are the number of components K, the size of each image, the number of counts N_0 per image, and the number of rotations N_R for the global rotation sampling step. The effect of these parameters on the results will be investigated below. Due to the small number of components, it is not necessary to use high resolution images, or many counts per image. The default values are $K = 100$ or $K = 200$, images of size 32×32 or 50×50, 1000 counts per image, and $N_R = 1000$ rotations for global rotation sampling.

To improve its robustness, the algorithm is repeated several times with the same input data, and the resulting models are ranked by their log-posterior probability. The final model which has the highest log-posterior probability is used as the input to the refinement stage. This model is usually the one with the best estimates of the rotations, as in the example below in Section 3.4.3.

The purpose of the initial stage is to obtain a very low resolution model whose rotations are close to the true rotations. These rotations are used to initialise the refinement stage, along with a random mixture model sampled from the prior. The number of mixture components is increased to improve the resolution of the model.

During the refinement stage, the mixture components, rotations and translations are sampled using the Gibbs sampler introduced in Section 2.4. Due to the larger number of components used (between 500 and 2000), the images should have a higher resolution, and more counts per image. Once the Gibbs sampler has converged, multiple mixture models are averaged as in Section 3.2.2 to estimate the posterior mean, which is the final result of the algorithm.

3.4.2 Example

Fig. 3.17 shows the application of the full algorithm to the 50S ribosome subunit. The input data of 25 class averages are simulated exactly as in Section 3.2, by converting an atomic model to a density map, and projecting it in random directions.

During the initial stage (Fig. 3.17 *top of A, B*), the class averages are downsampled to 32×32, they are discretised to have only 1000 counts per image, and the mixture model has only 100 components. During the refinement stage (Fig. 3.17 *bottom of A, C*), the number of components is increased to 2000. The final result, the posterior mean, is very similar to the reference model at 25 Å, with a cross-correlation of 0.990.

Fig. 3.17 *B, C* shows how the log-posterior changes. During the initial stage, the log-posterior makes a series of jumps during the first five steps. There are two such jumps during each step, corresponding to the mixture model update and the rotation update respectively. About seven steps into the initial stage, the log-posterior stabilises, except for a small dip directly after each global rotation update. This is because only $N_R = 1000$ rotations are considered during

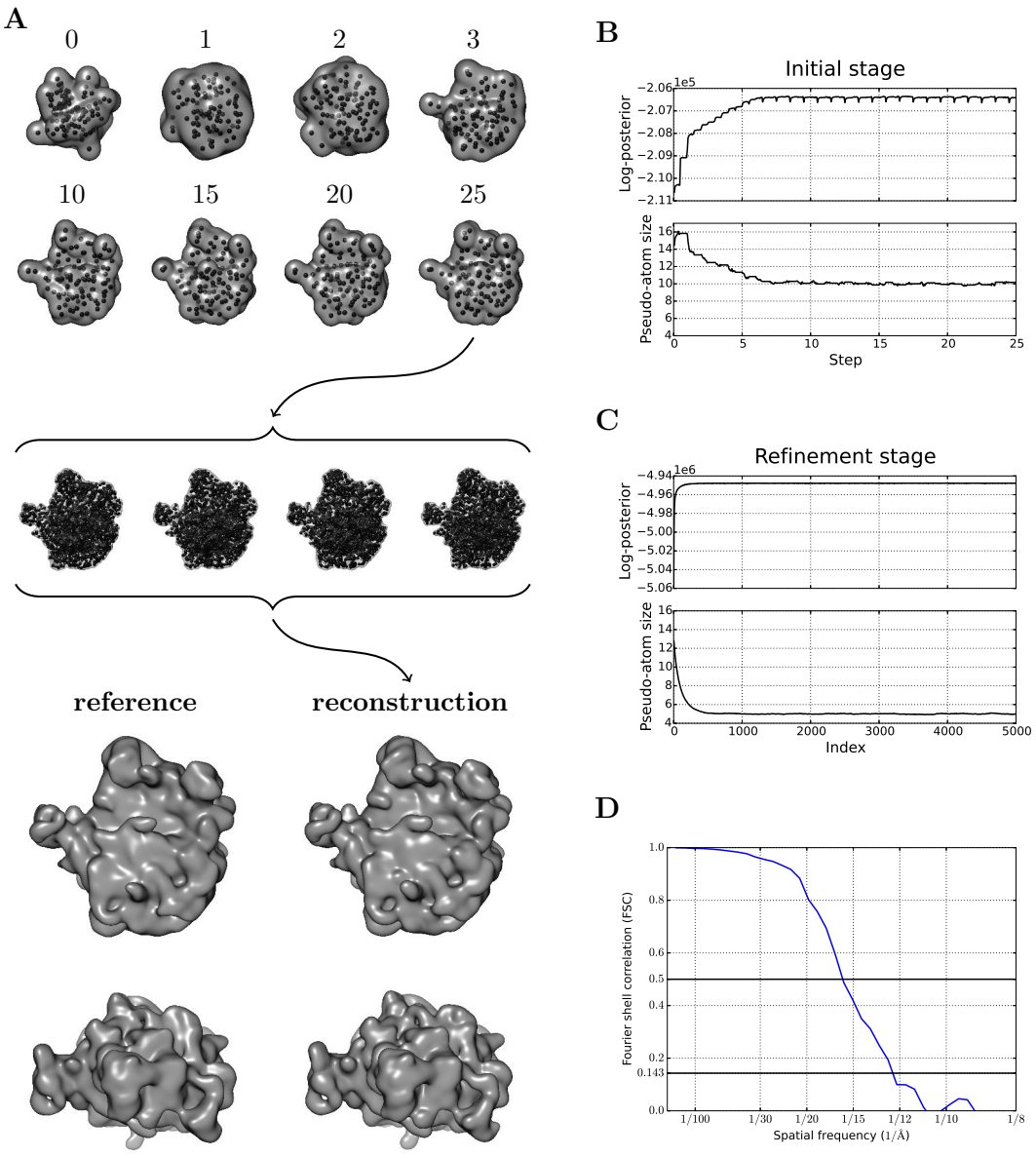

Figure 3.17: Results for the 50S ribosome. (*A*) Starting from a random initial model, the initial stage converges within ten steps. The number of components (*black spheres*) are then increased from 100 to 2000, and multiple models from the posterior distribution are shown. These are averaged to obtain the final reconstruction. The cross-correlation with the reference model at 25 Å is 0.990. (*B,C*) Monitoring the log-posterior and the component size shows that the algorithm converges quickly in both the initial and refinement stages. (*D*) The FSC curve between the reconstruction and the reference shows that they agree to a resolution of 15.9 Å at FSC = 0.5.

each global rotation update. If each of the 25 rotations has already converged to the vicinity of the true rotation, then choosing from only 1000 random rotations typically produces a slightly worse rotation. Subsequent local rotation updates quickly brings each rotation back to where it was.

During the refinement stage the log-posterior increases quickly and more evenly.

3.4.3 Multiple restarts

To improve the robustness of the initial stage, the algorithm is repeated several times with the same input data. Fig. 3.18 shows four independent runs of the initial stage using Pol II data.

The purpose of the initial stage is to estimate the rotations. The accuracy of the rotations is evaluated by comparing them to the true rotations. If R_i is the true rotation, and $\hat{R}_i$ is the corresponding estimate, then the relative rotation is given by $\hat{R}_i^T R_i$ (assuming no reflections, see below). If all the rotations were estimated correctly, then all the relative rotations should be the same. As shown in Fig. 3.18, this is the case for each of the four runs of the algorithm, for almost all the rotations. Furthermore, the runs with the fewest rotation errors can be identified as the ones with the highest log-posterior (at least in this case).

Note that the correct rotations can only be recovered relative to each other, not in an absolute sense. The inferred density map is thus a rotated version of the reference density (assuming no reflections). To see this, suppose that the true model parameters include rotations R_i and means μ_k. Then for any $A \in \mathrm{SO}(3)$, let the estimated rotations be $\hat{R}_i = R_i A^T$ and the estimated means $\hat{\mu}_k = A\mu_k$. Assume that all the other parameters were estimated correctly. Because the rotations and the means occur in the forward model (Eqn. 2.76) together as $R_i\mu_k$, the rotation A cancels out when using the estimated parameters:

$$\hat{R}_i\hat{\mu}_k = R_i A^T A\mu_k = R_i\mu_k. \tag{3.12}$$

Therefore, in this case the value of the log-posterior is exactly the same for the estimated parameters and the true parameters. The relative rotations shown in Fig. 3.18 give the value of A (assuming no reflections):

$$\hat{R}_i^T R_i = (R_i A^T)^T R_i = A. \tag{3.13}$$

The posterior probability is also invariant to a reflection of the density. This is known as the issue of handedness (or chirality) (Frank 2006, Chapter 4): it is not possible to determine the handedness of an electron density from its projections alone. To see this, let R_i and μ_k denote the true parameters as above. Then let the estimated rotations be $\hat{R}_i = SR_iS$ and the estimated means $\hat{\mu}_k = S\mu_k$, where

$$S = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.14}$$

is a reflection about the xy-plane. The invariance of the forward model follows from:

$$P_o\hat{R}_i\hat{\mu}_k = P_oSR_iSS\mu_k = P_oR_i\mu_k, \tag{3.15}$$

using $P_oS = P_o$.

If the density has undergone a reflection as well as a rotation, the new parameters would be:

$$\hat{\mu}_k = SA\mu_k \tag{3.16}$$

$$\hat{R}_i = SR_iA^TS, \tag{3.17}$$

and the relative rotation should be computed as

$$(S\hat{R}_iS)^TR_i = (SSR_iA^TSS)^TR_i = A. \tag{3.18}$$

For the examples in this section, where the true handedness is known, the handedness of the result of the algorithm can be determined by computing both versions of the relative rotations (Eqns. 3.12 and 3.18), and using the one that forms a cluster. This approach was used for Fig. 3.18 and the subsequent figures.

3.4.4 The number of components

One of the parameters to choose for the initial stage is the number of mixture model components (K). Fig. 3.19 shows the effect of K on the result of the initial stage. It shows that even with as few as 10 components, the algorithm is sometimes able to estimate the rotations correctly. From 50 components upwards they are usually correct, and therefore 100 components was chosen as the default.

The figure also shows that the total time required for the algorithm scales roughly linearly with the number of components.

3.4.5 Global rotation sampling

Another parameter needed for the initial stage is the number of rotations (N_R) to use when sampling the global rotations. More rotations ensures a more accurate approximation to the distribution over the global rotations, but increases the computational requirements.

Fig. 3.20 tests the effect of N_R on the results. It shows that if $N_R \leq 500$, then its precise value doesn't have much effect on the total time of the algorithm. With more rotations, the global rotation sampling step starts dominating the computation time, which increases linearly w.r.t. the number of rotations (in addition to a constant term). The figure also shows that for $N_R \geq 500$, almost all of the rotations are estimated correctly. Note that for other structures, a higher value of N_R might be required.

3.4.6 The number of counts

Another parameter for the initial stage is the number of counts (N_0) to use per image when discretising the real-valued class averages. Fig. 3.21 tests the effect of N_0 on the result.

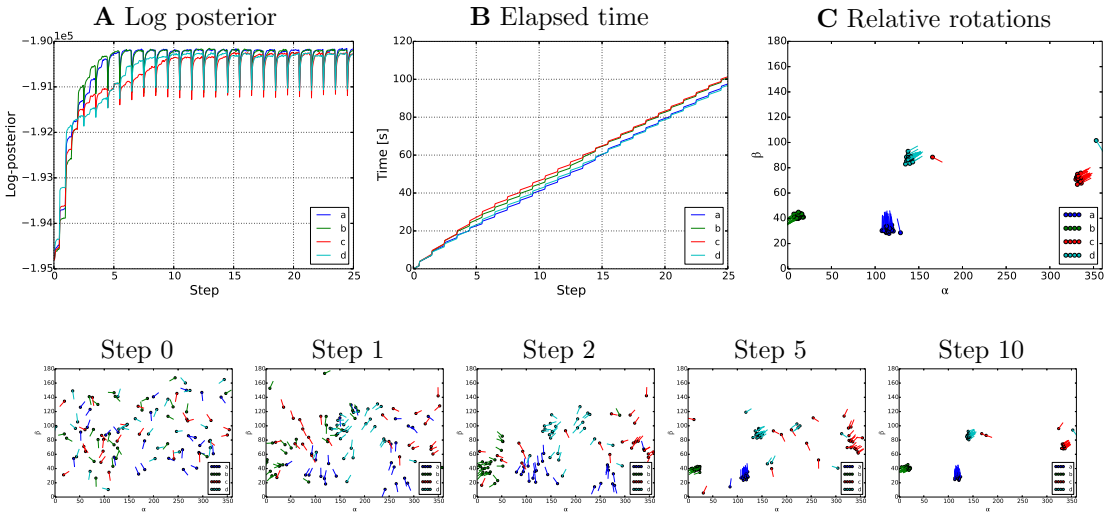

Figure 3.18: Initial stage for Pol II. Both a mixture model with 100 components and 25 rotations are sampled given 25 class averages with size 32×32 and 1000 counts per image. The input data and mixture models are not shown. The algorithm is repeated four times, corresponding to the four different colours. (A) In all four cases, the log-posterior converges to roughly the same value. (B) Every application of the algorithm takes around 100 seconds. The vertical jumps in the elapsed time at every step correspond to the global rotation sampling step, which uses 1000 random rotations in this case. (C) The rotations are compared to the true rotations; a cluster indicates that they are correct relative to each other. Each relative rotation is shown using Euler angles: α and β are the (x, y) coordinates of the marker, and γ is the direction of the line based at the marker. For two of the cases (*blue* and *green*) all rotations are correct, while for the other two cases (*red* and *cyan*) one rotation is incorrect. This is also seen in the log-posterior, which is higher for the cases with all rotations correct. (*Bottom row*) The relative rotations are initially random, but converge within a few steps.

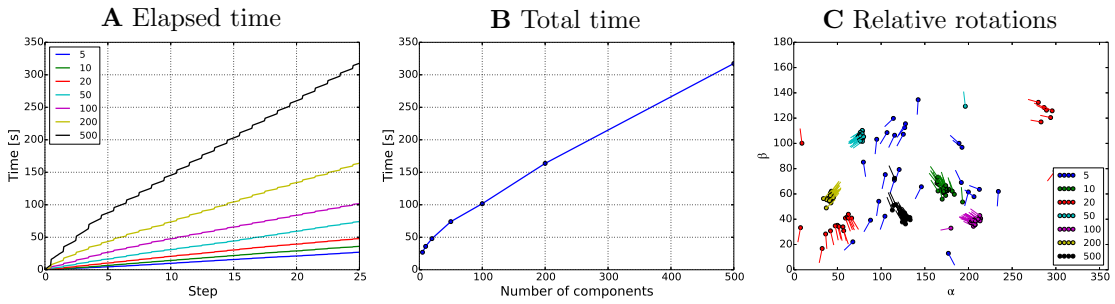

Figure 3.19: Initial stage for Pol II, varying the number of components. (*A*,*B*) As the number of components varies from 5 to 500, the total time needed by the algorithm increases roughly linearly. (*C*) With only 5 or only 20 components, the algorithm does not find the correct rotations, but in all other cases there is at most one incorrect rotation.

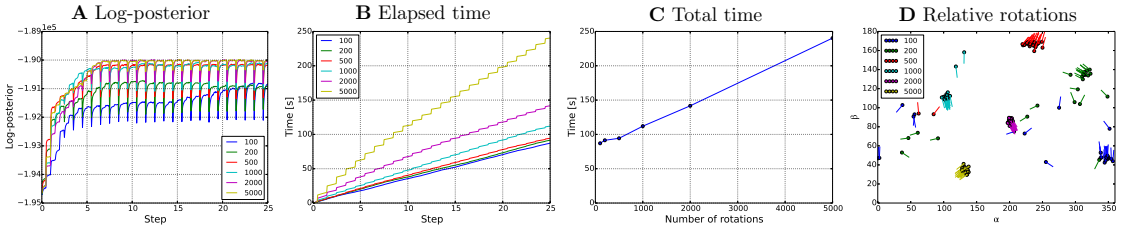

Figure 3.20: Initial stage for Pol II, varying the number of rotations (N_R) used in the global rotation sampling step. (*A*) The final log-posterior increases with N_R. The dip in the log-posterior after every global rotation sampling step decreases with increasing N_R. (*B, C*) As N_R increases from 100 to 5000, the proportion of time spent on the global rotation step increases. (*D*) With $N_R \leq 200$, only about half of the rotations are estimated correctly, while with $N_R \geq 2000$, all the rotations are correct. For the intermediate values of $N_R = 500$ and $N_R = 1000$, all but two rotations are estimated correctly.

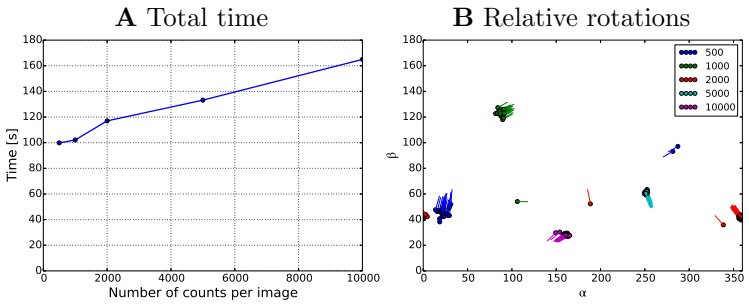

Figure 3.21: The effect of varying the number of counts (N_0). (A) The total time needed by the algorithm is composed of a constant term, and a term that depends roughly linearly on N_0. (B) In all cases, almost all rotations are estimated correctly. The variation in each cluster decreases with increasing N_0.

3.4.7 Phantom example

In addition to the 50S ribosome, the algorithm was also tested on simulated data used by Jaitly et al. (2010) to test their initial model algorithm. They projected a phantom structure in random directions to generate 50 projected images (class averages) of size 32×32. We used exactly the same 50 images as input to our algorithm to be able to compare our results with theirs.

Fig 3.22 shows the results of our algorithm using the phantom dataset. The initial stage was repeated four times with different random initial models (Fig 3.22*A*). In all four cases the log-posterior converged to roughly the same values, and the corresponding models are very similar to each other (after accounting for reflections and rotations). One of the four models was used to initialise the refinement stage (Fig 3.22*B*), where the log-posterior quickly converges.

Fig 3.22*D* shows the input images of the initial stage, and the projections of the final model of the initial stage. The input images have been discretised with 1000 counts. Corresponding images agree well. Fig 3.22*E* shows the same comparison for the refinement stage. The images are still only 32×32, but have been discretised with 10000 counts.

In Fig 3.22*F* the final posterior mean is compared to the reference model originally used by Jaitly et al. (2010) to create the input images. The two structures are visually very similar. This is confirmed by the FSC curve, which shows that they agree to a resolution of 25.4 Å at FSC=0.5 (Fig 3.22*C*).

The result obtained by Jaitly et al. (2010) using their algorithm agreed with the same reference to a resolution of only 48.9 Å at FSC=0.3. Using the same threshold of 0.3, our result agrees with the reference to a resolution of 28.8 Å, which is significantly better. Besides giving a significant improvement in resolution, our algorithm is also more than two orders of magnitude faster: it required less than an hour, compared to their reported computation time of about a week.

The algorithm of Jaitly et al. (2010) is the only other Bayesian algorithm for inferring initial models. One reason why it takes so much longer, is that it integrates over the rotations numerically by sampling rotations from the prior and using Monte Carlo integration. Most rotations produce projections that differ significantly from the input images, and therefore contribute very little to the integral being estimated. In our case, the local rotation sampler only considers rotations in a neighbourhood of the previous rotation, for which the projections are typically quite similar to the input images.

The comparatively low quality of their result can be explained by the 32×32×32 grid that is not able to represent the structure at a sufficiently high resolution. Note that even with such a coarse grid, $32^3 = 32768$ parameters are needed to represent the structure. In comparison, for the refinement stage we used 500 components; a total of 2000 parameters.

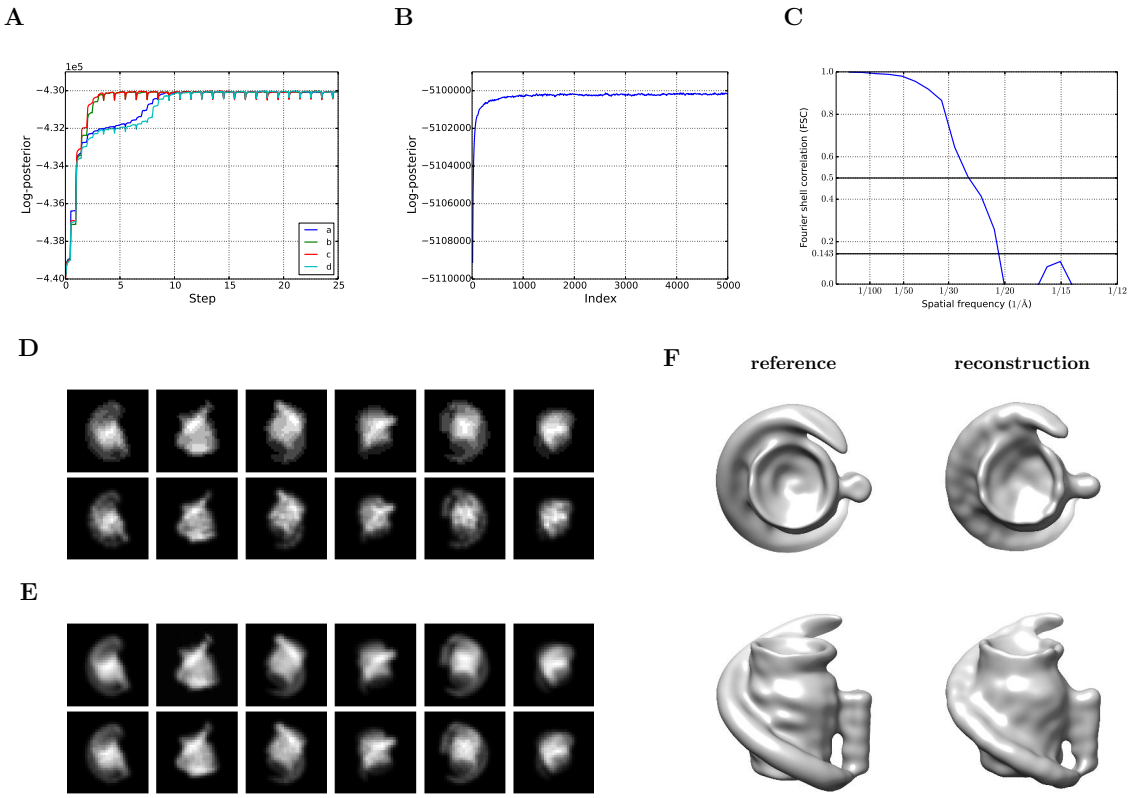

Figure 3.22: Results for the phantom from Jaitly et al. (2010), using their images as input. (*A*) Four different models are inferred during the initial stage starting from four different random models. The one with the highest log-posterior is used for the refinement stage, but any could have been used in this case. (*B*) The log-posterior converges quickly during the refinement stage. (*C*) The FSC curve between the result and the reference structures in *F* shows that the two densities agree to a resolution of 25.4 Å at FSC=0.5. (*D*) Six of the input images (*top*) are compared to the corresponding six projections of the final model of the initial stage (*bottom*). The similarity between the images shows that the rotations were estimated correctly. (*E*) The same comparison between input images and projections of the final model is also done for the refinement stage.

3.5 Computational efficiency

This section discusses implementation details which have a large effect on the running time of the algorithms.

All algorithms were implemented in a combination of Cython[1] and Python[2]. Cython translates Python code with static type annotations into C code.

3.5.1 Efficient evaluation of mixture models on grids

A common step in all the algorithms introduced in Chapter 2 is evaluating an isotropic mixture model on a regular grid. There are two cases: evaluating mixture models on three-dimensional grids to convert them to density maps, and evaluating them on three-dimensional or two-dimensional grids during the assignment sampling step.

In principle, each Gaussian component should be evaluated at each grid point. But for grid points that are far from the mean of the component, the value of the Gaussian will be vanishingly small. A very good approximation is to evaluate the Gaussian only at grid points that are within a distance of $k\sigma$ from the mean of the Gaussian, where σ^2 is the variance. A conservative threshold of $k = 5$ was used for all the algorithms in this thesis.

Introducing this cut-off significantly reduces the amount of computation needed, especially for small values of σ. For example, consider a mixture model with $K = 1000$ and $\sigma = 5\,\text{Å}$ that is converted to a density map on a grid with $N = 100^3$ voxels and a voxel size of $V = 2\,\text{Å}$. Without the cut-off, the conversion would require

$$NK = 10^9 \tag{3.19}$$

evaluations of Gaussian components. With a cut-off of $k\sigma$, there will approximately $(2k\sigma/V)^3$ grid points within a box with side length $2k\sigma$ centered at each mean. The total number of evaluations is thus only

$$\left(\frac{2k\sigma}{V}\right)^3 K = 15625000, \tag{3.20}$$

which is 1.56% of the original total. This ratio can be expressed as

$$\frac{1}{N}\left(\frac{2k\sigma}{V}\right)^3 = \left(\frac{\sigma}{20}\right)^3. \tag{3.21}$$

As another example, consider fitting a mixture model to a density map as in Section 3.1. If there were no cut-off, during the assignment step the soft assignments r_{ik} would have to be computed for each voxel i and each component k. Given the same parameters as above, that would amount to the same number of soft assignments. Restricting assignment evaluations to

[1] `cython.org`
[2] `python.org`

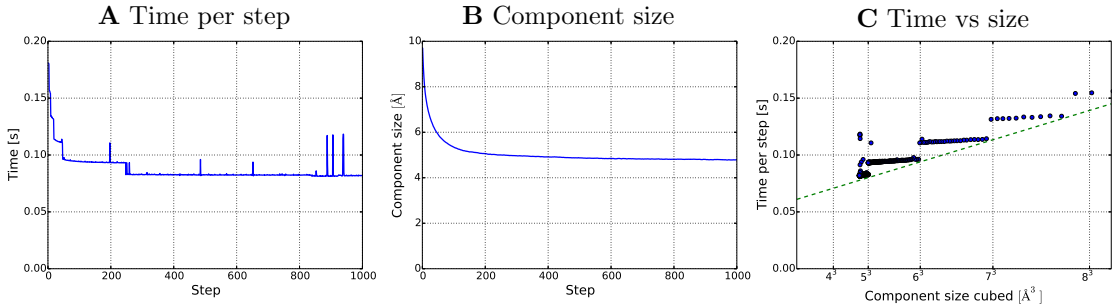

Figure 3.23: Effect of component size on duration of Gibbs sampling step. A 500-component mixture model was used to approximate a Pol II density map. (*A*,*B*) Both the time per step and the component size decrease, especially at the start of the algorithm. (*C*) The part of the computation time that depends on the component size is roughly proportional to the cube of the component size.

points within the box around each mean again reduces the required number of evaluations to $(\sigma/20)^3 = 1.56\%$ of the total.

In this example, σ is the only parameter that changes during the algorithm. The contribution of the assignment sampling step to the running time of the algorithm should therefore be proportional to σ^3. This is confirmed in Fig. 3.23. As shown in the figure, the component size decreases while running the algorithm, and therefore successive sampling steps become faster. The discrete transitions in the running time per sampling step correspond to changes in the number of grid points in each mean-centered box.

Introducing the cut-off also greatly reduces the memory requirements for the algorithm, which are dominated by the memory requirements for sampling the assignments. Again consider the example of fitting a mixture model to a density map with the parameters as given above. Without the cut-off, the 10^9 soft assignments would need about 4 GB of memory, assuming 4 bytes for each value. With the cut-off, the proportion of non-zero soft assignments is approximately $(\sigma/20)^3$, as above. The total amount of memory required is determined by the maximum value of σ. If the prior is centered around 10 Å, then for this example the maximum value of σ would be approximately 10 Å, translating to 0.5 GB of memory. Introducing the cut-off thus reduces the memory requirement by 87.5% in this case.

For inferring three-dimensional mixtures from class averages, the memory requirements are not that high. But the savings obtained from applying a threshold when evaluating the Gaussians are still significant.

Computing the soft assignments is the main contribution to the computation time for each Gibbs sampling step. This can be seen in Table 3.1, which shows the contribution of each conditional distribution to the computation time. The example used is a 500-component mixture model of Pol II with 10000 counts for each of 25 class averages. The sampler is initialised with

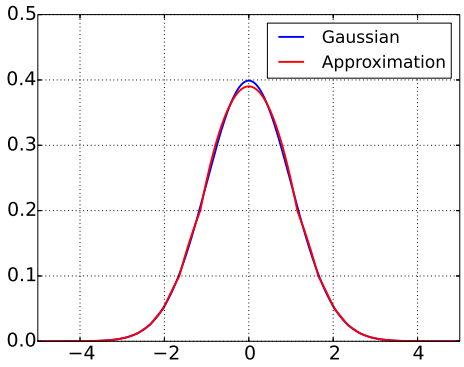

Figure 3.24: Approximating the Gaussian distribution using a fast approximation to the exponential function. The approximation is accurate enough for the Gibbs sampler to work, but faster to evaluate.

the true mixture and rotation parameters.

During each soft assignment computation the exponential function is evaluated once. A significant speed-up can be obtained by using an approximation to the exponential function described by Schraudolph 1999 (Fig 3.24). Using the approximate exponential function does not adversely affect the results of the algorithm, but reduces the computation time for the soft assignments by about 57%.

3.5.2 Sampling rotations

As mentioned in Section 2.4.2, the conditional distribution for the rotations is of the form

$$\exp \operatorname{tr}(A_i^T R_i), \tag{3.22}$$

(see Eqn. 2.92), and the algorithm by Habeck (2009) is used for sampling from it.

The most computationally expensive step in the sampling algorithm is computing the SVD (singular value decomposition) of the 3×3 matrix A_i. There are algorithms readily available for computing the SVD of a square matrix of arbitrary size, but tailor-made algorithms for 3×3 matrices are faster. We use a very efficient C++ implementation by McAdams et al. (2011).

Parameters	**Standard exponential**	**Fast approximation**
Soft assignments	46 ms (62.2 %)	20 ms (41.8 %)
Hard assignments	20 ms (27.1 %)	20 ms (41.8 %)
Missing components (x^m)	3.2 ms (4.3 %)	3.2 ms (6.7 %)
Weights (w)	0.1 ms (0.1 %)	0.1 ms (0.2 %)
Means (μ)	1.1 ms (1.5 %)	1.1 ms (2.3 %)
Precision (s)	2.5 ms (3.4 %)	2.5 ms (5.2 %)
Rotations (R), translations (t)	1.0 ms (1.4 %)	1.0 ms (2.1 %)
Total time	73.9 ms	47.9 ms

Table 3.1: The effect of using a fast approximation to the exponential function. Shown are the averages times for a single Gibbs sampling step of a 500-component mixture model with 25 rotations and translations. Using the approximate exponential function reduces the time needed to evaluate the soft assignments by 57%, and the total time by 35%.

Chapter 4

Experiments with real data

Section 3.4 of the previous chapter showed how to use our algorithm to infer initial models from simulated class averages. In this chapter, the same algorithm will be applied to real data.

The first step will be to obtain class averages that are suitable for our algorithm. The algorithm will then be tested on three different real datasets.

4.1 Computing class averages

The first step in the cryo-EM data processing pipeline (Section 1.2) is to form class averages from the individual particle images. These class averages are used as input by most initial model inference algorithms, including ours.

Fig. 4.1 shows an outline of a typical class averaging algorithm, using simulated data. The individual particle images are aligned relative to each other, and clustered to form groups. The images in each group are averaged together to create class averages.

The class averages shown in Fig. 4.1 would be suitable as input to our initial model algorithm. However, when using real data, the resulting class averages look different. Fig. 4.2 shows the class averages obtained using the software package EMAN2 with real GroEL data. Compared to Fig. 4.1, the EMAN2 class averages have both strong positive and negative values (the background value is zero). This is due to the effect of the contrast transfer function (CTF) of the electron microscope.

The CTF is part of the forward model of the electron microscope image formation process. It is not taken into account by most class averaging algorithms. Instead, a partial correction such as phase-flipping is applied to individual particle images before the class averaging step. But Fig. 4.2 shows that this is not enough to account for the effect of the CTF. As an alternative to phase-flipping, Wiener filtering is sometimes used, but it also produces class averages with both positive and negative values.

Our algorithm requires input images that are non-negative. For this we must fully correct for the effect of the CTF. Below we introduce a deconvolution algorithm that can be combined with an existing class averaging algorithm to do full CTF correction.

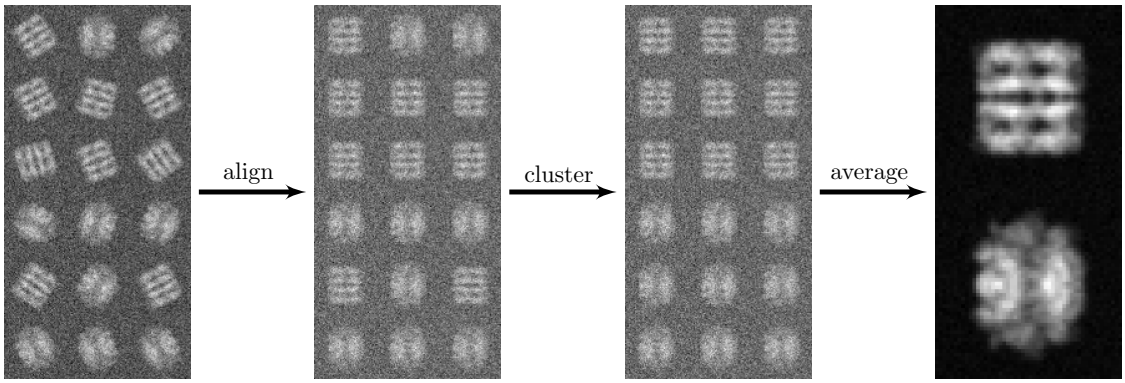

Figure 4.1: Simplified pipeline of typical class averaging algorithms. The raw images are aligned relative to each other, clustered into classes, and averaged to create class averages with high SNR. In practice, the alignment and clustering steps are combined in an iterative algorithm

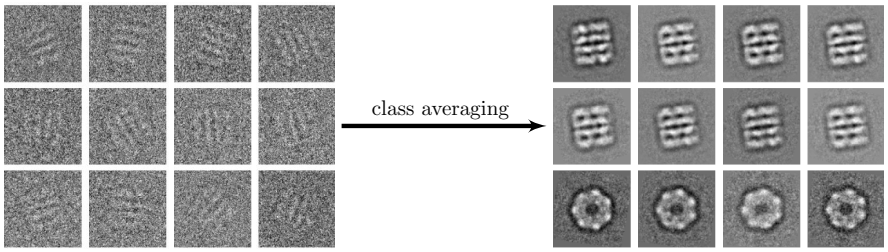

Figure 4.2: Class averaging applied to real GroEL data using EMAN2. About 5000 images were used to obtain 13 class averages. The class averages clearly show the effect of the CTF in the form of large negative values, such as the dark edges around the particles.

The deconvolution algorithm treats the class averaging problem as an inverse problem, and as in Chapter 1, the first step is to formulate the forward model.

There exists a class averaging algorithm that performs full CTF correction as part of the RELION software package. This can be used as an alternative to the deconvolution algorithm introduced below.

4.1.1 Standard image formation model

Section 1.4.5 introduced the forward model underlying many initial model algorithms. The forward model describes how each observed particle image is obtained from the electron density: by rotating the density, projecting it, and adding Gaussian noise to the projected image.

In this section we are concerned only with the last part of the forward model: from the projected image y to the observed image z. Instead of just adding Gaussian noise ϵ, the forward model is refined to include a convolution ($*$) with a point-spread function f. The final image z

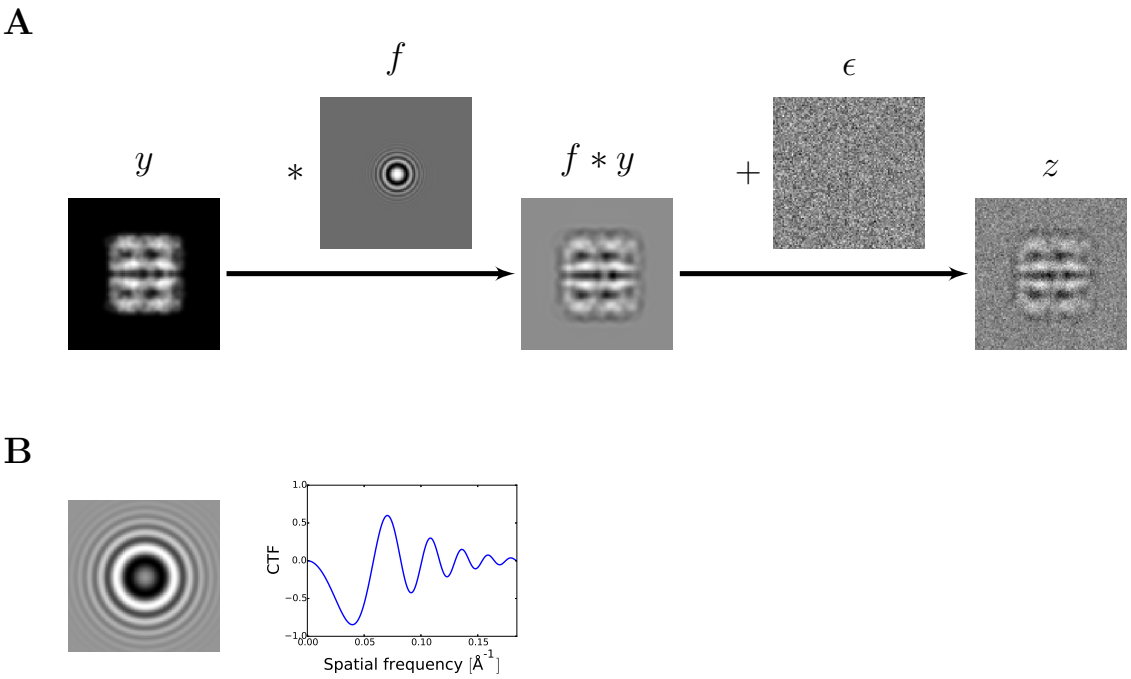

Figure 4.3: (A) The image formation model, relating the projected image y to the observed image z. The projected image is convolved by the PSF f before adding i.i.d. Gaussian noise ϵ. The negation of the actual PSF is used here, to make it easier to compare $f * y$ with y. (B) The CTF is the Fourier tranform of the PSF. In the absence of astigmatism, it is radially symmetric, and the radial profile shown here has a parametric form (Frank 2006, p. 34).

is thus obtained from y as follows (see also Fig. 4.3A):

$$z = f * y + \epsilon. \tag{4.1}$$

This model is known as the linear, weak-phase-object approximation of the image formation process in the electron microscope. For more information about image formation in cryo-EM, see Frank (2006) and Penczek (2010b).

The point-spread function (PSF) describes the effect of the electron microscope. Its Fourier transform is known as the contrast transfer function (CTF). The CTF (see Fig. 4.3B) has a parametric form, with some parameters determined by the microscope settings, and others that can be estimated from the recorded micrographs. For deconvolution, we therefore assume that the CTF (and thus the PSF) is known.

As can be seen from Fig. 4.3A, the CTF has a strong effect on the images. This effect must be taken into account at some stage of the reconstruction pipeline. A common approach is to apply a CTF-correction algorithm to individual images such as z, and then compute class averages using the CTF-corrected images. CTF-correction is thus separated from class averaging.

A widely-used CTF-correction algorithm is phase-flipping. Phase-flipping is a partial correc-

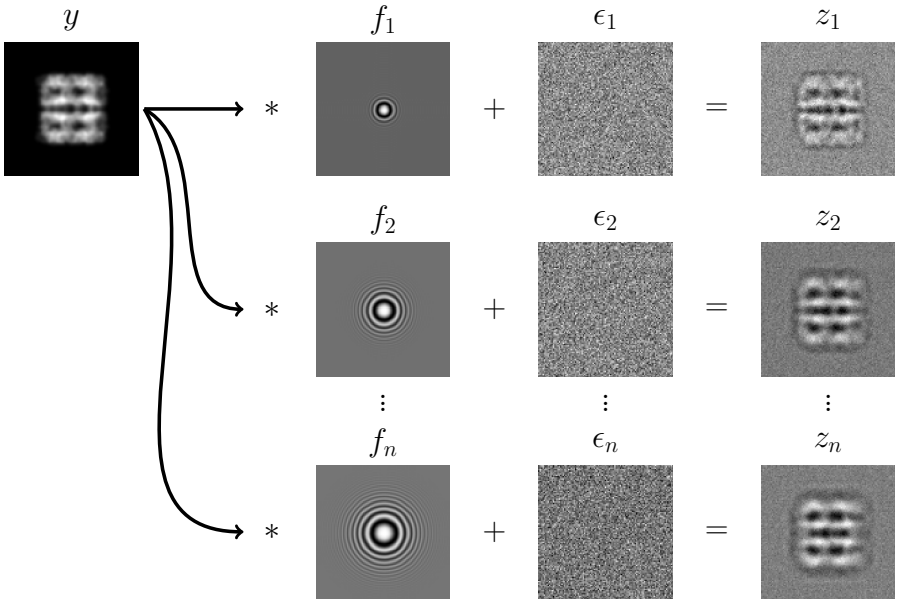

Figure 4.4: The forward model used for multi-frame deconvolution. Observed images z_i with similar orientations can be modelled as originating from the same projection image y. For each image z_i, the projection y is convolved with a different PSF f_i. The deconvolution problem is to estimate y given the z_is.

tion of the effect of the CTF where the image is transformed to Fourier space, multiplied by the sign of the CTF function, and transformed back to real space. This does not produce optimal results, because the amplitude of the CTF is ignored.

Our approach is to combine the CTF-correction step with the class averaging step, and perform full CTF-correction. This is done by appending a deconvolution step to one of the standard class averaging algorithms.

Fig. 4.4 shows how multiple images z_i are created from the same projected image y. The images z_i correspond to one of the clusters of images obtained during class averaging. The goal of deconvolution is to estimate y given the z_is.

The variation in the PSFs in Fig. 4.4 is due to changes in the defocus parameter of the CTF. Changing the defocus changes the positions of the zero-crossings of the CTF in Fig. 4.3*B*. When applying the CTF to an image, the information at the frequencies corresponding to the zero-crossings of the CTF is lost. This makes it difficult to estimate y given a single z_i. A better estimate of y can be obtained by combining z_is with different defocus settings. This is the idea behind multi-frame deconvolution.

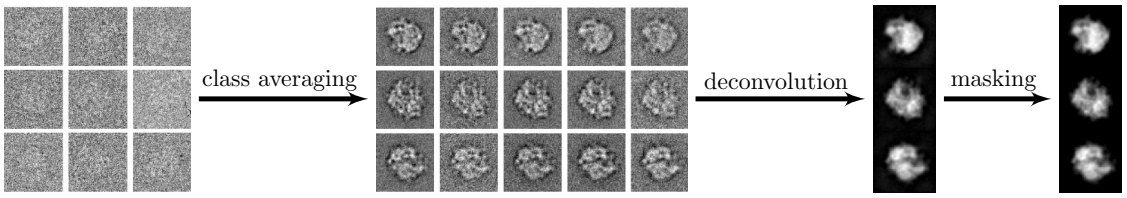

Figure 4.5: Class averaging and multi-frame deconvolution results for the 70S ribosome. The class averaging algorithm from ASPIRE is used to compute 50 class averages. The three rows correspond to the first three class averages. For each class average, seven defocus averages are computed, of which five are shown (the columns in the second image). The multi-frame deconvolution algorithm is applied to each set of seven defocus averages, followed by a masking step.

4.1.2 Deconvolution

There are many class averaging algorithms, such as the ones implemented in the software packages EMAN2 and ASPIRE (Zhao and Singer 2014). Although the output of the algorithm is a set of class averages, it is possible to determine the aligned images that were used for each class average (corresponding to the second-last image in Fig. 4.1).

The deconvolution algorithm takes the aligned images belonging to a given class as input (the z_is in Fig. 4.4), and estimates the corresponding deconvolved class average y. The forward model for generating each z_i from y is the same as in Eq. 4.1:

$$z_i = f_i * y + \epsilon_i, \tag{4.2}$$

where the PSF f_i is known, and the i.i.d. Gaussian noise ϵ_i has the same variance for all i.

The MAP (maximum a posteriori) estimate for y is found by minimizing the convex loss function:

$$L(y) = \frac{1}{2}\sum_i \|z_i - f_i * y\|^2 + \frac{1}{2}\alpha\|\nabla y\|^2, \tag{4.3}$$

subject to the constraint that y be non-negative. Here ∇ is the gradient operator, and α is a hyperparameter controlling the smoothness of y. The regularization parameter α needs to be specified. For our experiments we used $\alpha = 10$. Eq. 4.3 is optimised using the L-BFGS-B algorithm (Byrd et al. 1995).

The computation time for the optimisation algorithm depends on the number of input images z_i. The computation time can be reduced by averaging together images with a similar PSF. This is done by dividing the range of defocus values into equally sized intervals, and averaging together images whose PSFs have defocus values in the same interval. The averages are then used as the z_is in Eq. 4.3 with corresponding weights to indicate the number of images in each cluster. This simplification can have a significant effect on the computation time, without strongly affecting the results.

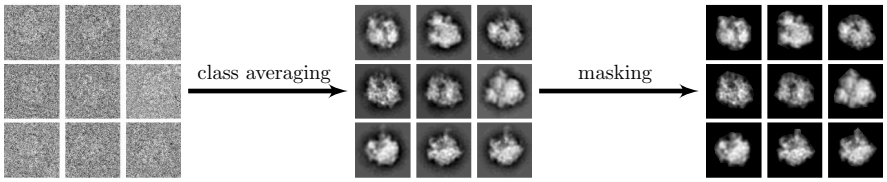

Figure 4.6: The class averaging algorithm of RELION with full CTF-correction applied to the same 70S ribosome data as in Fig. 4.5.

In the special case where there is only one defocus interval, all z_is are averaged together to form a single z, which is the original class average. The cost function then simplifies:

$$L(y) = \frac{1}{2}\|z - f * y\|^2 + \frac{1}{2}\alpha\|\nabla y\|^2. \tag{4.4}$$

This is known as single-frame deconvolution, in contrast to multi-frame deconvolution (Eq. 4.4). It can be applied directly to the class averages. Although the results are generally inferior to those obtained using multi-frame deconvolution, they can still be good enough to infer a low-resolution initial model.

Fig. 4.5 shows an application of the deconvolution algorithm to real 70S ribosome data. The original 5000 images are processed using ASPIRE's class averaging algorithm to obtain 50 class averages. The approximately 100 images contributing to each class average are clustered into 7 groups with similar defocus values, yielding 7 defocus averages. Fig. 4.5 shows only the first 5 defocus averages, for the first 3 class averages. Multi-frame deconvolution is applied to each cluster of 7 images, yielding 50 deconvolved class averages.

The deconvolved class averages are non-negative, but typically have small non-zero values near the edges (barely visible in 4.5). These values near the edge are removed by applying a mask to each image. The mask is constructed manually for each image by tracing the outline with a mouse pointer.

Instead of combining an existing class averaging algorithm with a multi-frame deconvolution step as above, an alternative is to use the class averaging algorithm from the software package RELION, which performs full CTF correction. Fig. 4.6 shows the results for the same 70S ribosome images. The edge effects are more severe, but can again be removed by applying a mask. The resulting class averages look similar to the ones from Fig. 4.5, and both can be used to infer an initial model.

4.2 Inferring initial models

The full algorithm for inferring initial models was tested on three datasets with real data.

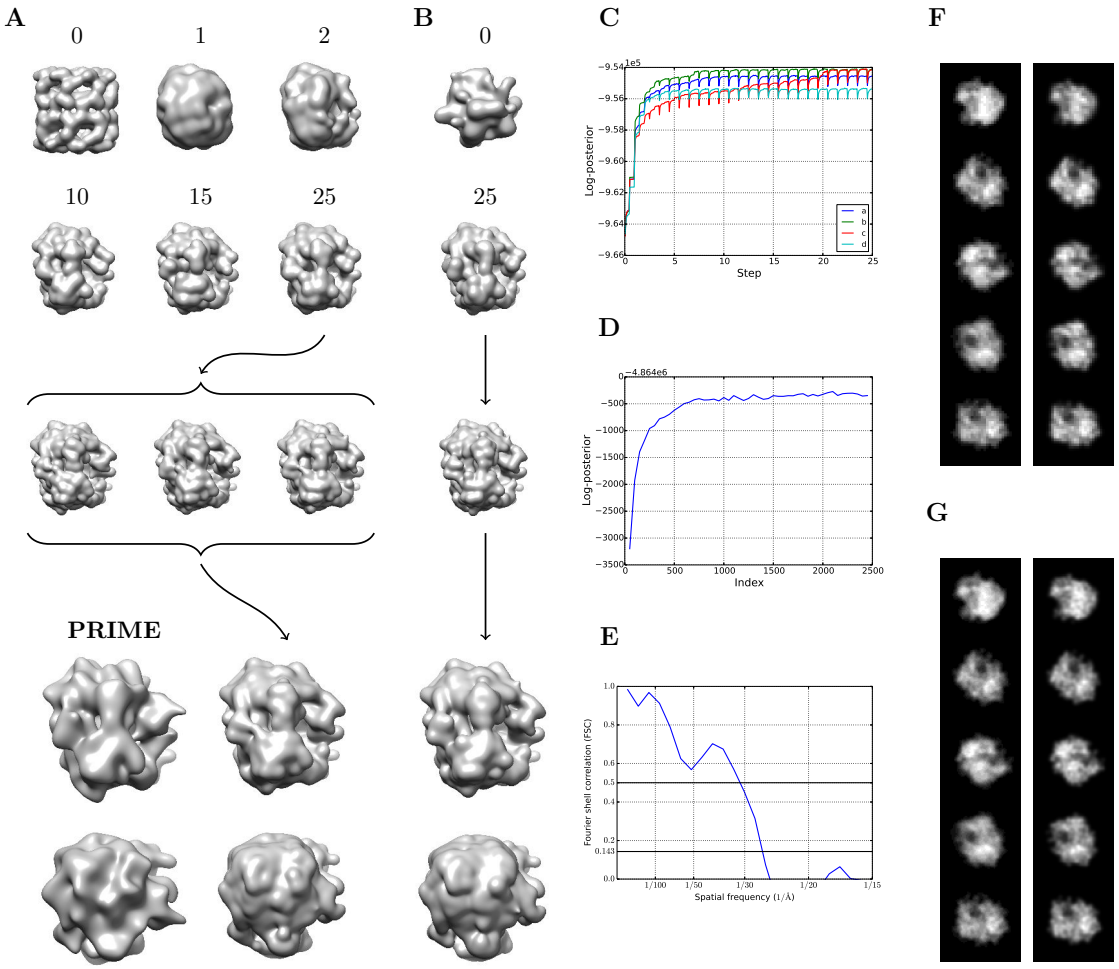

Figure 4.7: Results for the 70S ribosome, using real data. (*A*) Despite being initialised with an unrelated structure, GroEL, the algorithm converges to the correct 70S structure. The first two rows are steps from the initial stage, the next row shows models from the posterior distribution obtained during the refinement stage. At the bottom the posterior mean is compared to the result obtained using the PRIME algorithm, low-pass filtered, and using the same input data. (*B*) When initialised with a random model, the algorithm also converges to the correct structure. (*C*) During the initial stage, four different models are inferred, and only the one with the highest log-posterior (*b* in this case) is used for the refinement stage. (*D*) The log-posterior converges quickly during the refinement stage. (*E*) The FSC curve between the posterior mean in *B* and the PRIME result shows that the two densities agree to a resolution of 31.1 Å at FSC=0.5. The cross-correlation between each of the posterior means and the PRIME result is 0.900 and 0.895 respectively, which the cross-correlations between the two posterior means themselves is 0.986. (*F*) Five of the input images to the initial stage (*left*) are compared to the projections of the final model of the initial stage (*right*). (*G*) For the refinement stage the input images are compared to the projections of the final model.

4.2.1 70S ribosome example

The first dataset is the 70S ribosome, taken from the Electron Microscopy Data Bank (EMDB) (Frank 2014; Scheres et al. 2007). The EMDB has two 70S ribosome datasets with 5000 images each, representing slightly different structures. The first dataset is the ribosome with elongation factor G (EF-G), while the second dataset is without EF-G. The results for the first dataset are presented here to be able to compare to other initial model algorithms that have been tested on the same data.

The dataset consists of 5000 images of size 130×130 at a sampling rate of 2.82 Å/pixel. The class averaging and deconvolution step for the dataset was described in the previous section. The pipeline from the raw data to the deconvolved class averages is shown in Fig. 4.5.

The results of applying the initial model algorithm to the class averages are shown in Fig. 4.7. The figure shows two experiments. In the one case (Fig. 4.7*B*), the model parameters are initialised by sampling them from the prior, as usual. In the other case (Fig. 4.7*A*), a model of GroEL is used instead. In both cases the algorithm converges to the correct structure, showing that it is not biased by the choice of initial model.

For the initial stage (of Fig. 4.7*B*), four independent runs of the algorithm were carried out, while monitoring the log-posteriors (Fig. 4.7*C*). As seen in Fig. 4.7*C*, two of the runs have a similar log-posterior, higher than the other two. The one with the highest log-posterior was used to initialise the refinement stage. In the refinement stage in Fig. 4.7*D*, the log-posterior converges quickly. The posterior mean shown at the bottom of Fig. 4.7*B* was formed as the average of 50 samples from the refinement stage.

In Fig. 4.7*F*, the input images to the initial stage are compared to the projections of the final model of the initial stage. They match very well. The input images have size 32×32, and were discretised using 2000 counts. Fig. 4.7*G* shows a similar comparison between the input images to the refinement stage and the projections of the final model of the refinement stage. The input images to the refinement stage have size 64×64, and were discretised using 10000 counts.

For both experiments, the final posterior means were compared to the result obtained using the PRIME algorithm (Elmlund et al. 2013) with exactly the same input data. As can be seen from Fig. 4.7*A* and 4.7*B*, the structures are all very similar. The normalised cross-correlation between the result in Fig. 4.7*B* and the PRIME result is 0.900, and they agree to a resolution of 31.1 Å at FSC=0.5 (Fig. 4.7*E*).

The total computation time for the initial and refinement stages with GroEL as initial model was 28 minutes. The multi-frame deconvolution step took 24 minutes, while computing the class averages with ASPIRE took 50 minutes. The class averaging step used 8 cores on a desktop computer, while the other steps used a single core on a laptop. All the steps together therefore required less than 8 CPU hours. In comparison, computing the PRIME result took about 10 hours on a cluster with 40 cores. In general PRIME takes around 500 to 1000 CPU hours to compute an initial model. This example shows that the algorithm introduced in this thesis

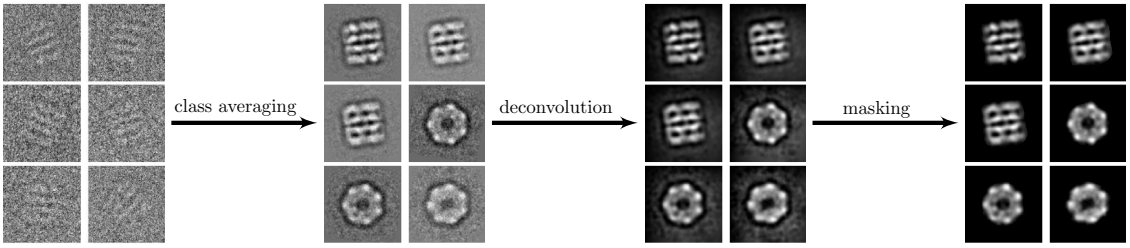

Figure 4.8: Preprocessing for GroEL data. EMAN2 is used to obtain 13 class averages, of which 6 are shown. Single-frame deconvolution and masking is applied to the class averages to obtain non-negative images that can be used as input to the initial model inference algorithm.

produces comparable results in a fraction of the time required by PRIME.

Vargas et al. (2014) also tested their proposed initial model inference algorithm using the same 70S ribosome dataset. Their result is not available for comparison, but we can compare the computation times. They used the algorithm CL2D from Xmipp to compute 16 class averages that were low-pass filtered to 25 Å. Computing the class averages took 435 minutes using two cores on a laptop. In terms of CPU hours, this takes about twice as long as computing our 50 class averages with ASPIRE. Applying their algorithm using two cores took another 50 minutes. In our case, the deconvolution and Gibbs sampling algorithm together took 52 minutes on a single core, which is again about twice as fast.

4.2.2 GroEL example

The second dataset is a publically available GroEL dataset (Ludtke 2014) consisting of about 5000 images of size 128×128 at a sampling rate of 2.12 Å/pixel.

Fig. 4.8 shows the pipeline starting from the raw particle images up to the deconvolved class averages. In the first step EMAN2 was used to obtain 13 class averages. These were then deconvolved using single-frame deconvolution, and the background was masked out. The resulting 13 images were used as input to our initial model inference algorithm.

Fig. 4.9 shows the result of applying the initial model inference algorithm. The algorithm was modified to enforce the known D7 symmetry group on the result. As with the 70S ribosome, four independent runs of the algorithm were used for the initial stage (Fig. 4.9*A*). The final model with the highest log-posterior was used for the refinement stage (Fig. 4.9*B*).

For the final model of the initial stage, its projections are very similar to the input data (Fig. 4.9*D*). The same is true for the final model of the refinement stage (Fig. 4.9*E*). The input images to the initial stage were downsampled to size 50×50, and discretised with 5000 counts. The refinement stage used the original input image size of 128×128, with 50000 counts.

In Fig. 4.9*F* the posterior mean produced by the algorithm is compared to a GroEL reference model (PDB:1OEL) at 20 Å. The two structures have a cross-correlation of 0.927, and they agree

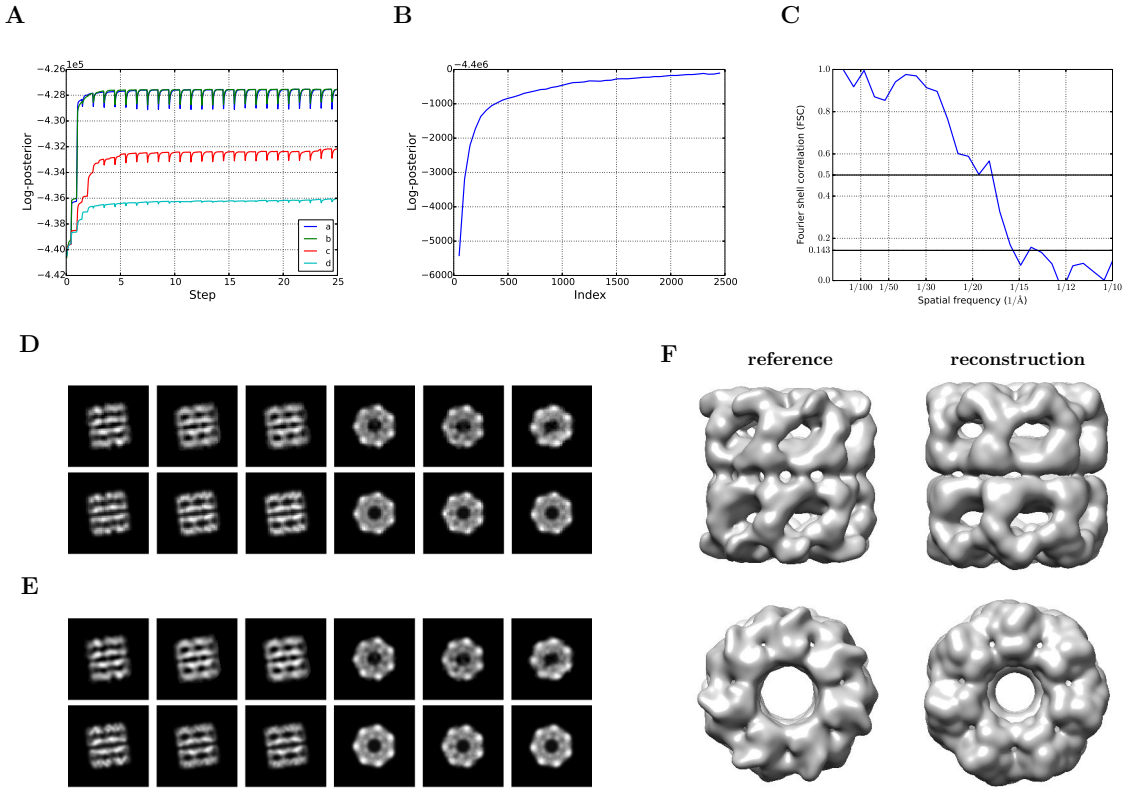

Figure 4.9: Results for GroEL, using real data. (*A*) Four different models are inferred during the initial stage; the one with the highest log-posterior is used for the refinement stage. (*B*) The log-posterior converges during the refinement stage. (*C*) The FSC curve between the result and the reference structures in *F* shows that the two densities agree to a resolution of 17.5 Å at FSC=0.5. (*D*) Six of the input images (*top*) are compared to the corresponding six projections of the final model of the initial stage (*bottom*). The images are very similar. (*E*) The same comparison between input images and projections of the final model is also done for the refinement stage.

to a resolution of 17.5 Å at FSC=0.5 (Fig. 4.9*C*).

Computing the class averages with EMAN2 took 19 minutes, and the deconvolution took 2 minutes. Applying the initial model inference algorithm took 13 minutes, from the initial stage to the posterior mean. All the steps besides the EMAN2 class averaging were performed on a single core of a laptop.

4.2.3 APC/C example

The third dataset is from the human Anaphase Promoting Complex (APC/C) (Frye et al. 2013). About 10000 particles of size 80×80 at a sampling rate of 4.9 Å/pixel were processed to produce 61 class averages. The class averages were deconvolved using single-frame deconvolution, and masks were applied. See Fig. 4.10 for the pipeline.

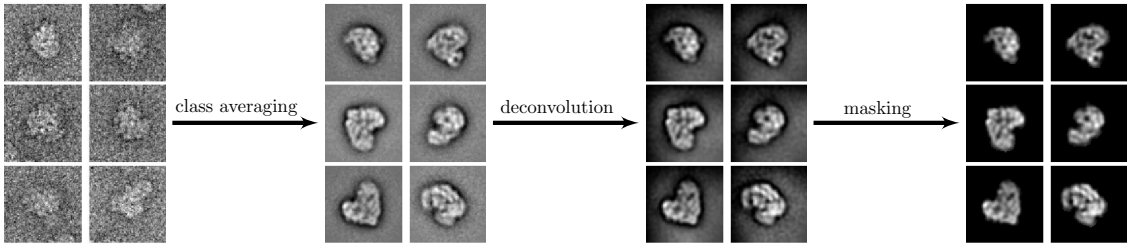

Figure 4.10: Preprocessing steps for APC/C data.

Fig. 4.11 shows the result of our initial model inference algorithm. As with the previous two experiments, exactly two of the four initial model runs reached roughly the same log-posterior (Fig. 4.11*A*). The refinement stage converged quickly (Fig. 4.11*B*).

The input data is compared to the final projections for the initial stage (Fig. 4.11*D*) and the refinement stage (Fig. 4.11*E*). There are small difference for the initial stage, but the images agree very well in the refinement stage. For the initial stage, the images were downsampled to 32×32, and discretised with 1000 counts. The refinement stage used the original input image size of 80×80, with 20000 counts.

In Fig. 4.11*F* the posterior mean produced by the algorithm is compared to the reconstruction (EMD-2354) published using data from the same source (Frye et al. 2013). The structures have a cross-correlation of 0.902 and agree to a resolution of 24.8 Å at FSC=0.5.

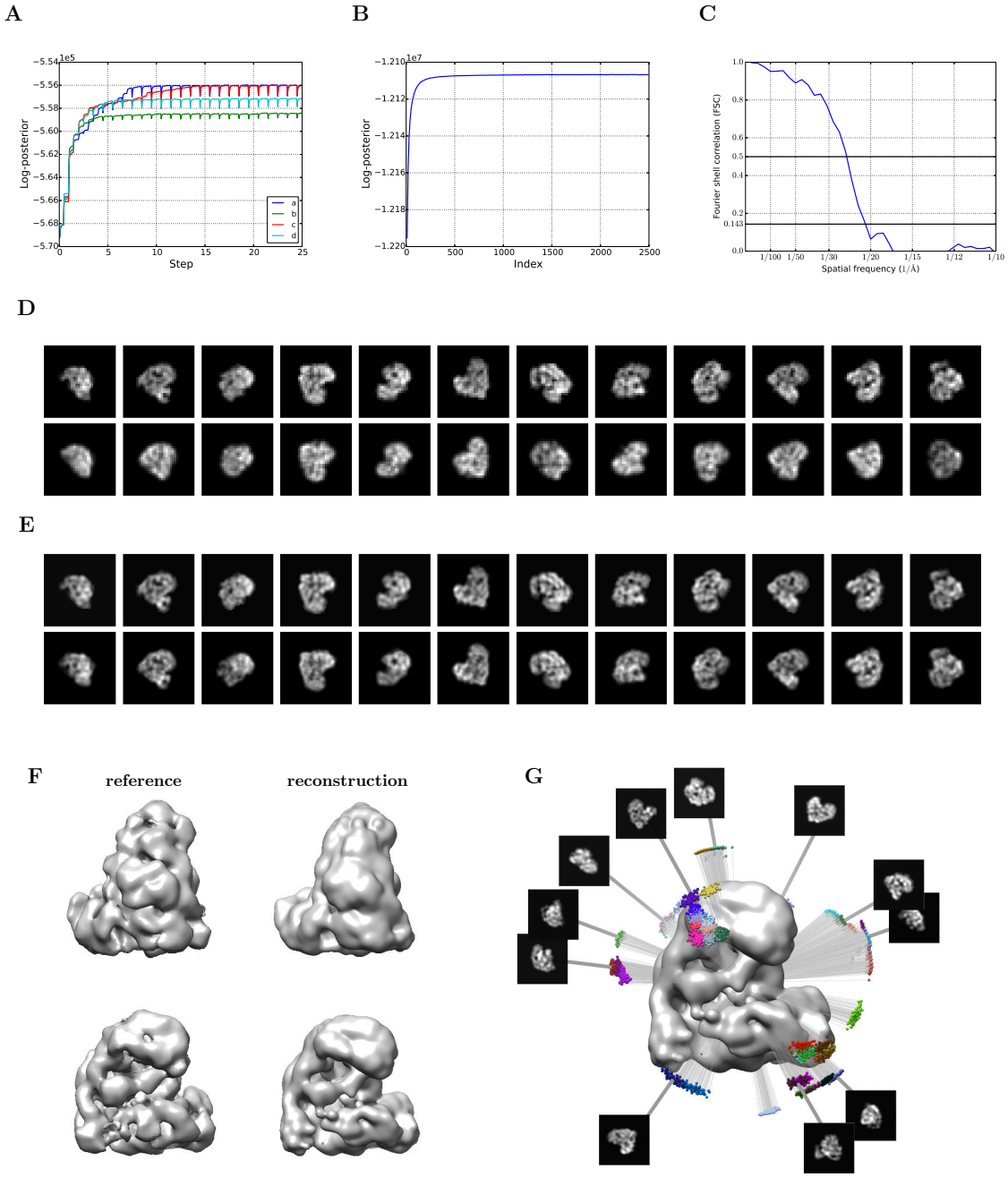

Figure 4.11: Results for APC/C, using real data. (*C*) Four different models are inferred during the initial stage; the one with the highest log-posterior is used for the refinement stage. (*D*) The log-posterior converges during the refinement stage. (*E*) The FSC curve between the result and the reference structures in *F* shows that the two densities agree to a resolution of 24.8 Å at FSC=0.5. (*D*,*E*) For the initial and refinement stages, the input images are compared to the projections of the final model. (*G*) The distribution of rotations at the end of the initial stage. For each image, a cluster of rotations corresponding to multiple Gibbs sampling steps are shown. The width of each cluster gives an indication of the precision of the estimated rotation.

Chapter 5

An alternative model with Gaussian noise

In Chapters 2, 3 and 4, we introduced an initial model inference algorithm, and tested it on simulated and experimental data.

One disadvantage of applying the algorithm to experimental data is that the images have to be non-negative. This is not the case for the class averages produced by most class averaging algorithms. In Chapter 4 we introduced a pre-processing deconvolution step to our algorithm to ensure that the input images are non-negative. In addition to the extra deconvolution step, it is also necessary to mask out the background in the deconvolved images.

In this chapter, we introduce a different initial model algorithm, one that does not need a deconvolution or masking step. Instead, the input data will be the unmodified class averages produced by common class averaging algorithms. In particular, the images no longer need to be non-negative.

The previous algorithm required images to be non-negative because it was based on a statistical forward model that generates only non-negative images. We therefore need to modify the forward model. In particular, the forward model will model the error as Gaussian noise. This is consistent with the algorithms reviewed in Chapter 1, most of which also assume a Gaussian noise model, either implicitly or explicitly.

The error model used by the previous algorithm can be viewed as an approximation to Poisson noise. To distinguish the two algorithms, we'll refer to the previous one as the Poisson algorithm, and the new one as the Gaussian algorithm.

We will present the modified forward model, followed by the new initial model algorithm based on it. The algorithm will be evaluated on simulated and experimental data.

5.1 Statistical forward model

Informally, the new forward model for generating an image is to rotate the electron density, project it along the z-axis, translate it, and add Gaussian noise.

The model parameters for the new forward model are mostly the same as before. The electron density is still represented by a mixture of Gaussians. One difference is that the mixture is not normalised, i.e. the weights do not have to add up to 1. A second difference is that all the weights are set to be equal to each other, i.e. there is only a single weight parameter λ. The experiments with simulated data in Section 3.1.5 suggest that this restriction does not seriously limit the ability of the model to represent electron densities accurately.

In addition to the weight λ, the other mixture model parameters are still the means μ for each of the K components, and the component size (as the precision s). For computing the projections, rotations R and translations t are used as before, one for each of the P images. A new parameter is the variance (or precision β) of the Gaussian noise in each image. All images are assumed to have noise with the same variance. The model parameters are denoted by $\theta = \{\mu, \lambda, s, \beta, R, t\}$.

The data $\mathcal{D}$ consists of the P input images. The regular grid for the ith image is described by its grid points $x_{ij} \in \mathbb{R}^2$, where j goes from 1 to N, the total number of pixels. The data $\mathcal{D}$ consists of all the values $y_{ij} \in \mathbb{R}$, one for each grid point. Whereas before the y_{ij}s were non-negative integers, they are now real values.

The forward model relating the data to the model parameters is:

$$p(\mathcal{D}|\theta) = \prod_{ij} \mathcal{N}(y_{ij}|\hat{y}_{ij}, 1/\beta), \tag{5.1}$$

where

$$\hat{y}_{ij} = \lambda \sum_k \exp\{-\frac{s}{2}\|x_{ij} - (P_o R_i \mu_k + t_i)\|^2\}, \tag{5.2}$$

$$P_o = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}. \tag{5.3}$$

Note that the coefficient $\frac{s}{2\pi}$ is missing from the Gaussian in Eqn. 5.2. This can be interpreted as reparametrising the pair (λ, s) to absorb the coefficient into λ. It does not change the parameter space of the model, but simplifies the computations.

Compared to the previous forward model, we no longer sample points from the probability density, or compute histograms of the projected points. It follows that the electron density does not need to be a probability density anymore. We are now free to use other functions to represent the electron density. Nevertheless, many of the advantages of the isotropic Gaussian mixture model are still valid here, such as how easy it is to compute projections, or its ability to represent structures using only a few parameters.

The algorithm for estimating the model parameters will again be based on a Bayesian approach. In addition to the likelihood given in Eqn. 5.1, we therefore again need to define a prior

over the parameters. The prior distibution factorises over all the parameters:

$$p(\theta) = p(\mu)p(\lambda)p(s)p(\beta)p(R)p(t). \tag{5.4}$$

The priors on the means μ and rotations R are the same as before:

$$p(\mu) = \prod_{k=1}^{K} p(\mu_k) = \prod_{k=1}^{K} \mathcal{N}(\mu_k|0, r^{-1}I) \tag{5.5}$$

$$p(R) \propto 1, \tag{5.6}$$

and the prior on the noise precision β is a Gamma distribution:

$$p(\beta) = \text{Gamma}(\beta|a_\beta, b_\beta). \tag{5.7}$$

For the component precision s ($s = 1/\sigma^2$, where σ^2 is the variance), the weight λ, and the translations t, we use constant, improper priors. In other words:

$$p(s) = 1 \tag{5.8}$$

$$p(\lambda) = 1 \tag{5.9}$$

$$p(t) = 1. \tag{5.10}$$

These functions do not have finite integrals, and are therefore not probability distributions. But they can be used in computing the posterior, and lead to proper posterior distributions. Formally they can be seen as limits of using the following priors (for s and t these are the same as used in the Poisson model):

$$p(s) = \text{Gamma}(s|a_s, b_s) \propto s^{a_s-1}e^{-b_s s} \tag{5.11}$$

$$p(\lambda) = \text{Gamma}(\lambda|a_\lambda, b_\lambda) \tag{5.12}$$

$$p(t) = \prod_{i=1}^{P} p(t_i) = \prod_{i=1}^{P} \mathcal{N}(t_i|0, r_t^{-1}). \tag{5.13}$$

The limit is taken as $a_s, a_\lambda \to 1$, $b_s, b_\lambda \to 0$ and $r_t \to 0$.

The use of improper priors is justified by the results from Section 3.1.5. There it was shown for the Poisson model that the final results were robust to large-scale variations in the prior hyperparameters. It is reasonable to assume that the same should apply to the algorithm in the present chapter.

Using improper priors is not critical to the algorithm introduced below. Replacing them by proper priors would require only minor changes.

5.2 Gibbs sampling algorithm

As before, we will use Gibbs sampling to generate samples from the posterior distribution $p(\theta|\mathcal{D})$. The previous approach was to introduce missing data $\mathcal{Z}$, and use Gibbs sampling to sample

from the extended posterior $p(\theta, \mathcal{Z}|\mathcal{D})$. Augmenting the parameters θ with $\mathcal{Z}$ ensures that the conditional distributions all have a tractable form which can be sampled from.

The new approach will not use any missing data. As a result, most of the conditional distributions do not have a simple form. We will use different techniques to sample from the different conditional distributions.

Instead of the posterior $p(\theta|\mathcal{D})$, many of the techniques work with the negative log-posterior, also known as the energy $E(\theta)$. Up to an additive constant the energy is defined as:

$$E(\theta) = -\log p(\theta|\mathcal{D}). \tag{5.14}$$

From Bayes' rule, it follows that, up to a constant:

$$E(\theta) = -\log p(\mathcal{D}|\theta) - \log p(\theta), \tag{5.15}$$

where the negative log-likelihood follows from Eqn. 5.1:

$$-\log p(\mathcal{D}|\theta) = -PN \log \beta + \frac{\beta}{2} \sum_{ij} (y_{ij} - \hat{y}_{ij})^2 \tag{5.16}$$

and the negative log-prior follows from Eqns. 5.5 to 5.7:

$$-\log p(\theta) = \frac{r}{2} \sum_{k=1}^{K} \mu_k^T \mu_k - \frac{a_\beta - 1}{\beta} + b_\beta. \tag{5.17}$$

Sampling from some of the conditional distributions will require the gradients of $E(\theta)$ relative to the parameters being sampled. The relevant gradients are given in Appendix D.

5.2.1 Means

The means μ are sampled using Hamiltonian Monte Carlo (HMC), also known as Hybrid Monte-Carlo (Duane et al. 1987; Neal 2011). This is a general MCMC algorithm for sampling parameters θ from a distribution $q(\theta)$ by making use of the gradient of $-\log q(\theta)$ relative to θ.

In our case, we would like to sample from the conditional distribution

$$p(\mu|\theta_{\setminus\mu}, \mathcal{D}), \tag{5.18}$$

where $\theta_{\setminus\mu}$ denotes all parameters excluding the means.

We consider μ as a vector of length $3K$ obtained by concatenating all the means. The corresponding energy function follows from Eqn. 5.15 by removing terms which do not depend on μ:

$$E(\mu) = \frac{\beta}{2} \sum_{ij} (y_{ij} - \hat{y}_{ij})^2 + \frac{r}{2} \sum_{k} \mu_k^T \mu_k. \tag{5.19}$$

The gradient of the energy function is derived in Appendix D:

$$\frac{\partial E}{\partial \mu_k} = -\beta\lambda s \sum_{ij} v_{ijk}(y_{ij} - \hat{y}_{ij}) R_i^T P_o^T [x_{ij} - (P_o R_i \mu_k + t_i)] + r\mu_k. \tag{5.20}$$

The gradient $\partial E/\partial \mu$ is a vector of length $3K$ obtained by concatenating the gradients $\partial E/\partial \mu_k$ from Eqn. 5.20. The energy function along with the gradient forms the input to the HMC algorithm.

Here we give an informal overview of HMC. For more information, see Bishop (2006) and Neal (2011).

We follow the notation and example of Neal (2011). Let q denote the model parameters instead of θ or μ, and let $U(q)$ be the energy function. A simple example is that of a two-dimensional puck (from ice-hockey) sliding across a frictionless surface of varying height. The position of the puck, seen from above, is given by a two-dimensional vector q, and its momentum (mass times velocity) is described by another two-dimensional vector p. The potential energy $U(q)$ of the puck is proportional to its height, i.e. $U(q)$ describes the surface. The kinetic energy $K(p)$ is given by $|p|^2/(2m)$, where m is the mass of the puck. The total energy is given by the Hamiltonian:

$$H(q, p) = U(q) + K(p). \tag{5.21}$$

As the puck moves across the surface, potential energy and kinetic energy are exchanged, but the value of the Hamiltonian stays the same.

Now suppose that $U(q)$ is the energy corresponding to a probability density function, and the current value of the model parameters is q_0. To use HMC, we start by sampling an initial value p_0 for the momentum from a Gaussian distribution around zero. The dynamics of the puck is then simulated for a certain predetermined time, after which it has reached position q_1. The movement is simulated using Hamilton's equations of motion (from classical mechanics):

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i} \tag{5.22}$$

$$\frac{dq_i}{dt} = -\frac{\partial H}{\partial q_i}, \tag{5.23}$$

where p_i and q_i are the components of the vectors p and q.

If the movement could be simulated exactly, the new position q_1 would then be the next sample in the Markov chain, and the procedure would be repeated with q_1 as the new q_0. Intuitively, the puck will naturally move to regions with low potential energy, i.e. with high probability. Every HMC update starts with a new velocity impulse in a random direction. This could help the puck escape local minima in the potential energy function.

In practise, the movement is simulated by discretising the equations of motion. Without going into detail, we only note that the important algorithmic parameters are the step size ϵ for

the discretisation, and the number of discrete steps L from (q_0, p_0) to (q_1, p_1). Furthermore, the proposed new position q_1 is accepted with probability

$$\min\left[1, \exp(-H(q_1, p_1) + H(q_0, p_0))\right]. \tag{5.24}$$

This is needed for detailed balance to hold, which is one of the requirements for a valid MCMC algorithm.

We fixed $L = 10$ for our experiments. The value of ϵ is modified after every HMC step: if the proposed position was accepted, ϵ is increased by 2%, while if the position was rejected, ϵ is decreased by 2%. Once the Gibbs sampler has converged, ϵ can be fixed to obtain legitimate samples. The initial value for epsilon was set to 0.1 for the means, and 0.01 for the rotations below.

5.2.2 Rotations

Each rotation R_i is also sampled using the HMC algorithm from the previous section. To apply the algorithm to the rotations, which live in SO(3), it is necessary to first choose a local parameterisation of SO(3) using Cartesian coordinates. The energy with respect to the local coordinates, together with its gradient, is the input to the HMC algorithm.

We use exponential coordinates to parametrise SO(3). Informally, this parametrisation associates with every $v \in \mathbb{R}^3$ a rotation $R(v)$ in the following way: Let $v = \theta\overline{v}$, where $\overline{v}$ has norm 1, and $\theta \geq 0$. Then $R(v) \in \mathrm{SO}(3)$ is a rotation of θ radians about the axis v.

More formally, let so(3) be the Lie algebra corresponding to the Lie group SO(3). The Lie algebra so(3) is represented using the 3×3 skew-symmetric matrices. For any vector $a \in \mathbb{R}^3$, let $[a]_\times \in \mathrm{so}(3)$ denote the following skew-symmetric matrix:

$$[a]_\times := \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}. \tag{5.25}$$

The exponential map from so(3) to SO(3) is given by the Euler-Rodrigues formula (Gallego and Yezzi 2014):

$$R = Id + \sin\theta[\overline{v}]_\times + (1 - \cos\theta)[\overline{v}_\times]^2. \tag{5.26}$$

The exponential coordinates described above are given by the map taking v to $R(v) = \exp([v]_\times)$, where exp denotes the matrix exponential.

In addition to the function from v to $R(v)$, we need the gradient of the function (see Gallego and Yezzi (2014) for the derivation):

$$\frac{\partial R}{\partial v_i} = \frac{v_i[v]_\times + [v \times (Id - R)e_i]_\times}{\|v\|^2} R. \tag{5.27}$$

To use the HMC algorithm, we need the energy $E(v)$ as a function of the exponential coordinates, and its gradient $\partial E/\partial v$. Appendix D derives the energy $E(R)$ as a function of the rotation, and its gradient:

$$E(R_i) = \frac{\beta}{2}\sum_j (y_{ij} - \hat{y}_{ij})^2 \tag{5.28}$$

$$\frac{\partial E}{\partial R_i} = -\beta\lambda s \sum_{jk} v_{ijk}(y_{ij} - \hat{y}_{ij})[x_{ij} - (P_o R_i \mu_k + t_i)]\mu_k^T. \tag{5.29}$$

The energy $E(v)$ is found by composing the function $R(v)$ with $E(R_i)$, and its gradient follows from applying the chain rule to Eqns. 5.27 and 5.29.

When using exponential coordinates, all rotations can be represented by vectors $\|v\| \leq \pi$. Gallego and Yezzi (2014) argue that v should be kept inside this sphere, preferably close to the origin. To achieve this, they recommend redefining the exponential coordinates to be relative to a reference rotation R_0: $R(v) = \exp([v]_\times)R_0$. In our case, R_0 is defined as the initial rotation at the beginning of every HMC trajectory. The HMC steps along the trajectory then use the same parametrisation $R(v)$ relative to R_0. As long as each rotation does not undergo a large change during a single HMC sampling step, v will stay close to the origin.

For sampling the rotations, the HMC algorithm uses the same values for the parameters L and ϵ as used for the means. The initial value for ϵ is 0.01 instead of 0.1.

5.2.3 Weight and component size

The weight λ and component precision s are sampled from the conditional distribution

$$p(\lambda, s|\theta_{\setminus\lambda,s}, \mathcal{D}), \tag{5.30}$$

to be denoted $q(\lambda, s)$ in this section. Our strategy is to sample s from the marginal $q(s)$, and then sample λ from the conditional $q(\lambda|s)$. The marginal is given by

$$q(s) = \int_{\mathbb{R}} q(\lambda, s) d\lambda. \tag{5.31}$$

The energy function of the joint distribution $q(\lambda, s)$ is given (up to a constant) by:

$$E(\lambda, s) = -\log q(\lambda, s) \tag{5.32}$$

$$= \frac{\beta}{2}\sum_{ij}(y_{ij} - \lambda\tilde{y}_{ij})^2, \tag{5.33}$$

where

$$\tilde{y}_{ij} = \sum_k \exp\left\{-\frac{s}{2}\|x_{ij} - (P_o R_i \mu_k + t_i)\|^2\right\}. \tag{5.34}$$

The energy function $E(\lambda, s)$ is quadratic in λ, i.e. $q(\lambda, s)$ is proportional to a Gaussian, seen as a function of λ. By integrating out λ, we obtain the energy function corresponding to the conditional (up to a constant, see Appendix D for the derivation):

$$\tilde{E}(s) = -\log q(s) \tag{5.35}$$

$$= \frac{1}{2}\log\left(\frac{\beta}{2}\sum_{ij}\tilde{y}_{ij}^2\right) - \frac{\beta}{2}\left(\frac{\left(\sum_{ij} y_{ij}\tilde{y}_{ij}\right)^2}{\sum_{ij}\tilde{y}_{ij}^2} - \sum_{ij} y_{ij}^2\right). \tag{5.36}$$

To sample s from $\tilde{E}(s)$, we use the Metropolis-Hastings algorithm. This is a simple MCMC method where, given the current value s_n, a new proposal s' is sampled from a Gaussian distribution $\mathcal{N}(s|s_n, \sigma_s^2)$. The new value s_{n+1} is given by:

$$s_{n+1} = \begin{cases} s' \text{ if } t < \exp\left\{-(\tilde{E}(s') - \tilde{E}(s_n))\right\} \\ s_n \text{ otherwise,} \end{cases} \tag{5.37}$$

where t is sampled from the uniform distribution on the unit interval.

The initial value s_0 is specified by the user (we used $s_0 = 10$ for the experiments in this chapter). The scale parameter σ_s is modified in the same way as ϵ in the previous section. I.e., if the proposal was accepted, it is increased by 2%, otherwise decreased by 2%. The initial value for σ_s was 0.0001.

To generate a single sample from $q(s)$, we repeat the above Metropolis-Hastings step 5 times, and use the final sample.

The next step is to sample λ from $q(\lambda|s)$. In Appendix D we show that the conditional distribution is Gaussian:

$$q(\lambda|s) = \mathcal{N}(\lambda|\mu_\lambda, s_\lambda^{-1}), \tag{5.38}$$

with mean and precision given by:

$$\mu_\lambda = \frac{\sum_{ij} y_{ij}\tilde{y}_{ij}}{\sum_{ij}\tilde{y}_{ij}^2} \tag{5.39}$$

$$s_\lambda = \beta\sum_{ij}\tilde{y}_{ij}^2. \tag{5.40}$$

The weight λ can therefore be sampled directly.

Instead of interpreting the weight as a scaling factor applied to the three-dimensional mixture model, we can view it as a scale applied to each projection of the mixture. Instead of using the same scale for all the image, we could introduce multiple weights λ_i, one for each image.

This modification to the model would be appropriate if different input images have different scaling factors. The above algorithm could still be used, after small modifications. The computational cost should be very similar.

Note that the energy function in $\tilde{E}(s)$ from Eqn 5.36 can also be written as

$$\tilde{E}(s) = \frac{1}{2}\log\left(\frac{\beta}{2}\sum_{ij}\tilde{y}_{ij}^2\right) + \frac{\beta}{2}\sum_{ij}y_{ij}^2(1-\rho^2) \tag{5.41}$$

$$\rho^2 = \frac{(\sum_{ij} y_{ij}\tilde{y}_{ij})^2}{\sum_{ij}\tilde{y}_{ij}^2 \sum_{ij} y_{ij}^2}, \tag{5.42}$$

where ρ is the cross-correlation between the observed and predicted images. Minimising the second term of $\tilde{E}(s)$ in s is therefore equivalent to maximising the cross-correlation. Note the similarity to the algorithms reviewed in the Introduction (Section 1.4.5, Eqn 1.9), which also maximise the cross-correlation.

5.2.4 Translations

As with the Poisson model, the posterior distribution over the translations factorises over the projection directions:

$$p(t|\theta_{\setminus t}, \mathcal{D}) = \prod_i p(t_i|\theta_{\setminus t}, \mathcal{D}). \tag{5.43}$$

For each t_i, the energy of its posterior is given (up to a constant) by:

$$E(t_i) = -\log p(t_i|\theta_{\setminus t}, \mathcal{D}) \tag{5.44}$$

$$= \frac{\beta}{2}\sum_j (y_{ij} - \hat{y}_{ij})^2. \tag{5.45}$$

We again use the Metropolis-Hastings algorithm described above to sample t_i. The initial value for the scale parameter σ_t was set to 0.1 for the experiments in this chapter.

5.2.5 Noise precision

The noise precision β can be sampled directly from its conditional distribution:

$$p(\beta|\theta_{\setminus\beta}, \mathcal{D}) = \text{Gamma}(\beta|\tilde{a}_\beta, \tilde{b}_\beta), \tag{5.46}$$

where

$$\tilde{a}_\beta = PN + a_\beta \tag{5.47}$$

$$\tilde{b}_\beta = \frac{1}{2}\sum_{ij}(y_{ij} - \hat{y}_{ij})^2 + b_\beta, \tag{5.48}$$

and a_β and b_β are the parameters of the Gamma prior on the noise precision.

As with the weight λ, it would also be possible to use multiple β values, one β_i for every image. This would be appropriate if the noise level varies appreciably between different images. The equations are very similar to the above, and the computation cost should be similar.

5.3 Experiments

The model and algorithm introduced in this chapter were tested on simulated and real data.

As with the algorithm based on the Poisson model, we again divided the algorithm into an initial stage and a refinement stage. During the initial stage, the images were again downsampled to 32×32, and 100 components were used. During the refinement stage, the images were downsampled to 50×50, and 500 components were used. The reported result of the algorithm was once again the posterior mean.

5.3.1 Pol II example

The first example is with a simulated Pol II dataset. The 25 class averages were generated as in Section 3.2, by converting an atomic model to a density map, and projecting it in random directions. The images were of size 50×50, with a pixel size of 4 Å. An extra step was to add i.i.d. Gaussian noise to the projected images. The variance of the noise was selected to obtain a signal-to-noise ratio (SNR) of 10, where the SNR was defined as the ratio of the signal variance to the noise variance.

Fig. 5.1 shows the results of applying the Gaussian noise algorithm to the data. All four initial stage runs converged to roughly the same log-posterior value (Fig. 5.1*B*), and estimated all the rotations correctly.

The result of the algorithm (the posterior mean) agrees well with the reference model at 20 Å, to a resolution of 14.1 Å at FSC=0.5 (Fig. 5.1*D*).

The algorithm took 37 minutes for the initial stage, and 21 minutes for the refinement stage, i.e. less than an hour in total.

5.3.2 APC/C example

The algorithm was tested on the same APC dataset from Section 4.2.3. We used the original 61 class averages, without any deconvolution or masking applied. The images were centered as in Chapter 4: a transformation was applied to each image to move the center of mass to the middle of the image.

Fig. 5.2 shows the results using the centered class averages. Out of the eight separate runs of the initial stage, three estimated all the rotations correctly (Fig. 5.2*A*). The refinement stage converges quickly (Fig. 5.2*B*).

The posterior mean looks similar to the reference (Fig. 5.2*F*), but is somewhat smaller. Due to the difference in size, the two structures agree only to a resolution of 51.5 Å at FSC=0.5, as also seen from the FSC curve (Fig. 5.2*C*).

Comparing the input images to the projections of the final density map (Fig. 5.2*D* and *E*), suggests that the final density map is smaller due to the effect of the CTF on the data combined with the positivity constraint in the model. The CTF causes a ring of negative values around

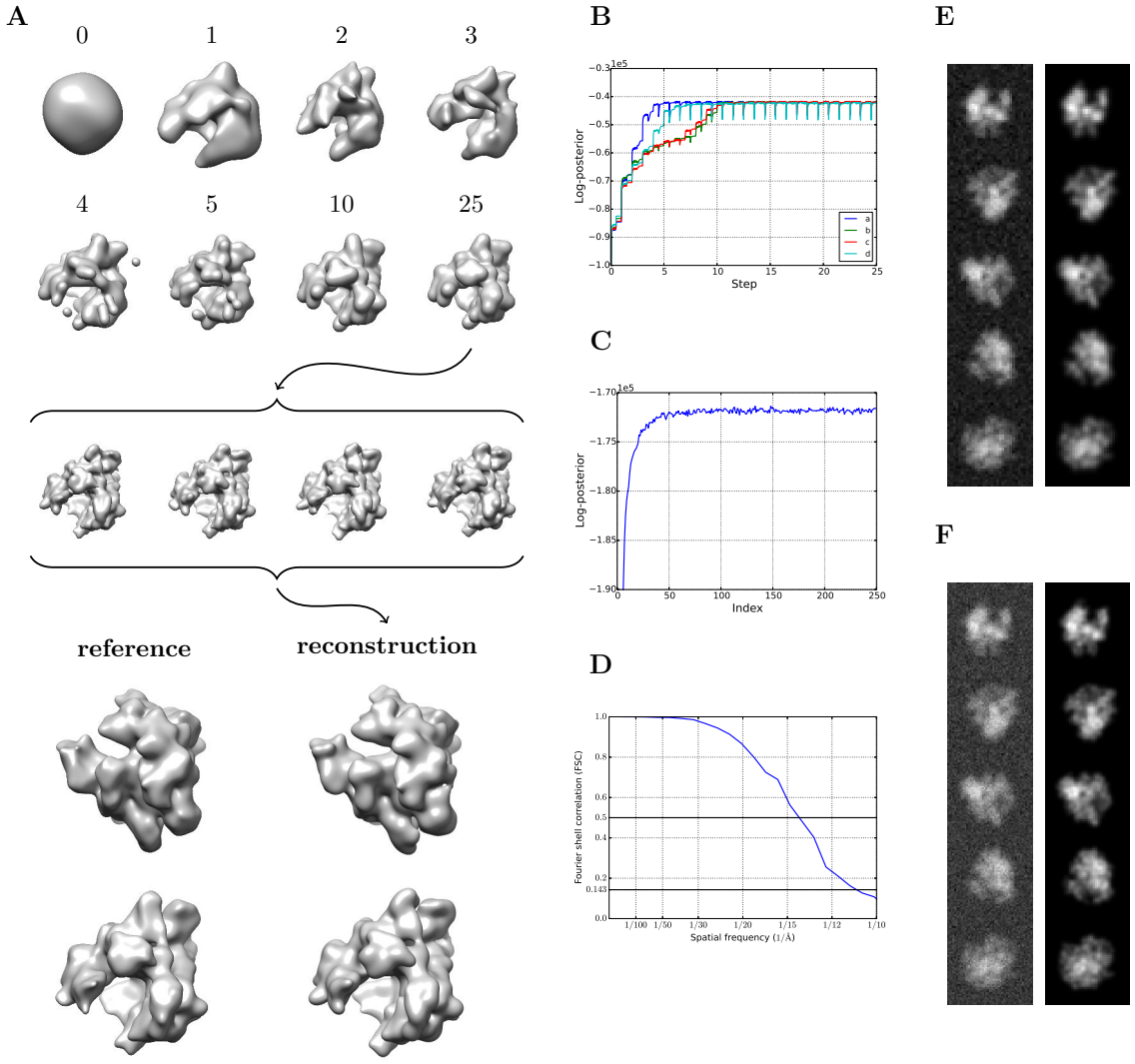

Figure 5.1: Results of the Gaussian noise algorithm for Pol II, using simulated data. (*A*) The first two rows show steps from the initial stage, the next row shows models from the posterior distribution obtained during the refinement stage. At the bottom the posterior mean is compared to the reference model at 20 Å. (*B*) Four different models are inferred during the initial stage, all of which attain roughly the same log-posterior value. (*C*) The log-posterior converges quickly. (*D*) The FSC curve between the posterior mean and the reference shows that they agree to a resolution of 14.1 Å at FSC=0.5. (*E*) Five of the input images to the initial stage (*left*) are compared to the corresponding projections of the final model of the initial stage (*right*). (*F*) The same comparison is done for the refinement stage.

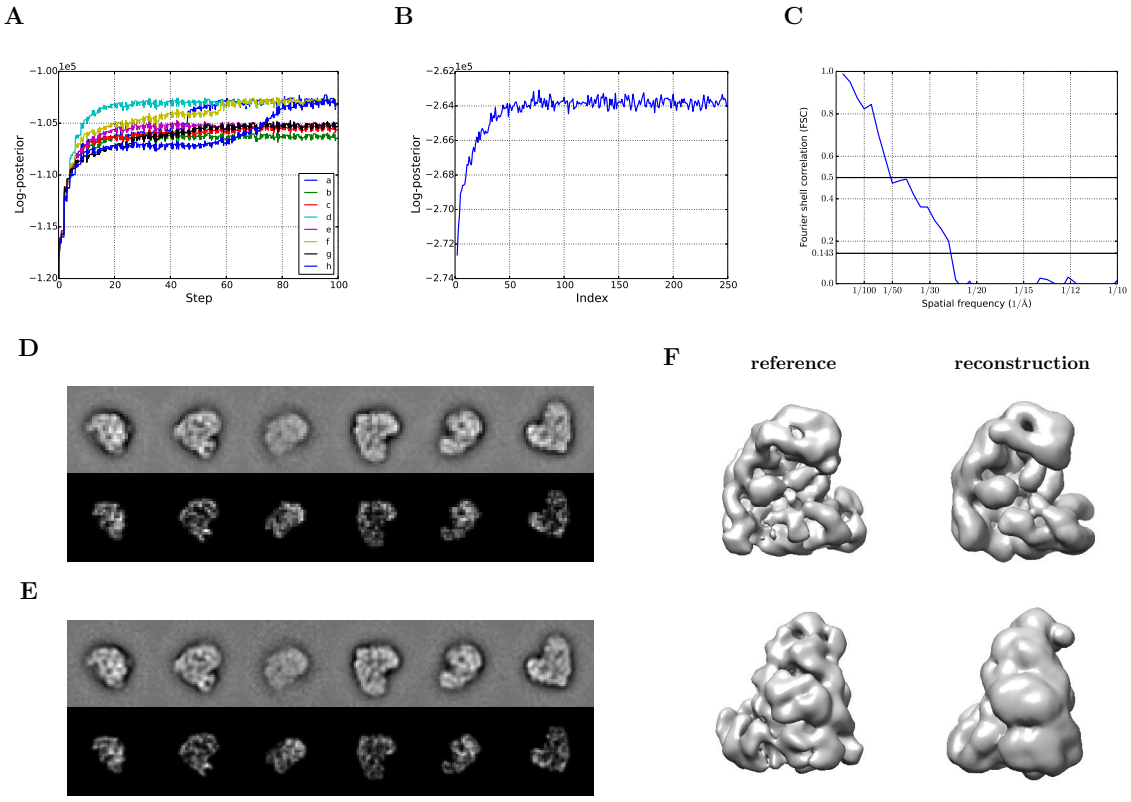

Figure 5.2: Results for APC, using the original class averages, without deconvolution. (*A*) Eight different models are inferred during the initial stage; the one with the highest log-posterior is used for the refinement stage. (*B*) The log-posterior converges during the refinement stage. (*C*) The FSC curve between the result and the reference structures in *F* shows that the two densities agree to a resolution of 51.5 Å at FSC=0.5. (*D*) Six of the input images (*top*) are compared to the corresponding six projections of the final model of the initial stage (*bottom*). (*E*) The same comparison between input images and projections of the final model is also done for the refinement stage. (*F*) The final result is compared to the reference model.

each particle image, which the estimated density map tries to avoid. If negative values were allowed, it would be possible to match the input images more accurately.

The running time for the algorithm was 90 minutes for the initial stage, and 70 minutes for the refinement, i.e. about 2.5 hours in total.

Better results are obtained if we use the same deconvolved images used in Section 4.2.3. Fig. 5.3 shows the results using the 61 deconvolved images. In this case, four of the eight runs estimated all the rotations correctly (Fig. 5.3*A*). The final structure (Fig. 5.3*F*) agrees with the reference to a resolution of 28.9 Å at FSC=0.5, (Fig. 5.3*C* shows the FSC curve). This is a somewhat lower resolution than the value of 24.8 Å obtained with the Poisson algorithm in Section 4.2.3.

The projections of the final density map agree well with the input images (Fig. 5.3*D* and *E*).

The algorithm takes signifantly longer using the deconvolved images than with the original class averages: the initial stage takes about 3 hours 45 minutes, and the refinement stage takes about 70 minutes, i.e. almost 5 hours in total. The reason for this is that the component size is larger when using the deconvolved images, hence evaluating each projection requires more evaluations of the exponential function than with smaller components (see Section 3.5.1).

5.3.3 70S ribsome example

For the second example with real data we used the same deconvolved images used in Section 4.2.1 for the 70S ribosome.

As shown in Figure 5.4, the algorithm obtains a result very similar to the one obtained in Section 4.2.1 by the Poisson algorithm. However, it takes significantly longer. The initial stage took 17 hours, and the refinement stage took 1 hour. Furthermore, only one of the eight runs of the algorithm estimated all the rotations correctly. The corresponding stages with the Poisson algorithm took less than one hour in total.

5.4 Comparison between two algorithms

Using the experimental results obtained above, we can now compare the Gaussian algorithm introduced in this chapter with the Poisson algorithm from earlier in the thesis.

The main difference between the two algorithms is the noise model: the Gaussian algorithm assumes i.i.d Gaussian noise, while the Poisson algorithm is based on an approximation to Poisson noise. One implication is that the non-negativity restriction on the images does not apply to the Gaussian algorithm, only to the Poisson algorithm. This means that the algorithm can be applied directly to class averages obtained by any algorithm; it is not necessary to completely remove the effect of the CTF.

Nevertheless, the APC example (Section 5.3.2) shows that using the original class averages leads to inferior results. This appears to be due to the implied positivity constraint: there is

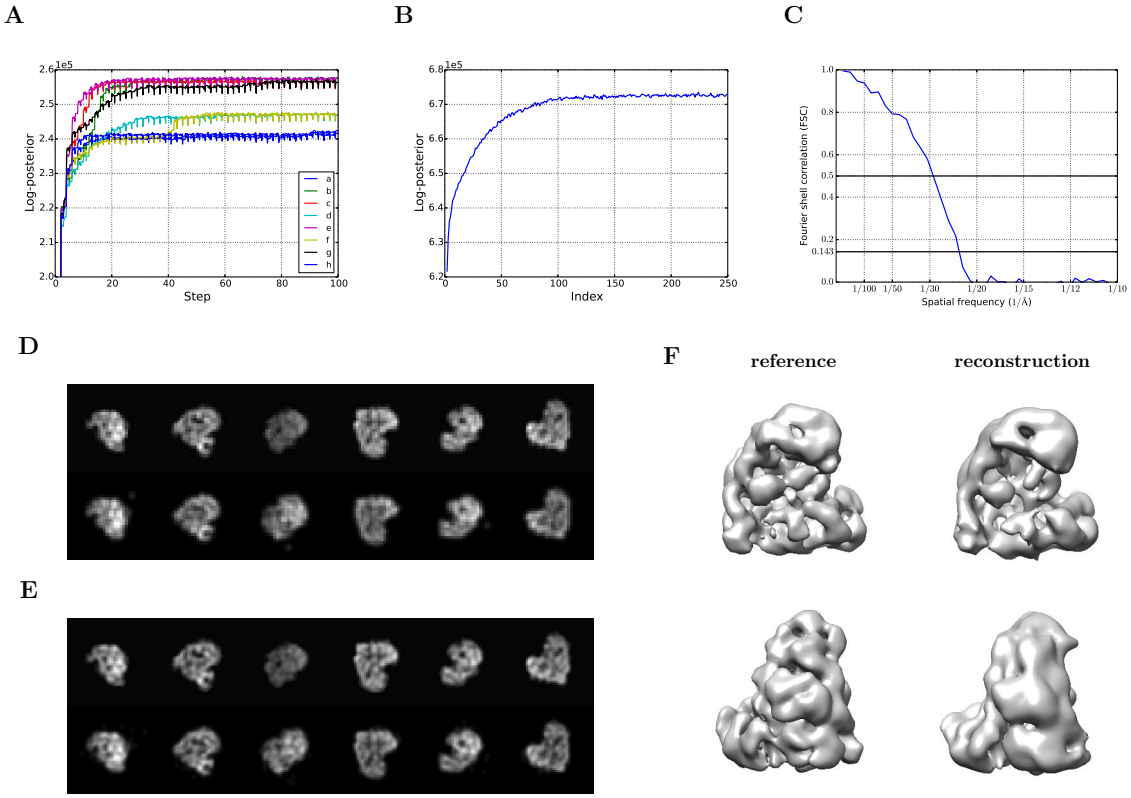

Figure 5.3: Results for APC, using deconvolved class averages. The subplots are analogous to Fig. 5.2.

a single weight λ for all the components, hence the entire density must be either positive or negative. The solution is not to remove the positivity constraint, but rather to improve the forward model. In particular, the next step would be to include the CTF in the forward model.

A major advantage of the Gaussian algorithm is its flexibility. The noise model can be modified to account for coloured noise, for instance. Or the effect of the CTF can be added to the forward model. These modifications would slow down the algorithm, but the required changes to the equations are straight-forward.

Another example of flexibility is that the Gaussian algorithm does not depend on the use of Gaussian components to model the density. The Gaussian functions could be replaced by other blob-like structures, as long as it is possible to compute their projections analytically. An example would be to use the blobs from Marabini et al. (1998).

The Gaussian algorithm is even more flexible than the Poisson algorithm when it comes to incorporating prior information. For instance, in both cases the positions of the components can be constrained, for example to satisfy symmetry constraints or to fix a known subunit. But in the Gaussian case, we could introduce attractive and repulsive forces between components, or require them to lie along a flexible chain to trace a protein backbone. Such modifications are

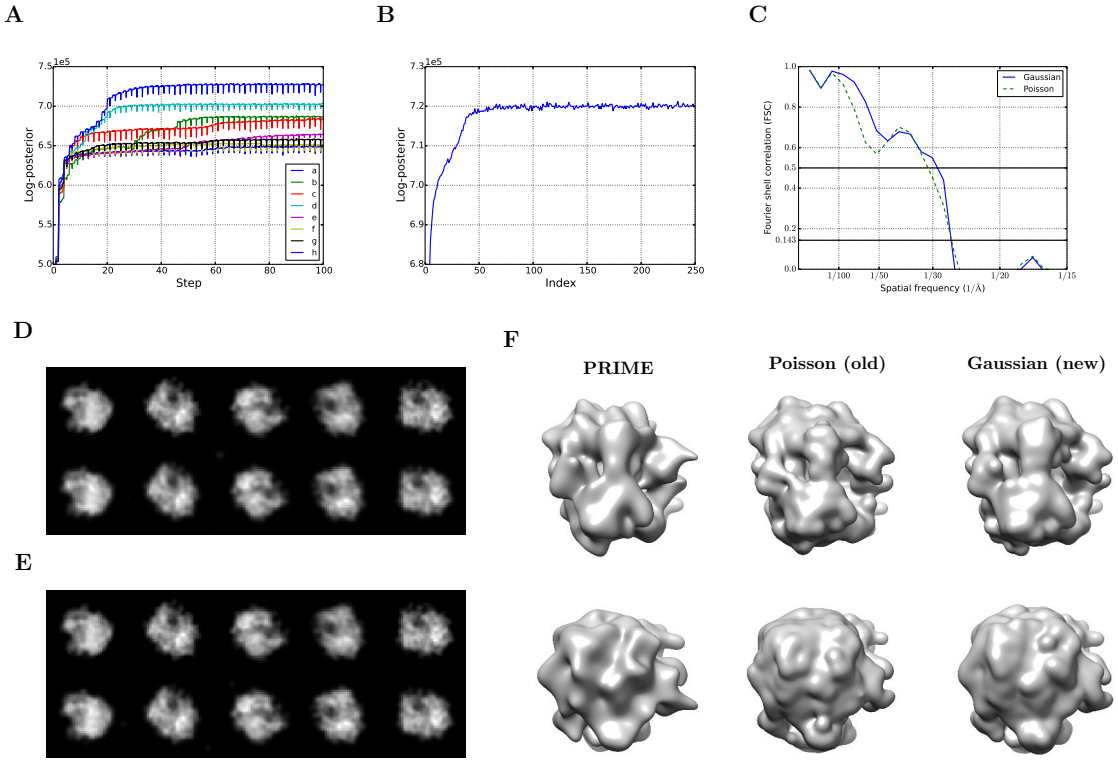

Figure 5.4: Results for 70S ribosome, using deconvolved class averages. The subplots are analogous to Figs. 5.2 and 5.3. (*C*) The FSC curve for the current result (*Gaussian*) is slightly better than the previous result (*Poisson*). It shows that the result agrees with the PRIME reconstruction to a resolution of 29.0 Å at FSC=0.5. (*F*) The current result (*on the right*) looks similar to the previous result (*in the middle*), and both are similar to the result obtained using the PRIME algorithm (*on the left*).

not possible with the Poisson algorithm.

Unfortunately, our current implementation of the Gaussian algorithm is much slower than the Poisson algorithm: for the same 70S ribosome dataset, the Poisson algorithm needed less than an hour for the initial and refinement stages, while the Gaussian algorithm needed around 18 hours.

One reason for the difference is that it is not possible to use the fast approximation to the exponential function (Section 3.5) in the Gaussian algorithm: it is not accurate enough for HMC, which makes up the bulk of the computation.

Another reason is that in the examples with real data, the Gaussian algorithm required more Gibbs sampling steps to converge than the Poisson algorithm.

One advantage of the Gaussian model is that it is easy to anneal the energy function, i.e. to vary the temperature. This could be used in simulated annealing or replica exchange approaches,

which are typically more robust. When annealing the Poisson model by varying the temperature, it is no longer possible to use the Poisson algorithm that we presented.

For the Gaussian algorithm it is not necessary to discretise the image data. The count parameter N_0 that had to be specified for the Poisson case is no longer needed.

Although we presented the model with a single weight, it would be possible to have a separate scale parameter for every image. The noise level could also be estimated for every image individually. This would allow the model to accommodate images with different noise levels, and different scales. It's not clear how to do this with the Poisson approach.

To summarise, although our results show that the Gaussian algorithm is slower than and produces equivalent or inferior results to the Poisson algorithm, its greater flexibility makes it the more promising of the two approaches.

Chapter 6

Conclusions

6.1 Summary

In this thesis, we introduced new algorithms for estimating electron densities from cryo-EM particle images. The algorithms represent the electron densities in a way that is very different from the representations used by other initial model algorithms and reconstruction algorithms. We will summarise the contributions of the thesis by focusing on these two aspects: the model for the electron densities, and the algorithms used to estimate the model parameters.

The electron densities are represented using a Gaussian mixture model, under the restriction that all components should have the same isotropic covariance matrix (i.e. all components must be spherical and of the same size). Other algorithms use a regular grid with voxels (most algorithms) or blobs (Marabini et al. 1998) fixed at the grid points. In contrast, the Gaussian components can be moved around freely by adjusting the coordinates of their means.

We showed that by allowing the positions and size of the components to change, the mixture model representation needs far fewer components to be able to represent an electron density at the resolutions typically used for initial models. As a result, the number of model parameters is greatly reduced, leading to faster and more robust algorithms.

The mixture model representation was shown to have several other benefits, such as making it simpler to compute two-dimensional projections, which are themselves Gaussian mixture models. It also made it possible to use the EM and Gibbs sampling algorithms often used with Gaussian mixture models as a starting point for deriving our first reconstruction algorithm. Another benefit is allowing symmetry constraints to be imposed on the mixture model by making minor adjustments to the algorithm (Section 4.2.2).

We introduced two algorithms for estimating the mixture model parameters, based on different models of how the data is generated. The first algorithm, to which the bulk of the thesis is devoted, assumes that the observed images are histograms of projected points sampled from the mixture model. We argued in Section 2.3 that this is an approximation to evaluating the projected mixture model on the image grid, and adding Poisson noise.

The second algorithm assumes that the images are obtained by evaluating the projected mixture model, and adding Gaussian noise.

The difference between Poisson and Gaussian noise has implications for the images that can be used as input to the corresponding algorithms. In the Poisson case, the images have to be non-negative. This is because images generated under this forward model are always non-negative. In contrast, images generated with Gaussian noise can assume negative values as well. Therefore, any class averages can be used as input to the Gaussian noise based algorithm.

To ensure that the input images to the Poisson algorithm are non-negative, we introduced a deconvolution algorithm that can be combined with an existing class averaging algorithm to fully correct for the CTF. Note that when using the class averaging algorithm of RELION, it is possible to do full CTF correction. In this case our deconvolution algorithm is not needed.

We showed that either algorithm can be used to infer initial models. They were applied both to simulated data, and real data. In addition, we investigated many aspects of the Poisson algorithm in detail in Chapter 3. The Poisson algorithm was also compared to other algorithms (Elmlund et al. 2013; Jaitly et al. 2010) in Chapter 4, and shown to produce either equivalent or better results.

One of its main advantages is that it is several orders of magnitude faster than many competing algorithms. Initial model and reconstruction algorithms using a Bayesian approach tend to be particularly slow. Our Poisson algorithm shows that it is possible to use a Bayesian approach to this problem without being slow. One reason for the speed improvement is that while other Bayesian approaches integrate out the rotations by using a grid covering the entire space of rotations, our approach automatically considers only a small set of rotations which are integrated out using Gibbs sampling.

The Gaussian algorithm was compared to the Poisson algorithm in Chapter 5, and found to be somewhat slower. It also produces inferior results on images that have not been fully CTF corrected. This is probably due to its restriction to positive densities. Other algorithms usually do not impose any positivity constraint on the reconstructed density, which allows its projections to better match the input images. Although the Gaussian algorithm is not as fast as the Poisson algorithm, and produces equivalent or inferior results, its greater flexibility makes it the more promising of the two approaches. See Section 5.4 for a discussion of the differences between the two algorithms.

In addition to the reduced computational complexity, another advantage of both algorithms is robustness through reducing the model complexity. Fewer parameters means that we reduce the number of possible three-dimensional structures that can be represented using our model. During the initial stage of the algorithm where there are only a few large components, it is not possible to represent high-frequency information. Because we are interested in a low-resolution model, this excludes a large number of undesired models from the search space, thereby simplifying the problem and making the algorithm more robust. Some other reconstruction algorithms (both *ab initio* (Elmlund et al. 2013) and refinement (Scheres and Chen 2012)) apply a low-pass filter

to the current volume at every iteration to achieve a similar effect. However in our case this is a property of the model itself.

Both our algorithms use a Bayesian approach. In each case, a well-defined statistical forward model is combined with a prior distribution to obtain a posterior distribution over the model parameters. Samples are then drawn from the posterior distribution using MCMC algorithms, and used to estimate the model parameters. We showed in Section 3.1.5 that the posterior is quite robust to variations in the hyperparameters of the prior.

As explained in Chapter 1, an important advantage of this approach is separating the model parameters (that influence the posterior) from the algorithmic parameters (that have no influence over the posterior distribution itself). Other algorithms often conflate these parameters, and thereby become susceptible to user bias by tweaking parameters.

A common way of representing the results obtained by a Bayesian approach is to estimate the posterior mean and variance. We estimated the mean by evaluating samples (i.e. mixture models) from the posterior on a regular grid, and computing their mean. We showed that this estimate gives better results than when using just a single sample from the posterior (Section 3.2.2).

We also estimated the standard deviation of the posterior about its mean, but for our examples this did not provide meaningful insight. However, for other parameters such as the rotations (Fig. 4.11) and the component size, considering multiple samples from the posterior does give an idea of the precision of the mean estimate.

The initial model problem is non-convex, and as with all other initial model algorithms, we cannot guarantee that our algorithm will always find a good solution. In our case, the Gibbs sampler can sometimes take very long to converge to the region of highest posterior probability. A common approach used by initial model algorithms to alleviate this problem is to repeat the algorithm with many different starting models. However for many algorithms it is difficult to rank the resulting models. The advantage of our approach is that we can compare the models using their log-posterior value, and select the model with the highest log-posterior. We validated this approach for simulated examples where we know the true rotations. In all cases we found that if all rotations were estimated correctly, the resulting log-posterior was higher than if one or more rotations were incorrect. The log-posterior is therefore a very reliable measure of comparison for ranking models.

In addition to the initial model inference algorithms, we also obtained algorithms for fitting mixture models directly to three-dimensional electron densities. For the Poisson case this was shown in Section 3.1, but it can be done assuming Gaussian noise as well. Such an algorithm is useful in many applications independently of reconstruction algorithms (Section 2.1.3).

6.2 Future work

We will first consider possible improvements to the models, and then to the algorithms. Finally we discuss how to modify the model and algorithms to solve related problems.

The number of components is a model parameter that must be specified in advance. We developed guidelines for choosing the number of components in Section 3.1.3, and the algorithm is fast enough to try out different values. But another approach would be to estimate the number of components automatically. One way that this could be done is by using the infinite Gaussian mixture model (Rasmussen 1999).

Currently we use Gaussians as mixture components. This is essential for the Poisson algorithm, but for the Gaussian algorithm we could replace the Gaussian component by another blob-like structure. The motivation would be to speed up the algorithm by not having to evaluate the computationally expensive exponential function. One possible example would be the rotationally symmetric basis function used by the Xmipp software package (Sorzano et al. 2004). Evaluating its two-dimensional projections requires only multiplications, addition and taking the square root.

There are several other improvements which only apply to the Gaussian model. One example was mentioned at the end of Section 5.4: instead of having a single scale parameter (the weight λ) for all images, we could have a separate scale parameter for every image. Similarly, instead of a single noise level for all images, each image could have its own noise level. This would require only minor changes to the algorithm, and should have no effect on the computation time.

The CTF could be made part of the forward model in the Gaussian case. This would be necessary to obtain better results from images that have not been fully CTF corrected (see results in Section 5.3.2). This would increase the computation time. The simplest approach would be to evaluate the PSF on a small grid with the same grid spacing as the image grid, and compute the convolution in the real domain.

Another modification to the Gaussian case is to change the noise model, for example by introducing local correlations between neighbouring pixels. This would also increase the computation time, and might not have a significant effect on the results at the low resolutions used for initial models.

A natural way to improve the algorithms would be to add some form of annealing, such as simulated annealing or replica exchange. For the Poisson model it is not clear how to introduce a temperature parameter, because the standard approach of scaling the log-posterior is computationally intractable in this case. There are different alternatives, but extensive experiments led us to conclude that none of them work. For the Gaussian model the standard approach is easy to implement. Replica exchange would be computationally intensive, but would probably make the algorithm more robust.

Each sampler currently runs only on a single core, even though different samplers may run in parallel. Many of the sampling steps consist of independent computations performed for every image, and could be parallelised. This would allow the algorithm to run on a cluster, and should

be much faster. If replica exchange is used, another way of parallelising the algorithm would be to run different replicas on separate cores.

After computing an initial model, the next step is usually to refine the initial model using the individual particle images instead of the class averages. One research direction would be to modify our approach to be used as a refinement algorithm for computing a high-resolution structure. There would be more components, and more images, which would make a parallel implementation essential. The CTF would also have to be included in the forward model. To include the CTF and use the individual particle images, the Gaussian noise model would be more suited than the Poisson noise model.

An algorithm that can compute high resolution structures could be used with random conical tilt (RCT) data. The experiments from Section 3.2.5 suggest that the Poisson algorithm reduces the artifacts caused by the missing cone.

Our algorithms used only cryo-EM data to estimate the electron density. In many cases additional information about the structure is available. This could be cross-linking/mass-spectrometry data, crystallography data, or knowledge about some subunits. The Bayesian framework makes it possible to combine different sources of information. A research direction could be to extend our approach to use some of these additional sources of data.

Another direction of research would be to take conformational heterogeneity into account. The algorithm would then produce multiple structures instead of only a single one. The simplest approach would be to assume that the structures are independent, as done by ML3D (Scheres et al. 2007). But it would also be possible to model the relations between different conformations as constraints that should be satisfied by the model parameters.

In conclusion, this is a promising direction of research, with many possibilities for future work.

Appendices

Appendix A

The EM algorithm for an isotropic Gaussian mixture model

The expectation maximisation (EM) algorithm was introduced in Section 2.2.2 to fit an isotropic Gaussian mixture model to a point cloud. This appendix provides a detailed derivation of the algorithm.

The data $\mathcal{D}$, model parameters θ, prior distribution $p(\theta)$ and forward model $p(\mathcal{D}|\theta)$ are as described in Section 2.2.1.

The goal of the EM algorithm is to find a local maximum of the posterior distribution $p(\theta|\mathcal{D})$, which is equivalent to finding a local maximum of

$$l(\theta) = \log p(\mathcal{D}|\theta) + \log p(\theta). \tag{A.1}$$

The algorithm is initialised with an estimate of the model parameters θ_0. It then proceeds iteratively to produce a sequence of estimates $\theta_1, \theta_2, \ldots$ such that $l(\theta_{n+1}) \geq l(\theta_n)$, with equality only if $\nabla_{\theta_n} l = \nabla_{\theta_{n+1}} l$.

The idea behind the algorithm is to define a lower bound that depends on the current estimate of the model parameters:

$$Q(\theta; \theta_n) \leq l(\theta). \tag{A.2}$$

And then to obtain the next estimate by maximising the lower bound:

$$\theta_{n+1} := \arg\max_{\theta} Q(\theta; \theta_n). \tag{A.3}$$

Whereas maximising $l(\theta)$ directly is infeasible, the maximum of the lower bound $Q(\theta; \theta_n)$ can be obtained analytically.

The first step is to expand $l(\theta)$. The log-likelihood term in Eqn. A.1 was given in Eqn. 2.32:

$$\log p(\mathcal{D}|\theta) = \sum_{i=1}^{N} \log \sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1}I), \tag{A.4}$$

while the log-prior term follows from Eqns. 2.22, 2.23 and 2.24:

$$\begin{aligned}\log p(\theta) &= \log p(\mu|s) + \log p(s) + \log p(w) && \text{(A.5)}\\ &= \frac{dK}{2}\log s - \frac{rs}{2}\sum_k \|\mu_k\|^2 \\ &\quad + (a-1)\log s - bs \\ &\quad + (\alpha_0 - 1)\sum_{k=1}^{K}\log w_k + c_{\text{prior}}. && \text{(A.6)}\end{aligned}$$

In Eqn. A.6, the terms that are independent of θ are collectively denoted by c_{prior}.

To define the lower bound, let $\tilde{\theta} = \theta_n$ be the current estimate of the model parameters. Then define the assignment parameters

$$r_{ik} := \frac{\tilde{w}_k \mathcal{N}(x_i|\tilde{\mu}_k, \tilde{s}^{-1}I)}{\sum_{l=1}^{K}\tilde{w}_l \mathcal{N}(x_i|\tilde{\mu}_l, \tilde{s}^{-1}I)}. \tag{A.7}$$

First obtain a lower bound on the likelihood (Eqn. A.4) by noting that $\sum_{k=1}^{K} r_{ik} = 1$ and using Jensen's inequality:

$$\begin{aligned}\log p(\mathcal{D}|\theta) &= \sum_{i=1}^{N}\log\sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1}I) && \text{(A.8)}\\ &= \sum_{i=1}^{N}\log\sum_{k=1}^{K} r_{ik}\frac{w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)}{r_{ik}} && \text{(A.9)}\\ &\geq \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\log\frac{w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)}{r_{ik}} && \text{(A.10)}\\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\log w_k \mathcal{N}(x_i|\mu_k, s^{-1}I) + c_{\text{likelihood}}. && \text{(A.11)}\end{aligned}$$

As before, $c_{\text{likelihood}}$ denotes the terms that are independent of θ.

Then add the log-prior to obtain the desired lower bound:

$$\begin{aligned}Q(\theta;\tilde{\theta}) = \frac{dK}{2}\log s - \frac{rs}{2}\sum_{k=1}^{K}\|\mu_k\|^2 + (a-1)\log s - bs \\ + (\alpha_0 - 1)\sum_{k=1}^{K}\log w_k + \sum_{k=1}^{K} N_k \log w_k + \frac{dN}{2}\log s - \frac{s}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|x_i - \mu_k\|^2 \\ + c_{\text{prior}} + c_{\text{likelihood}}. \qquad \text{(A.12)}\end{aligned}$$

The bound is tight at $\theta = \tilde{\theta}$, i.e. $l(\tilde{\theta}) = Q(\tilde{\theta};\tilde{\theta})$. This can be seen by substituting $\tilde{w}_k$, $\tilde{\mu}_k$ and $\tilde{s}$ for w_k, μ_k and s in Eqns. A.9 and A.10, and noting that the $\geq$ sign becomes an $=$ sign in this case.

The final step is to derive the next estimates of the parameters $\hat{\theta} = \theta_{n+1}$ by setting the gradients of the lower bound $Q(\theta; \tilde{\theta})$ to 0 and solving for θ to obtain

$$\hat{\theta} = \arg\max_{\theta} Q(\theta; \tilde{\theta}). \tag{A.13}$$

First the precision:

$$\nabla_s Q(\theta; \tilde{\theta}) = \frac{1}{s}\left(\frac{d(K+N)}{2} + a - 1\right) - \frac{r}{2}\sum_{k=1}^{K}\|\mu_k\|^2 - b - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|x_i - \mu_k\|^2 = 0 \tag{A.14}$$

$$\implies \hat{s} = \frac{(N+K)d + 2(a-1)}{\sum_{i=1}^{N}\sum_{k=1}^{K} r_{ik}\|x_i - \mu_k\|^2 + r\sum_{k=1}^{K}\|\mu_k\|^2 + 2b}. \tag{A.15}$$

For the weights, the constraint $\sum_{k=1}^{K} w_k = 1$ is enforced by using a Lagrange multiplier:

$$Q'(w) = \sum_{k=1}^{N}(N_k + \alpha_0 - 1)\log w_k + \lambda\left(\sum_{k=1}^{K} w_k - 1\right) \tag{A.16}$$

$$\frac{\partial Q'}{\partial w_k} = \frac{N_k + \alpha_0 - 1}{w_k} + \lambda = 0 \tag{A.17}$$

$$\implies w_k = \frac{N_k + \alpha_0 - 1}{-\lambda} \tag{A.18}$$

$$\sum_{k=1}^{K} w_k = 1 \tag{A.19}$$

$$\implies \lambda = -(N + K(\alpha_0 - 1)) \tag{A.20}$$

$$\implies \hat{w}_k = \frac{N_k + \alpha_0 - 1}{N + K(\alpha_0 - 1)}. \tag{A.21}$$

Finally, the means:

$$\nabla_{\mu_k} Q(\theta; \tilde{\theta}) = -rs\mu_k - s\sum_{i=1}^{N} r_{ik}(\mu_k - x_i) = 0 \tag{A.22}$$

$$\implies \hat{\mu}_k = \frac{1}{N_k + r}\sum_{i=1}^{N} r_{ik} x_i. \tag{A.23}$$

The update equations for $\hat{s}$, $\hat{w}_k$ and $\hat{\mu}_k$ are the same as in Eqns. 2.34, 2.38 and 2.39.

The EM algorithm thus consists of two steps: the E-step and the M-step. During the E-step, the lower bound is constructed, which involves the computation of the assignment parameters. During the M-step, the lower bound is optimised.

Appendix B

The Gibbs sampling algorithm for an isotropic Gaussian mixture model

The Gibbs sampling algorithm was introduced in Section 2.2.3 to sample the parameters of a Gaussian mixture model from the posterior distribution $p(\theta|\mathcal{D})$ given a point cloud as data. Below is a detailed derivation of the conditional distributions of the Gibbs sampler.

The data $\mathcal{D}$, model parameters θ, prior distribution $p(\theta)$ and forward model $p(\mathcal{D}|\theta)$ are as described in Section 2.2.1. The extended likelihood was derived in that section (Eqn. 2.16):

$$p(\mathcal{D}, \mathcal{Z}|\theta) = \prod_{i=1}^{N} \prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1}I) \right]^{z_{ik}} . \tag{B.1}$$

Gibbs sampling is used to sample from the full posterior:

$$p(\theta, \mathcal{Z}|\mathcal{D}) \propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\theta). \tag{B.2}$$

The conditional distribution for each parameter is derived by using the above expression, removing terms that do not depend on the parameter, and simplifying. In each case, the result is a well-known distribution.

The first step is to sample the component assignments, z:

$$\begin{aligned} p(z|\mu, s, w, \mathcal{D}) &\propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\theta) && \text{(B.3)} \\ &\propto p(\mathcal{D}, \mathcal{Z}|\theta) && \text{(B.4)} \\ &= \prod_{i=1}^{N} \prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1}I) \right]^{z_{ik}} && \text{(B.5)} \\ p(z|\mu, s, w, \mathcal{D}) &= \prod_{i=1}^{N} p(z_i|\mu, s, w, \mathcal{D}) && \text{(B.6)} \end{aligned}$$

where

$$p(z_i|\mu, s, w, \mathcal{D}) \propto \prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1})\right]^{z_{ik}} \tag{B.7}$$

$$p(z_i|\mu, s, w, \mathcal{D}) = \frac{\prod_{k=1}^{K} \left[w_k \mathcal{N}(x_i|\mu_k, s^{-1})\right]^{z_{ik}}}{\sum_{k=1}^{K} w_k \mathcal{N}(x_i|\mu_k, s^{-1})} \tag{B.8}$$

$$= \prod_{k=1}^{K} r_{ik}^{z_{ik}} \tag{B.9}$$

where as in Eqn. 2.35, r_{ik} is defined as

$$r_{ik} = \frac{w_k \mathcal{N}(x_i|\mu_k, s^{-1}I)}{\sum_l w_l \mathcal{N}(x_i|\mu_l, s^{-1}I)}. \tag{B.10}$$

The next step is the conditional distribution on the means:

$$p(\mu|s, w, z, \mathcal{D}) \propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\theta) \tag{B.11}$$

$$\propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\mu|s) \tag{B.12}$$

$$\propto \prod_{i=1}^{N}\prod_{k=1}^{K} \mathcal{N}(x_i|\mu_k, s^{-1}I)^{z_{ik}} \cdot \prod_{k=1}^{K} \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \tag{B.13}$$

$$= \prod_{k=1}^{K} \left[\prod_{i=1}^{N} \mathcal{N}(x_i|\mu_k, s^{-1}I)^{z_{ik}} \cdot \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I)\right] \tag{B.14}$$

$$p(\mu|s, w, z, \mathcal{D}) = \prod_{k=1}^{K} p(\mu_k|s, w, z, \mathcal{D}) \tag{B.15}$$

where

$$p(\mu_k|s, w, z, \mathcal{D}) = p(\mu_k|s, w, z, \mathcal{D}) \tag{B.16}$$

$$\propto \left[\prod_{i=1}^{N} \mathcal{N}(\mu_k|x_i, (z_{ik}s)^{-1}I)\right] \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I). \tag{B.17}$$

By convention, if $z_{ik} = 0$, the entire Gaussian term of the product in Eqn. B.17 is a constant.

The product of two Gaussians is proportional to another Gaussian, with its natural parameters (Σ^{-1} and $\Sigma^{-1}\mu$) being the sum of the natural parameters of the two Gaussians. This can be used to first simplify the product in Eqn. B.17, and then multiply the final two Gaussians.

$$\prod_{i=1}^{N} \mathcal{N}(\mu_k|x_i, (z_{ik}s)^{-1}I) \propto \mathcal{N}(\mu_k|\tilde{\mu}, \tilde{s}^{-1}I), \tag{B.18}$$

where

$$\tilde{s} = \sum_{i=1}^{N} z_{ik} s = N_k s \tag{B.19}$$

$$\tilde{s}\tilde{\mu} = \sum_{i=1}^{N} z_{ik} s x_i = s \sum_{i=1}^{N} z_{ik} x_i \tag{B.20}$$

$$\tilde{\mu} = \frac{\sum_{i=1}^{N} z_{ik} x_i}{N_k} \tag{B.21}$$

and

$$\mathcal{N}(\mu_k|\tilde{\mu}, \tilde{s}^{-1} I)\mathcal{N}(\mu_k|0, r^{-1}s^{-1} I) \propto \mathcal{N}(\mu_k|\hat{\mu}, \hat{s}^{-1} I), \tag{B.22}$$

where

$$\hat{s} = N_k s + rs \tag{B.23}$$

$$\hat{s}\hat{\mu} = N_k s \frac{\sum_{i=1}^{N} z_{ik} x_i}{N_k} \tag{B.24}$$

$$\hat{\mu} = \frac{\sum_{i=1}^{N} z_{ik} x_i}{N_k + r} \tag{B.25}$$

Thus

$$p(\mu_k|s, w, z, \mathcal{D}) = \mathcal{N}\left(\mu_k \Big| \frac{\sum_{i=1}^{N} z_{ik} x_i}{N_k + r}, \frac{1}{s(N_k + r)} I\right), \tag{B.26}$$

as in Eqn. 2.47.

The precision is next:

$$p(s|\mu, w, z, \mathcal{D}) \propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\theta) \tag{B.27}$$

$$\propto p(\mathcal{D}, \mathcal{Z}|\theta)p(s)p(\mu|s) \tag{B.28}$$

$$\propto \prod_{i=1}^{N}\prod_{k=1}^{K} \mathcal{N}(x_i|\mu_k, s^{-1} I)^{z_{ik}} \cdot s^{a-1} e^{-bs} \cdot \prod_{k=1}^{K} \mathcal{N}(\mu_k|0, r^{-1}s^{-1} I). \tag{B.29}$$

Taking the logarithm on both sides, and ignoring additive constants independent of the precision:

$$\log p(s|\mu, w, z, \mathcal{D}) = \sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik} \log\left[\left(\frac{s}{2\pi}\right)^{d/2} \exp\left(\frac{-s}{2}\|x_i - \mu_k\|^2\right)\right] + (a-1)\log s - bs$$
$$+ \sum_{k=1}^{K} \log s^{d/2} - \sum_{k=1}^{K} \frac{rs}{2}\|\mu_k\|^2 \tag{B.30}$$
$$= \frac{dN}{2}\log s - \frac{s}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik}\|x_i - \mu_k\|^2 + (a-1)\log s - bs$$
$$+ \frac{dK}{2}\log s - sr\sum_{k=1}^{K}\|\mu_k\|^2 \tag{B.31}$$
$$= \left[\frac{d(N+K)}{2} + a - 1\right]\log s - \left[\frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik}\|x_i - \mu_k\|^2 + b - r\sum_{k=1}^{K}\|\mu_k\|^2\right] s \tag{B.32}$$
$$= (\tilde{a} - 1)\log s - \tilde{b}s. \tag{B.33}$$

This implies that

$$p(s|\mu, w, z, \mathcal{D}) = \text{Gamma}(s|\tilde{a}, \tilde{b}) \tag{B.34}$$

where

$$2\tilde{a} = 2a + (N+K)d \tag{B.35}$$
$$2\tilde{b} = 2b + \sum_{i=1}^{N}\sum_{k=1}^{K} z_{ik}\|x_i - \mu_k\|^2 + r\sum_{k=1}^{K}\|\mu_k\|^2 \tag{B.36}$$

as given by Eqn. 2.48.

Finally, the conditional for the weights:

$$p(w|\mu, s, z, \mathcal{D}) \propto p(\mathcal{D}, \mathcal{Z}|\theta)p(w) \tag{B.37}$$
$$\propto \prod_{i=1}^{N}\prod_{k=1}^{K} w_k^{z_{ik}} \prod_{k=1}^{K} w_k^{\alpha_0 - 1} \tag{B.38}$$
$$= \prod_{k=1}^{K} w_k^{N_k + \alpha_0 - 1}. \tag{B.39}$$

The above derivation of the conditional distributions was done with the standard conjugate prior on the means that depends on the precision:

$$p(\mu|s) = \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I). \tag{B.40}$$

An alternative prior which is used for the final algorithm is to remove the dependency on the precision:

$$p(\mu) = \mathcal{N}(\mu_k|0, r^{-1}I). \tag{B.41}$$

The advantage is that it is easier to interpret the hyperparameter r. Using the alternative prior for the Gibbs sampler requires only small changes in the conditionals for the mean and the precision. The conditional on the means becomes:

$$p(\mu_k|s, w, z, \mathcal{D}) = \mathcal{N}\left(\mu_k \Big| \frac{\sum_{i=1}^{N} z_{ik} x_i}{sN_k + r}, \frac{1}{sN_k + r} I\right), \tag{B.42}$$

while the conditional for the precision is:

$$p(s|\mu, w, z, \mathcal{D}) = \text{Gamma}(s|\tilde{a}, \tilde{b}) \tag{B.43}$$

$$2\tilde{a} = 2a + dN \tag{B.44}$$

$$2\tilde{b} = 2b + \sum_{i=1}^{N} \sum_{k=1}^{K} z_{ik} \|x_i - \mu_k\|^2 \tag{B.45}$$

Appendix C

The Gibbs sampling algorithm for a mixture model with projections

This appendix contains the derivation of the Gibbs sampling conditional distributions given in Eqns. 2.85 to 2.94 in Section 2.4.2.

The goal of the algorithm is to sample the parameters of a d-dimensional isotropic mixture model given $(d-1)$-dimensional projections. In the applications, $d = 3$ or $d = 2$.

The data $\mathcal{D}$, latent variables $\mathcal{Z}$, model parameters θ, prior distribution $p(\theta)$ and forward model $p(\mathcal{D}|\theta)$ are as described in Section 2.2.1.

To summarise, the data $\mathcal{D}$ consists of the observed $(d-1)$-dimensional images x^o, where x^o_{ijl} is a $(d-1)$-dimensional point for the ith image, at the center of the jth pixel. There are y_{ij} such points, indexed by l. Since they are all located at the same place, we will sometimes denote x^o_{ijl} simply by x^o_{ij}.

The latent variables $\mathcal{Z}$ consists of the missing components x^m and the component assignments z. Each missing component x^m_{ijl} is a one-dimensional point. The d-dimensional point obtained by appending x^m_{ijl} to the end of x^o_{ijl} is denoted by x_{ijl}. There is a component assignment variable z_{ijl} for each point x_{ijl}. It is a length K vector with $K-1$ zeros and 1 one. We will often not be interested in exactly which of the y_{ij} points (at the jth pixel of image i) was assigned to which component, but rather how many points were assigned to each component. This is given by the component assignment variables z_{ij}, where

$$z_{ijk} = \sum_{l=1}^{y_{ij}} z_{ijlk}. \tag{C.1}$$

Each z_{ij} is also a length K vector, but the entries (z_{ijk}) can be any non-negative integers.

The model parameters θ consist of the means μ, the precision s and the weights w of the mixture model, and the rotations R and translations t.

Gibbs sampling is used to sample from the full posterior:

$$p(\theta, \mathcal{Z}|\mathcal{D}) \propto p(\mathcal{D}, \mathcal{Z}|\theta)p(\theta), \tag{C.2}$$

where the full likelihood is

$$p(\mathcal{D}, \mathcal{Z}|\theta) = p(x^o, x^m, z|\mu, s, w, R, t) \tag{C.3}$$
$$= p(x, z|\mu, s, w, R, t), \tag{C.4}$$

and the prior is

$$p(\theta) \propto p(\mu|s)p(s)p(w)p(R)p(t). \tag{C.5}$$

The expressions for the full likelihood and the prior are:

$$p(x, z|\mu, s, w, R, t) = \prod_{ijlk} \left[w_k \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)\right]^{z_{ijlk}} \tag{C.6}$$
$$p(\mu|s) = \prod_k \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \tag{C.7}$$
$$p(s) \propto s^{a-1}e^{-bs} \tag{C.8}$$
$$p(w) \propto \prod_k w_k^{\alpha_0 - 1} \tag{C.9}$$
$$p(R) \propto 1 \tag{C.10}$$
$$p(t) = \prod_i \mathcal{N}(t_i|0, r_t^{-1}). \tag{C.11}$$

For some computations it will be useful to factorise the full likelihood:

$$p(\mathcal{D}, \mathcal{Z}|\theta) = p(\mathcal{D}|\mathcal{Z}, \theta)p(\mathcal{Z}|\theta) \tag{C.12}$$
$$p(x^o, x^m, z|\mu, s, w, R, t) = p(x^o|x^m, z, u, s, w, R, t)p(x^m|z, \mu, s, w, R, t)p(z|\mu, s, w, R, t) \tag{C.13}$$
$$= p(x^o|z, \mu, s, R, t)p(x^m|z, \mu, s, R, t)p(z|w), \tag{C.14}$$

where the variables that are not involved in the expressions have been removed in the last line. These expressions are:

$$p(x^o|z, \mu, s, R, t) = \prod_{ijlk} \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I)^{z_{ijlk}} \tag{C.15}$$
$$p(x^m|z, \mu, s, R, t) = \prod_{ijlk} \mathcal{N}(x^m_{ijl}|P_m(R_i\mu_k + t_i), s^{-1}I)^{z_{ijlk}} \tag{C.16}$$
$$p(z|w) \propto \prod_{ijlk} w_k^{z_{ijlk}} = \prod_{ijk} w_k^{z_{ijk}}. \tag{C.17}$$

The full posterior is

$$p(\theta, \mathcal{Z}|\mathcal{D}) = p(\mu, s, w, R, t, z, x^m|x^o). \tag{C.18}$$

To use Gibbs sampling to sample from the full posterior, the variables are partitioned into groups, and the conditional distribution for each group of variables conditioned on the rest is computed. In our case, z and x^m are grouped together, followed by μ, s, w, R, and finally t.

Assignments and missing components

The latent variables z and x^m are sampled using ancestral sampling. In other words, the combined conditional is factorised as

$$p(z, x^m|x^o, \mu, s, w, R, t) = p(x^m|z, \mu, s, w, R, t)p(z|x^o, \mu, s, w, R, t), \tag{C.19}$$

and z is sampled from $p(z|x^o, \mu, s, w, R, t)$ followed by x^m from $p(x^m|z, \mu, s, w, R, t)$.

To obtain expressions for the latent variable distributions, note that

$$\begin{aligned} p(z, x^m|x^o, \mu, s, w, R, t) &\propto p(x^o, x^m, z|\mu, s, w, R, t) && \text{(C.20)} \\ &= p(x^m|z, \mu, s, R, t)p(x^o|z, \mu, s, R, t)p(z|w). && \text{(C.21)} \end{aligned}$$

By comparing Eqns. C.19 and C.21, it follows that

$$\begin{aligned} p(z|x^o, u, s, w, R, t) &\propto p(x^o|z, \mu, s, R, t)p(z|w) && \text{(C.22)} \\ &\propto \prod_{ijlk} \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I)^{z_{ijlk}} \prod_{ijlk} w_k^{z_{ijlk}} && \text{(C.23)} \\ &= \prod_{ijlk} \left[w_k \mathcal{N}(x^o_{ijl}|P_o(R_i\mu_k + t_i), s^{-1}I)\right]^{z_{ijlk}} && \text{(C.24)} \\ &= \prod_{ijk} \left[w_k \mathcal{N}(x^o_{ij}|P_o(R_i\mu_k + t_i), s^{-1}I)\right]^{z_{ijk}} && \text{(C.25)} \\ &= \prod_{ij} p(z_{ij}|x^o, \mu, s, w, R, t), && \text{(C.26)} \end{aligned}$$

where

$$p(z_{ij}|x^o, \mu, s, w, R, t) \propto \prod_k \left[w_k \mathcal{N}(x^o_{ij}|P_o(R_i\mu_k + t_i), s^{-1}I)\right]^{z_{ijk}}. \tag{C.27}$$

In words, the d-dimensional mixture is transformed using R_i and t_i, and projected to $(d-1)$-dimensions. For every pixel, the assignments z_{ij} is sampled from a multinomial distribution obtained by evaluating each of the projected components at the pixel.

We could also sample each z_{ijl} (from a categorical distribution), but all subsequent sampling steps will depend on z_{ijl} via z_{ij}. The latent variables themselves will be discarded: they are only needed to sample the model parameters. Therefore it is sufficient to sample z_{ij}. This significantly reduces the number of computations.

The next step is to sample the missing components:

$$\begin{aligned} p(x^m|z, \mu, s, w, R, t) &\propto \prod_{ijlk} \mathcal{N}(x^m_{ijl}|P_m(R_i\mu_k + t_i), s^{-1})^{z_{ijlk}} && \text{(C.28)} \\ &= \prod_{ijl} p(x^m_{ijl}|z, \mu, s, w, R, t), && \text{(C.29)} \end{aligned}$$

where

$$p(x_{ijl}^m|z,\mu,s,w,R,t) = \prod_k \mathcal{N}(x_{ijl}^m|P_m(R_i\mu_k + t_i), s^{-1})^{z_{ijlk}}. \tag{C.30}$$

In words: the missing component of each point is sampled around the last component of the mean of the mixture component to which the point was assigned.

As with the component assignments, the individual missing components are not needed in the subsequent sampling of model parameters. What is needed for each image is only the mean and variance of the missing components of those points that were assigned to the same mixture component. New notation is introduced to describe variables that are proportional to the mean and variance of the missing components:

$$\nu_{ik}^m := \sum_{jl} z_{ijlk} x_{ijl}^m \tag{C.31}$$

$$\tau_{ik} := \sum_{jl} z_{ijlk}(x_{ijl}^m - P_m(R_i\mu_k + t_i))^2 \tag{C.32}$$

These can be sampled from a Gaussian and a chi-squared distribution respectively:

$$\nu_{ik}^m \sim \mathcal{N}(\nu_{ik}^m|N_{ik}P_m(R_i\mu_k + t_i), N_{ik}s^{-1}) \tag{C.33}$$

$$\tau_{ik} \sim \frac{1}{s}\chi^2(N_{ik}), \tag{C.34}$$

where $N_{ik} = \sum_{jl} z_{ijlk}$. By sampling ν_{ik}^m and τ_{ik} instead of x_{ijl}^m, many unnecessary calculations are avoided.

It will also be convenient to add the following definitions:

$$\nu_{ik}^o := \sum_{jl} z_{ijlk} x_{ijl}^o \tag{C.35}$$

$$\nu_{ik} := \begin{bmatrix} \nu_{ik}^o \\ \nu_{ik}^m \end{bmatrix}. \tag{C.36}$$

Weights

The next parameter to sample is the weights:

$$p(w|x^o, x^m, z, \mu, s, R, t) \propto p(z|w)p(w) \tag{C.37}$$

$$\propto \prod_{ijlk} w_k^{z_{ijlk}} \cdot \prod_k w_k^{\alpha_0 - 1} \tag{C.38}$$

$$= \prod_k w_k^{N_k + \alpha_0 - 1}, \tag{C.39}$$

where $N_k = \sum_{ijl} z_{ijlk}$.

Means

The conditional for the means is proportional to a product of Gaussians, and therefore itself a Gaussian. Recall that the product of two Gaussians is proportional to another Gaussian, with its natural parameters (Σ^{-1} and $\Sigma^{-1}\mu$) being the sum of the natural parameters of the two Gaussians.

$$p(\mu|s,z,s,R,t) \propto p(x,z|\mu,s,w,Rt)p(\mu|s) \tag{C.40}$$

$$\propto \prod_{ijlk} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)^{z_{ijlk}} \cdot \prod_k \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \tag{C.41}$$

$$= \prod_k p(\mu_k|x,z,s,R,t), \tag{C.42}$$

where

$$p(\mu_k|x,z,s,R,t) \propto \prod_{ijl} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I) \cdot \mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \tag{C.43}$$

The first product can be simplified:

$$\prod_{ijl} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)^{z_{ijlk}} = \prod_{ijl} \mathcal{N}(R_i\mu_k|x_{ijl} - t_i, s^{-1}I)^{z_{ijlk}} \tag{C.44}$$

$$= \prod_{ijl} \mathcal{N}(\mu_k|R_i^T(x_{ijl} - t_i), z_{ijlk}s^{-1}I) \tag{C.45}$$

$$= \mathcal{N}(\mu_k|\tilde{\mu}, \tilde{s}^{-1}I), \tag{C.46}$$

where

$$\tilde{s} = \sum_{ijl} z_{ijlk}s = N_k s \tag{C.47}$$

$$\tilde{s}\tilde{\mu} = s\sum_{ijl} z_{ijlk}R_i^T(x_{ijl} - t_i) \tag{C.48}$$

$$= s\sum_i [R_i^T(\sum_{jl} z_{ijlk}x_{ijl} - \sum_{jl} z_{ijlk}t_i)] \tag{C.49}$$

$$= s\sum_i R_i^T(\nu_{ik} - N_{ik}t_i) \tag{C.50}$$

$$\tilde{\mu} = \frac{1}{N_k}\sum_i R_i^T(\nu_{ik} - N_{ik}t_i). \tag{C.51}$$

Next, compute the product with the prior:

$$\mathcal{N}(\mu_k|\tilde{\mu}, \tilde{s}^{-1}I)\mathcal{N}(\mu_k|0, r^{-1}s^{-1}I) \propto \mathcal{N}(\mu_k|\hat{\mu}_k, \hat{s}^{-1}I), \tag{C.52}$$

where

$$\hat{s} = N_k s + rs = s(N_k + r) \tag{C.53}$$

$$\hat{s}\hat{\mu} = s \sum_i R_i^T (\nu_{ik} - N_{ik} t_i) \tag{C.54}$$

$$\hat{\mu} = \frac{1}{N_k + r} \sum_i R_i^T (\nu_{ik} - N_{ik} t_i). \tag{C.55}$$

The final conditional on the means becomes:

$$p(\mu_k | x, z, s, R, t) = \mathcal{N}(\mu_k | \frac{1}{N_k + r} \sum_i R_i^T (\nu_{ik} - N_{ik} t_i), \frac{1}{s(N_k + r)} I). \tag{C.56}$$

Precision

The next step is the precision, which is easier to compute using logarithms. After taking logarithms, equality signs are taken to mean equality up to an additive constant independent in s.

$$p(s|x, z, \mu, r, t) \propto p(x, z|\mu, s, w, R, t) p(\mu|s) p(s) \tag{C.57}$$

$$= \prod_{ijlk} \mathcal{N}(x_{ijl} | R_i \mu_k + t_i, s^{-1} I)^{z_{ijlk}} \cdot \mathcal{N}(\mu_k | 0, r^{-1} s^{-1} I) \cdot s^{a-1} e^{-bs} \tag{C.58}$$

$$\log p(s|x, z, \mu, r, t) = \sum_{ijlk} z_{ijlk} \log \left[\left(\frac{s}{2\pi} \right)^{d/2} \exp \left(\frac{-s}{2} \|R_i \mu_k + t_i - x_{ijl}\|^2 \right) \right]$$

$$+ \sum_k \log \left[\left(\frac{s}{2\pi} \right)^{d/2} \exp \left(\frac{-rs}{2} \|\mu_K\|^2 \right) \right] + (a-1) \log s - bs \tag{C.59}$$

$$= \sum_{ijlk} z_{ijlk} \frac{d}{2} \log s - \frac{s}{2} \sum_{ijlk} z_{ijlk} \|R_i \mu_k + t_i - x_{ijl}\|^2$$

$$+ \frac{dK}{2} \log s - \frac{rs}{2} \sum_k \|\mu_k\|^2 + (a-1) \log s - bs \tag{C.60}$$

$$= -\frac{s}{2} \sum_{ijlk} z_{ijlk} \left((P_m(R_i \mu_k + t_i) - x_{ijl}^m)^2 + \|P_o(R_i \mu_k + t_i) - x_{ijl}^o\|^2 \right)$$

$$+ \left[\frac{d(K+N)}{2} + a - 1 \right] \log s - s \left[\frac{r}{2} \sum_k \|\mu_k\|^2 + b \right] \tag{C.61}$$

$$= (\tilde{a} - 1) \log s - \tilde{b} s, \tag{C.62}$$

where

$$2\tilde{a} = 2a + (N + K)d \tag{C.63}$$

$$2\tilde{b} = 2b + \sum_{ik} \tau_{ik} + \sum_{ijk} z_{ijk} \|P_o(R_i \mu_k + t_i) - x_{ij}^o\|^2 + r \sum_k \|\mu_k\|^2. \tag{C.64}$$

Thus s is sampled from a Gamma distribution with parameters $\tilde{a}$ and $\tilde{b}$.

Rotations

The rotations are next:

$$p(R|x,z,\mu,s,w,t) \propto p(x,z|\mu,s,w,R,t)p(R) \tag{C.65}$$
$$\propto \prod_{ijlk} \left[w_k \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)\right]^{z_{ijlk}} \tag{C.66}$$
$$= \prod_i p(R_i|x,z,\mu,s,w,t), \tag{C.67}$$

where

$$p(R_i|x,z,\mu,s,w,t) \propto \prod_{jlk} \exp\left[-\frac{s}{2}\|x_{ijl} - (R_i\mu_k + t_i)\|^2\right]^{z_{ijlk}} \tag{C.68}$$
$$= \prod_{jlk} \left(\exp\left[-\frac{s}{2}(x_{ijl} - t_i - R_i\mu_k)^T(x_{ijl} - t_i - R_i\mu_k)\right]\right)^{z_{ijlk}} \tag{C.69}$$
$$\propto \prod_{jlk} (\exp[s(x_{ijl} - t_i)^T R_i\mu_k])^{z_{ijlk}} \tag{C.70}$$
$$= \exp \sum_{jlk} z_{ijlk} s \operatorname{tr}[(x_{ijl} - t_i)^T R_i\mu_k] \tag{C.71}$$
$$= \exp \sum_{jlk} z_{ijlk} s \operatorname{tr}[\mu_k(x_{ijl} - t_i)^T R_i] \tag{C.72}$$
$$= \exp \operatorname{tr}(A_i^T R_i), \tag{C.73}$$

where

$$A_i = s \sum_{jlk} z_{ijlk}(x_{ijl} - t_i)\mu_k^T \tag{C.74}$$
$$= s \sum_k (\nu_{ik} - N_{ik}t_i)\mu_k^T. \tag{C.75}$$

This conditional for the rotations is the only one that is not of a common form. Therefore we implemented the algorithm described by Habeck (2009).

Translations

Finally, the translations:

$$p(t|x,z,\mu,s,w,R) \propto p(x,z|\mu,s,w,R,t)p(t) \tag{C.76}$$
$$\propto \prod_{ijlk} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)^{z_{ijlk}} \cdot \prod_i \mathcal{N}(t_i|0, r_t^{-1}) \tag{C.77}$$
$$= \prod_i p(t_i|x,z,\mu,s,w,R), \tag{C.78}$$

where

$$p(t_i|x,z,\mu,s,w,R) \propto \prod_{jlk} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)^{z_{ijlk}} \cdot \mathcal{N}(t_i|0, r_t^{-1}). \tag{C.79}$$

The products of the Gaussians are computed as above in the case of the means:

$$\prod_{jlk} \mathcal{N}(x_{ijl}|R_i\mu_k + t_i, s^{-1}I)^{z_{ijlk}} \propto \prod_{jlk} \mathcal{N}(t_i|x_{ijl} - R_i\mu_k, (z_{ijlk}s)^{-1}I) \tag{C.80}$$

$$\propto \mathcal{N}(t_i|\tilde{\mu}, \tilde{s}^{-1}I), \tag{C.81}$$

where

$$\tilde{s} = \sum_{jlk} z_{ijlk}s = N_i s \tag{C.82}$$

$$\tilde{s}\tilde{\mu} = \sum_{jlk} z_{ijlk}s(x_{ijl} - R_i\mu_k) = s\sum_k (\nu_{ik} - N_{ik}R_i\mu_k) \tag{C.83}$$

$$\tilde{\mu} = \frac{1}{N_i}\sum_k (\nu_{ik} - N_{ik}R_i\mu_k) \tag{C.84}$$

and for the final product

$$\mathcal{N}(t_i|\tilde{\mu}, \tilde{s}^{-1}I) \cdot \mathcal{N}(t_i|0, r_t^{-1}I) = \mathcal{N}(t_i|\hat{\mu}, \hat{s}^{-1}I), \tag{C.85}$$

where

$$\hat{s} = N_i s + r_t \tag{C.86}$$

$$\hat{s}\hat{\mu} = s\sum_k (\nu_{ik} - N_{ik}R_i\mu_k) \tag{C.87}$$

$$\hat{\mu} = \frac{s\sum_k (\nu_{ik} - N_{ik}R_i\mu_k)}{N_i s + r_t}. \tag{C.88}$$

The resulting conditional distribution for the translations is

$$p(t_i|x,z,\mu,s,w,R) = \mathcal{N}\left(t_i \middle| \frac{s\sum_k (\nu_{ik} - N_{ik}R_i\mu_k)}{N_i s + r_t}, \frac{1}{N_i s + r_t} I\right). \tag{C.89}$$

If we let r_t tend to 0, the conditional simplifies to

$$p(t_i|x,z,\mu,s,w,R) = \mathcal{N}\left(t_i \middle| \frac{1}{N_i}\sum_k (\nu_{ik} - N_{ik}R_i\mu_k), \frac{1}{N_i s} I\right). \tag{C.90}$$

This can be interpreted as using an improper prior on the translations, which approaches a uniform distribution. In practise, the influence of r_t would be far outweighed by the data, so it makes sense to remove it in this way.

Appendix D

Derivations for model with Gaussian noise

Here we derive the equations used in Chapter 5. These include the gradients of the negative log-posterior needed for sampling the means and rotations, and the marginal and conditional distributions needed for sampling the weight and component precision.

The negative log-posterior is refered to as the energy E. It is a scalar function of the parameters θ, where

$$\theta = \{\mu, \lambda, s, \beta, R, t\}, \tag{D.1}$$

and the energy is given by

$$E(\theta) = -\log p(\mathcal{D}|\theta) - \log p(\theta) \tag{D.2}$$

$$-\log p(\mathcal{D}|\theta) = -PN \log \beta + \frac{\beta}{2} \sum_{ij} (y_{ij} - \hat{y}_{ij})^2 \tag{D.3}$$

$$-\log p(\theta) = \frac{r}{2} \sum_{k=1}^{K} \mu_k^T \mu_k - \frac{a_\beta - 1}{\beta} + b_\beta. \tag{D.4}$$

In the above expression for the energy, the value of the ith projection at the jth pixel is given by

$$\hat{y}_{ij} = \lambda \sum_k v_{ijk}, \tag{D.5}$$

where

$$v_{ijk} := \exp\{-\frac{s}{2} \|d_{ijk}\|^2\} \tag{D.6}$$

$$d_{ijk} := x_{ij} - (P_o R_i \mu_k + t_i). \tag{D.7}$$

We will also denote the unscaled projection by

$$\tilde{y}_{ij} := \sum_k v_{ijk}, \tag{D.8}$$

i.e. $\hat{y}_{ij} = \lambda \tilde{y}_{ij}$.

Means gradient

We start by computing the gradients relative to the means. The terms of the energy function that depend on the means are:

$$E(\mu) = \frac{\beta}{2}\sum_{ij}(y_{ij} - \hat{y}_{ij})^2 + \frac{r}{2}\sum_{k}\mu_k^T\mu_k. \tag{D.9}$$

The gradient of the energy E w.r.t. to the mean μ_k can be derived as follows:

$$\frac{\partial E}{\partial \mu_k} = \beta\sum_{ij}(y_{ij} - \hat{y}_{ij})\left(-\frac{\partial \hat{y}_{ij}}{\partial \mu_k}\right) + r\mu_k \tag{D.10}$$

$$\frac{\partial \hat{y}_{ij}}{\partial \mu_k} = \lambda\frac{\partial v_{ijk}}{\partial \mu_k} \tag{D.11}$$

$$\frac{\partial v_{ijk}}{\partial \mu_k} = v_{ijk}\left(-\frac{s}{2}\right)\frac{\partial}{\partial \mu_k}\left[d_{ijk}^T d_{ijk}\right] \tag{D.12}$$

$$= s v_{ijk} R_i^T P_o^T d_{ijk}. \tag{D.13}$$

Substituting back gives

$$\frac{\partial E}{\partial \mu_k} = -\beta\lambda s\sum_{ij} v_{ijk}(y_{ij} - \hat{y}_{ij})R_i^T P_o^T d_{ijk} + r\mu_k. \tag{D.14}$$

Rotations gradient

For a given rotation R_i, the energy function is:

$$E(R_i) = \frac{\beta}{2}\sum_{j}(y_{ij} - \hat{y}_{ij})^2. \tag{D.15}$$

The gradient is derived as follows:

$$\frac{\partial E}{\partial R_i} = -\beta\sum_{j}(y_{ij} - \hat{y}_{ij})\frac{\partial \hat{y}_{ij}}{\partial R_i} \tag{D.16}$$

$$\frac{\partial \hat{y}_{ij}}{\partial R_i} = \lambda\sum_{k}\frac{\partial v_{ijk}}{\partial R_i} \tag{D.17}$$

$$\frac{\partial v_{ijk}}{\partial R_i} = v_{ijk}\left(-\frac{s}{2}\right)\frac{\partial}{\partial R_i}\|d_{ijk}\|^2 \tag{D.18}$$

$$\frac{\partial}{\partial R_i}\|d_{ijk}\|^2 = \frac{\partial}{\partial R_i}(x_{ij} - P_o R_i \mu_k - t_i)^T(x_{ij} - P_o R_i \mu_k - t_i) \tag{D.19}$$

$$= \frac{\partial}{\partial R_i}(\mu_k^T R_i^T P_o^T P_o R_i \mu_k - 2(x_{ij} - t_i)^T P_o R_i \mu_k) \tag{D.20}$$

$$= 2P_o^T P_o R_i \mu_k \mu_k^T - 2P_o^T(x_{ij} - t_i)\mu_k^T \tag{D.21}$$

$$= -2P_o^T d_{ijk}\mu_k^T. \tag{D.22}$$

The gradients in Eqn. D.21 follow from Petersen and Pedersen (2008, Eqns. 62, 74).

Substituting back, we obtain the final gradient:

$$\frac{\partial E}{\partial R_i} = -\beta\lambda s \sum_{jk} (y_{ij} - \hat{y}_{ij}) v_{ijk} P_o^T d_{ijk} \mu_k^T. \tag{D.23}$$

Component precision marginal

To sample the precision s in Section 5.2.3, we have to marginalise out the weight λ from the posterior

$$q(\lambda, s) = p(\lambda, s|\theta_{\setminus\lambda,s}\mathcal{D}) \tag{D.24}$$

to obtain $q(s)$. The energy function corresponding to $q(\lambda, s)$ is given (up to a constant) by

$$\begin{aligned}
E(\lambda, s) &= -\log q(\lambda, s) && \text{(D.25)}\\
&= \frac{\beta}{2} \sum_{ij} (y_{ij} - \lambda \tilde{y}_{ij})^2 && \text{(D.26)}\\
&= \frac{\beta}{2} \sum_{ij} (\lambda^2 \tilde{y}_{ij}^2 - 2\lambda y_{ij} \tilde{y}_{ij} + y_{ij}^2) && \text{(D.27)}\\
&= \left[\frac{\beta}{2} \sum_{ij} \tilde{y}_{ij}^2\right] \lambda^2 + \left[\beta \sum_{ij} y_{ij} \tilde{y}_{ij}\right] \lambda + \frac{\beta}{2} \sum_{ij} y_{ij}^2 && \text{(D.28)}\\
&= a\lambda^2 + b\lambda + c, && \text{(D.29)}
\end{aligned}$$

where the coefficients a, b and c are functions of s. By completing the square, we obtain:

$$\begin{aligned}
E(\lambda, s) &= a(\lambda + \frac{b}{2a})^2 + \frac{b^2 - 4ac}{4a} && \text{(D.30)}\\
&= d(\lambda - \mu)^2 + f, && \text{(D.31)}
\end{aligned}$$

where $d = a$, $\mu = -b/(2a)$ and $f = -(b^2 - 4ac)/(4a)$.

By using the identity

$$\int_{\mathbb{R}} \sqrt{\frac{t}{2\pi}} e^{-\frac{t}{2}(\lambda-\mu)^2} d\lambda = 1 \tag{D.32}$$

with $d = t/2$, we obtain

$$\begin{aligned}
q(s) &= \int_{\mathbb{R}} q(\lambda, s) d\lambda && \text{(D.33)}\\
&= \int_{\mathbb{R}} e^{-E(\lambda,s)} d\lambda && \text{(D.34)}\\
&= \int_{\mathbb{R}} e^{-d(\lambda-\mu)^2} e^{-f} d\lambda && \text{(D.35)}\\
&= \sqrt{\frac{\pi}{a}} \exp\left[\frac{b^2 - 4ac}{4a}\right]. && \text{(D.36)}
\end{aligned}$$

The corresponding energy function is given by

$$\tilde{E}(s) = -\log q(s) \tag{D.37}$$

$$= \frac{1}{2}\log a - \frac{(b^2 - 4ac)}{4a} \tag{D.38}$$

$$= \frac{1}{2}\log\left[\frac{\beta}{2}\sum_{ij}\tilde{y}_{ij}^2\right] - \frac{\beta}{2}\left[\frac{\left(\sum_{ij} y_{ij}\tilde{y}_{ij}\right)^2}{\sum_{ij}\tilde{y}_{ij}^2} - \sum_{ij} y_{ij}^2\right]. \tag{D.39}$$

Weight conditional

From Eqn. D.31 it follows that the conditional of λ given s has a quadratic energy function:

$$E(\lambda) = -\log q(\lambda|s) \tag{D.40}$$

$$= d(\lambda - \mu)^2 + f. \tag{D.41}$$

Thus $q(\lambda|s)$ is Gaussian, with mean μ_λ and precision s_λ, where

$$\mu_\lambda = \mu = -b/(2a) = \frac{\sum_{ij} y_{ij}\tilde{y}_{ij}}{\sum_{ij}\tilde{y}_{ij}^2} \tag{D.42}$$

$$s_\lambda = 2d = 2a = \beta\sum_{ij}\tilde{y}_{ij}^2. \tag{D.43}$$

Bibliography

Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer, p. 738.

Byrd, R. et al. (1995). 'A limited memory algorithm for bound constrained optimization'. In: *SIAM Journal on Scientific Computing*.

Cadez, I. V. et al. (2002). 'Maximum likelihood estimation of mixture densities for binned and truncated multivariate data'. In: *Machine Learning* 47.1, pp. 7–34.

Dempster, A., N. Laird and D. Rubin (1977). 'Maximum likelihood from incomplete data via the EM algorithm'. In: *Journal of the Royal Statistical Society* 39, pp. 1–38.

Duane, S. et al. (1987). 'Hybrid Monte Carlo'. In: *Physics Letters B* 195.2, pp. 216–222.

Elmlund, D. and H. Elmlund (2012). 'SIMPLE: Software for ab initio reconstruction of heterogeneous single-particles'. In: *Journal of Structural Biology* 180, pp. 420–427.

Elmlund, D., R. Davis and H. Elmlund (2010). 'Ab initio structure determination from electron microscopic images of single molecules coexisting in different functional states'. In: *Structure* 18.7, pp. 777–786.

Elmlund, H., D. Elmlund and S. Bengio (2013). 'PRIME: probabilistic initial 3D model generation for single-particle cryo-electron microscopy.' In: *Structure (London, England : 1993)* 21.8, pp. 1299–306.

Elmlund, H. et al. (2008). 'A New Cryo-EM Single-Particle Ab Initio Reconstruction Method Visualizes Secondary Structure Elements in an ATP-Fueled AAA+ Motor'. In: *Journal of Molecular Biology* 375.4, pp. 934–947.

Frank, J. (2006). *Three-Dimensional Electron Microscopy of Macromolecular Assemblies*. Oxford University Press, p. 432.

— (2014). *70S E. coli ribosome*. `http://www.ebi.ac.uk/pdbe/emdb/test_data.html`. Accessed December 2014.

Frye, J. et al. (2013). 'Electron microscopy structure of human APC/C(CDH1)-EMI1 reveals multimodal mechanism of E3 ligase shutdown.' In: *Nature structural & molecular biology* 20.7, pp. 827–35.

Gallego, G. and A. Yezzi (2014). 'A compact formula for the derivative of a 3-D rotation in exponential coordinates'. In: *Journal of Mathematical Imaging and Vision* none, pp. 1–7.

Geman, S. and D. Geman (1984). 'Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images'. In: *Pattern Analysis and Machine Intelligence* 6.6, pp. 721–741.

Ghahramani, Z. and M. I. Jordan (1995). *Learning from incomplete data*. Tech. rep. MIT.

Gilbert, P. (1972). 'Iterative methods for the three-dimensional reconstruction of an object from projections'. In: *Journal of theoretical biology* 36.1, pp. 105–117.

Gordon, R., R. Bender and G. T. Herman (1970). 'Algebraic reconstruction techniques (ART) for three-dimensional electron microscopy and X-ray photography'. In: *Journal of theoretical Biology* 29.3, pp. 471–481.

Habeck, M. (2009). 'Generation of three-dimensional random rotations in fitting and matching problems'. In: *Computational Statistics* 24.4, pp. 719–731.

Jaitly, N. et al. (2010). 'A Bayesian method for 3D macromolecular structure inference using class average images from single particle electron microscopy.' In: *Bioinformatics* 26.19, pp. 2406–15.

Jin, Q. and C. Sorzano (2014). 'Iterative Elastic 3D-to-2D Alignment Method Using Normal Modes for Studying Structural Dynamics of Large Macromolecular Complexes'. In: *Structure* 22.3, pp. 496–506.

Kawabata, T. (2008). 'Multiple subunit fitting into a low-resolution density map of a macromolecular complex using a gaussian mixture model.' In: *Biophysical journal* 95.10, pp. 4643–58.

Ludtke, S. J. (2014). *EMAN2.1 Workshops, Summer 2014*. `http://blake.bcm.edu/emanwiki/Ws2014`. Accessed December 2014.

Lyumkis, D. et al. (2013). 'Optimod – An automated approach for constructing and optimizing initial models for single-particle electron microscopy'. In: *Journal of Structural Biology* 184.3, pp. 417–426.

Marabini, R., G. Herman and J. Carazo (1998). '3D reconstruction in electron microscopy using ART with smooth spherically symmetric volume elements (blobs).' In: *Ultramicroscopy* 72.1-2, pp. 53–65.

McAdams, A. et al. (2011). 'Computing the Singular Value Decomposition of 3x3 matrices with minimal branching and elementary floating point operations'. In: *University of Wisconsin - Madison technical report TR1690* May.

McLachlan, G. and P. Jones (1988). 'Fitting mixture models to grouped and truncated data via the EM algorithm'. In: *Biometrics* 44.2, pp. 571–578.

Miles, R. E. (1965). 'On random rotations in $\mathbb{R}^3$'. In: *Biometrika* 52.3, pp. 636–639.

Neal, R. M. (2011). 'MCMC using Hamiltonian dynamics'. In: *Handbook of Markov Chain Monte Carlo* 2.

Nocedal, J. (1980). 'Updating quasi-Newton matrices with limited storage'. In: *Mathematics of computation* 35.151, pp. 773–782.

Nogales-Cadenas, R. and S. Jonic (2013). '3DEM Loupe: analysis of macromolecular dynamics using structures from electron microscopy'. In: *Nucleic acids research* 41.41, pp. 363–7.

Penczek, P., R. Grassucci and J. Frank (1994). 'The ribosome at improved resolution: new techniques for merging and orientation refinement in 3D cryo-electron microscopy of biological particles.' In: *Ultramicroscopy* 53.3, pp. 251–70.

Penczek, P., J. Zhu and J. Frank (1996). 'A common-lines based method for determining orientations for N greater than 3 particle projections simultaneously'. In: *Ultramicroscopy* 63.3, pp. 205–218.

Penczek, P. A. (2010a). 'Chapter One-Fundamentals of Three-Dimensional Reconstruction from Projections'. In: *Methods in enzymology* 482, pp. 1–33.

— (2010b). 'Chapter Two-Image Restoration in Cryo-Electron Microscopy'. In: *Methods in enzymology* 482, pp. 35–72.

Penczek, P. A., R. Renka and H. Schomberg (2004). 'Gridding-based direct Fourier inversion of the three-dimensional ray transform'. In: *JOSA A* 21.4, pp. 499–509.

Petersen, K. B., M. S. Pedersen et al. (2008). 'The matrix cookbook'. In: *Technical University of Denmark* 7, p. 15.

Radermacher, M. (1988). 'Three-dimensional reconstruction of single particles from random and nonrandom tilt series'. In: *Journal of electron microscopy technique* 9.4, pp. 359–394.

Rasmussen, C. E. (1999). 'The infinite Gaussian mixture model.' In: *NIPS*. Vol. 12, pp. 554–560.

Sanz-Garcia, E., A. Stewart and D. Belnap (2010). 'The random-model method enables ab initio 3D reconstruction of asymmetric particles and determination of particle symmetry.' In: *Journal of structural biology* 171.2, pp. 216–22.

Scheres, S. (2012a). 'A Bayesian view on cryo-EM structure determination.' In: *Journal of molecular biology* 415.2, pp. 406–18.

— (2012b). 'RELION: implementation of a Bayesian approach to cryo-EM structure determination.' In: *Journal of structural biology* 180.3, pp. 519–30.

Scheres, S. and S. Chen (2012). 'Prevention of overfitting in cryo-EM structure determination.' In: *Nature methods* 9.9, pp. 853–4.

Scheres, S. et al. (2007). 'Disentangling conformational states of macromolecules in 3D-EM through likelihood optimization'. In: *Nature Methods* 4.1, pp. 27–29.

Schraudolph, N. (1999). 'A fast, compact approximation of the exponential function'. In: *Neural Computation* 862, pp. 853–862.

Singer, A. and Y. Shkolnisky (2011). 'Three-Dimensional Structure Determination from Common Lines in Cryo-EM by Eigenvectors and Semidefinite Programming().' In: *SIAM journal on imaging sciences* 4.2, pp. 543–572.

Singer, A. et al. (2010). 'Detecting consistent common lines in cryo-EM by voting'. In: *Journal of structural biology* 169.3, pp. 312–322.

Sorzano, C. et al. (2004). 'XMIPP: a new generation of an open-source image processing package for electron microscopy.' In: *Journal of structural biology* 148.2, pp. 194–204.

Sorzano, C. et al. (2015). 'A statistical approach to the initial volume problem in Single Particle Analysis by Electron Microscopy'. In: *Journal of structural biology* 189.3, pp. 213–219.

Tang, G. et al. (2007). 'EMAN2: An extensible image processing suite for electron microscopy'. In: *Journal of Structural Biology* 157.1, pp. 38–46.

van Heel, M. (1987). 'Angular reconstitution: a posteriori assignment of projection directions for 3D reconstruction'. In: *Ultramicroscopy* 21.

Vargas, J. et al. (2014). 'Efficient initial volume determination from electron microscopy images of single particles.' In: *Bioinformatics (Oxford, England)* d, pp. 1–8.

Wang, L., A. Singer and Z. Wen (2013). 'Orientation determination of cryo-EM images using least unsquared deviations'. In: *SIAM journal on imaging sciences* 6.4, pp. 2450–2483.

Yan, X. et al. (2007). 'Ab initio random model method facilitates 3D reconstruction of icosahedral particles'. In: *Journal of structural biology* 157.1, pp. 211–225.

Yang, C., E. Ng and P. Penczek (2005). 'Unified 3-D structure and projection orientation refinement using quasi-Newton algorithm.' In: *Journal of structural biology* 149.1, pp. 53–64.

Zhao, Z. and A. Singer (2014). 'Rotationally invariant image representation for viewing direction classification in cryo-EM'. In: *Journal of Structural Biology* 186.1, pp. 153–166.

Curriculum Vitae

Personal details

Name:	Paul Joubert
Date of birth:	01.12.1981
Place of birth:	Cape Town, South Africa

Education

02/2000 - 08/2005:	University of Stellenbosch, South Africa
	Bachelor of Engineering (Electric & Electronic), 05.12.2003
	Bachelor of Science with honours (Mathematics), 03.12.2004
	Master of Science (Mathematics), 14.04.2006
09/2005 - 06/2006:	Université de Bordeaux I, Erasmus Mundus (ALGANT)
08/2006 - 06/2007:	Leiden University, ALGANT (Algebra, Geometry and Number Theory)
	Master of Science (Mathematics), 28.06.2007
09/2010 - 12/2012:	MPI for Intelligent Systems, Tübingen
	Department of Empirical Inference (Schölkopf)
01/2013 - 03/2016:	PhD student in Computer Science, Georg-August-Universität Göttingen

Work experience

12/2003 - 01/2004:	CSIR (Council for Scientific and Industrial Research)
	Vacation project, modelling of underwater communications
07/2007 - 07/2010:	IMS (Institute for Mining Seismology)
	Seismology research and software development

Publications

Joubert, P. (2006) *Geometric actions of the absolute Galois group.* Masters thesis, Stellenbosch University, http://hdl.handle.net/10019.1/2508.

Joubert, P. (2007) *The topology of isolated singularities on complex hypersurfaces.* Masters thesis, Leiden University, http://www.math.leidenuniv.nl/nl/theses/81

Joubert, P. (2010) *Microseismic monitoring of hydraulic fractures in block cave mines.* Proceedings of the Second International Symposium on Block and Sublevel Caving, pp. 685-690.

Joubert, P. and Habeck, M. (2015) *Bayesian inference of initial models in cryo-electron microscopy using pseudo-atoms.* Biophysical Journal 108.5, pp. 1165-75.

Awards

2003	ECSA medal for best final-year student at engineering faculty
2004	Dean's medal for best final-year student at science faculty

Skills

Languages	Afrikaans (native), English (C2), German (B2), French (B1), Russian (B1)
Programming	Python, Cython, Java, Matlab