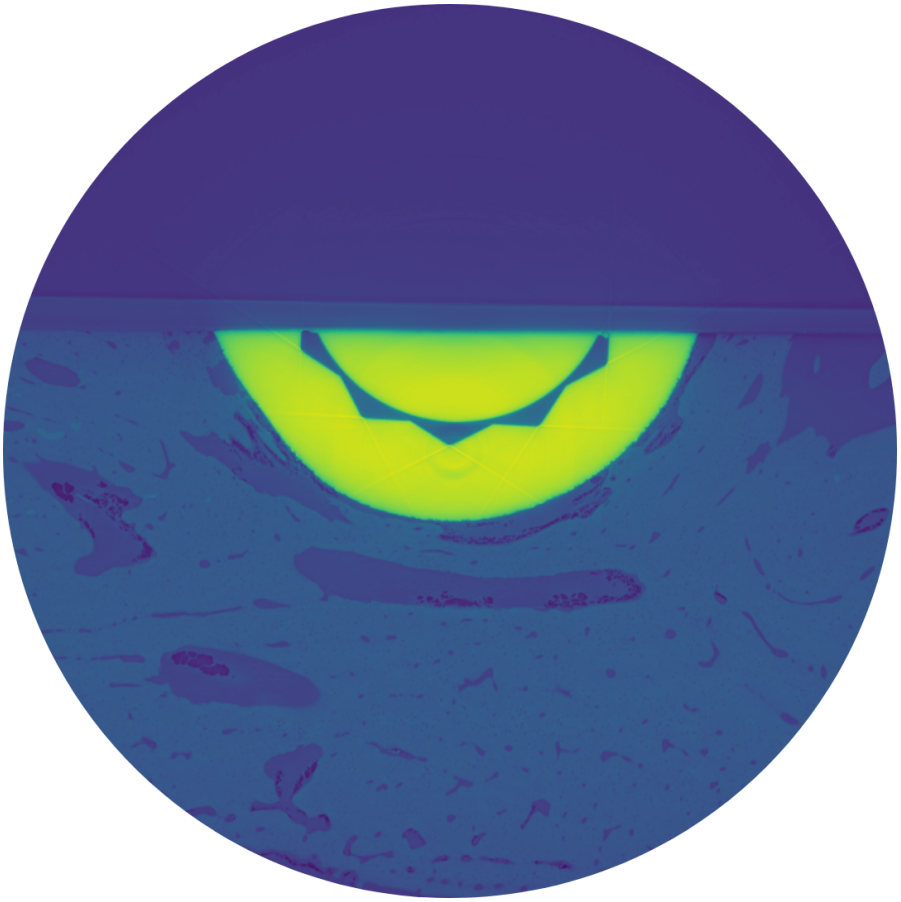
MATHEMATICAL MODELLING OF BONE MICROSTRUCTURE

JAMES-IESTYN WATKIN



Thesis for the M.Sc. degree in physics

August 6 2021

James-Iestyn Watkin: *Mathematical Modelling of Bone Microstructure,*
Thesis for the M.Sc. degree in physics, © August 6 2021

SUPERVISORS:
James Avery

DEPARTMENTS:
The Niels Bohr Institute
Faculty of Natural Sciences
University of Copenhagen

LOCATION:
Copenhagen

DATE OF SUBMISSION:
August 6 2021

I love my girlfriend. Like a lot. She's amaxing.
— James-Iestyn Watkin I swear

ABSTRACT

Micrometer-scale resolution synchrotron micro-CT 3D images provide some of the highest quality and clearest three dimensional images of the microstructure of bone that we can create. However, due to physical distortion effects there are problems when trying to systematically make quantitative analysis of bone based on these images. If we could remove those distortion effects such an analysis would be possible.

In this thesis, I have designed and developed algorithms to automatically analyze these images to reduce and remove these distortions. I have done this by combining spatial dependencies in the probability distributions of these images; to segment them into air, blood vessels, bone and titanium. This research has shown that whilst it is possible to reduce the effect of the distortions, it is very challenging to remove them entirely, though this should give a benefit to the MAXIBONE project in general, as it brings the automatic and systematic analysis of bone even closer.

Further work is required to automatically analyse the segmented tomograms to quantify bone-contact and blood flow to the implants around which the new bone grows.

FOREWORD AND ACKNOWLEDGMENTS

Due to time constraints, both aspects of the project and this thesis remain incomplete at the time of submission. In addition due to a number of technical issues the font size of some of the figures is very small, please zoom in to fully appreciate them. I hope you can find some method to my madness within.

I would like to thank my partner, Karen Alavi, for putting up with me throughout this process. I would like to thank my son, Milo, for being a talkative new addition to this journey. Finally I would like to thank my supervisor, James, for the patience and support he has shown. Without any of these people I owuld not have been able to ramble on for countable pages in this document .

CONTENTS

LIST OF FIGURES

INTRODUCTION

In this thesis I aimed to develop and implement algorithms to automatically analyze micrometer-scale resolution synchrotron micro-CT 3D images of bone, which has replaced standard histomorphometry for the analysis of bone micro-architecture. This technique is a non-destructive way of obtaining three-dimensional images with a high resolution across all spatial axes, whilst it avoids physical sample preparation. This analysis used a combination of classical automatic image analysis techniques with computational physics.

Due to the extremely large size of the micrometer-scale resolution synchrotron micro-CT data sets, of 160GB per sample and the full dataset of dozens of terabytes, standard Python-based imaging software was unsuitable for use. My focus was on efficiently and robustly implementing selected parts of the functionality needed for analysis, so that it is able to deal with these enormous data sets efficiently. This work could then be integrated into the MAXIBONE project.

I have implemented a new segmentation method developed for micrometer-scale resolution synchrotron micro-CT 3D images that aims to reverse physical distortion effects present by combining spatially dependent probability distributions over several axes, and apply it to segment into air, titanium, bone, and blood vessels a data set of 35 goat mandible bone samples. This sample consisted of seven goats, each with four different experimental methods for bone regeneration and one control.

In the next part of this thesis I will present the theory behind synchrotron X-ray computed tomography as well as a brief introduction to structure and composition of bone. Following that, I present the method and algorithms I have used to automatically analyze micrometer-scale resolution synchrotron micro-CT 3D images of bone. This then leads on to a presentation of the results of this investigation with a discussion on those results. Finally there is a chapter containing my conclusions and an indication of further work in this area.

Part I

THEORY

1

SYNCHROTRON X-RAY TOMOGRAPHY

In order to image bones for medical purposes X-ray radiography is the easiest and oldest form of examination. Though this method of bone examination has some limitations. X-ray radiography only allows a two dimensional flattened projection of the three dimensional object of investigation to be rendered. On the other hand X-ray computed tomography circumvents this projection problem present in radiography by imaging 'slices' in the object. X-ray computed tomography is the method of imaging an object by taking a series of two dimensional slices, or images, of an object using penetrative radiation, in this case X-rays. The image produced by this technique is called a tomogram. A tomogram is made up of many voxels; which constitute a notional three dimensional space in the image. In this case the data each voxel contains is the intensity of the X-ray energy photons that were incident on the object, have then been transmitted by it and finally detected by our detection mechanism.

When an electrically charged particle is subjected to an acceleration it will emit photons as electromagnetic waves. In a synchrotron, a circular accelerator, electrons are accelerated using magnetic fields. This acceleration is caused by a radial force which attracts these electrons to the centre of the ring shaped accelerator. The photons emitted by this process are called synchrotron radiation. There is a wide spectrum of possible radiation produced by a synchrotron, from the X-ray all the way to infrared. The X-ray range, a frequency of between 10^{16} to 10^{20} Hz, can only be reached when the energy of the electrons is very high, in the order of 10^{-10} J per electron.

The synchrotron produces a beam of X-ray photons to penetrate the object under examination. As an X-ray beam passes through the object it is attenuated. The rate of attenuation is dependent on the energy of the X-ray and the material that it passes through. In order to gather useful information from this attenuation it is important to use a beam of suitable frequency so that the attenuation is governed by the photoelectric effect. This then means the intensity of the X-rays detected after passing through the object is given as:

$$I = I_0 \exp\left(\frac{-\mu}{\rho}\right)\rho x \tag{1.1}$$

Here I_0 is the intensity of the incident photon and I is the intensity of the photon after it travels a distance, x, through the material, which has a linear attenuation coefficient of μ.

$\frac{-\mu}{\rho}$ is known as the mass attenuation coefficient ($cm^{-2}\ g^{-1}$) of the object. Rewriting this equation gives mass attenuation coefficient as a

function of the density of the material, the distance it travels through the material as well as the incident and transmitted photon's intensity.

$$\frac{-\mu}{\rho} = \frac{1}{\rho x} \ln\left(\frac{I}{Io}\right) \tag{1.2}$$

However, the Beer-Labmert law, which relates the attenuation of light to properties of the material it travels through and is expressed in equation [2 above], is only completely obeyed when the beam of photons are all of the same exact frequency. In real world applications of X-ray tomography the beam will contain a range of frequencies. These frequencies are not attenuated uniformly when passing through the same material, this phenomenon is called beam hardening.

Beam hardening happens when the X-ray beam passes through a dense material in the object of the tomogram. The X-ray beam will consist of a series of photons of different frequency distributed around some target frequency. When this beam passes through the object there is a selective attenuation of the beam as the photons of lower frequency, and thus energy, are more easily attenuated and can even be completely absorbed when passing through the object. When the tomogram is then constructed from the detection of the transmitted X-rays, if we assume the attenuation of the beam is linear throughout the object then edges of the object will have a higher intensity, appearing brighter, even though the object may be made of an entirely homogeneous material. This means that beam hardening will skew results of a tomogram which will lead to an incorrect analysis of an object's density and thus composition.

Another effect that will affect the tomogram's representation of an object's density is reflection. At very high contrast interfaces between one material and another inside the object, i.e. titanium to human tissue, there will be reflection due to high refraction of the X-ray beam as it passes from the dense medium to another that is much less dense. This X-ray scattering at the interface will lead to areas in the less dense material around the interface appearing much more dense than they are in reality. The value of these voxels is higher and would appear to correspond to a denser material than in other locations in the object, which are actually the same material but lie further away from these interfaces.

Both the X-ray beam and the individual materials in the object being imaged are not uniform. In the case of the beam of X-ray frequency photons produced by the synchrotron, which are of a very high quality, they still have a distribution of frequencies centred around a target frequency. Additionally as mentioned the individual materials in the object being imaged are not uniform. Not only are there impurities which we cannot explain with our model as we don't know their composition but the materials themselves will produce slightly different attenuation due to their structure on a nanometre scale, which is the wavelength of the X-ray photons in the beam. This means that we can use gaussian distributions to model the intensity of the X-rays

detected from each different material, as the spread of intensity values from the transmitted photons should follow a gaussian distribution. Of course there are instances where this is not completely the case, due to some of the effects mentioned earlier like beam hardening and reflection. The 'errors' in the tomogram slices are systematic and so I was able to account for them in the techniques used to mitigate their effect.

2

COMPOSITION OF BONE

There are a number of roles that bone fulfils in an organism. The structure of bone is different depending on the scale of observation. At a micrometer scale there are two main regimes to this structure that can be discerned; cortical and cancellous bone. These are a dense external shell and a porous inner material composed of thin trabeculae, which are an irregular network of spaces inside the bone. For the purposes of this analysis we have focused on images and analysis of cortical bone. Cortical bone plays a primary role in the strength of the bone, and the fragility of the bone ís dependent on its microstructure (Peter and PEYRIN, 2011).

Bone tissue, otherwise known as osseous tissue, is a type of specialised connective tissue. The composition of bone tissue is a mix of water, collagen and a mineral (Hydroxyapatite (HA) crystals). Whilst the collagen gives bone its toughness the hydroxyapatite crystals are responsible for bone's characteristic stiffness. The internal structure of bone resembles a honeycomb, it is a lattice of different types of bone cells. These are osteoblasts, osteocytes and osteoclasts. The osteoblasts are responsible for the formation of mineral structures in the bone. Once an osteoblast has produced so much bone matrix around itself that it has become trapped it is then referred to as an osteocyte. Osteocytes remain in contact, and communication, with other bone cells. The third type of bone cell, osteocytes, are responsible for breaking down the mineral structure of the bone. Bone is continually altering its shape and structure due to the activity of the osteoblasts and osteoclasts.

Though this makes up the majority of the structure of bone, there are other types of cells and tissue present. These include: bone marrow, nerves and blood vessels. As cells in the bone require nutrients and oxygen, like other cells in the human body, the blood vessels are woven throughout the bone structure. The blood vessels in bone can be extremely narrow, making them very small artifacts to detect even with a micrometer-scale resolution synchrotron micro-Computed Tomography 3D image. It is also important to note that even at this scale individual osteocytes still cannot be distinguished on a micrometer-scale resolution synchrotron micro-Computed Tomography 3D image. At different scales imaging techniques will provide different types of information. In order to study bone microstructure resolutions of between 5 μm - 10 μm are necessary, as we have used in this project. To examine the ultra-structural level a submicrometer resolution is needed, and for a study of the crystalline structure a nanometre scale of observation is appropriate.

Part II

METHOD

3

PROGRAMMING LANGUAGE

The extremely large size of the micrometer-scale resolution synchrotron micro-CT 3D images data sets, around 160GB per sample with dataset in tens of terabytes, meant that standard Python-based imaging software was unsuitable for use. In order to maintain easy readability Python was used throughout this project to efficiently and robustly implement selected parts of the functionality needed for analysis, so that it is able to deal with these extremely large data sets efficiently, and then so that it could be integrated it into the common source code for the MAXIBONE project.

4

FINDING THE MATERIALS IN A SINGLE TOMOGRAM SLICE

First I looked at a single slice of a tomogram, along a single plane. In this case I define this plane as the x-y plane. The objective is to find, using the intensity of the voxel at a specific location, the material present at that location. We already know the different materials which are present in the tomogram and those are; bone, blood vessel, metal, resin/air and any other impurities in the bone. The important materials we wish to identify and clearly distinguish between are the blood vessels and the bone. The first step to identify the different materials present is to take a histogram of the intenisites present in our tomogram slice.

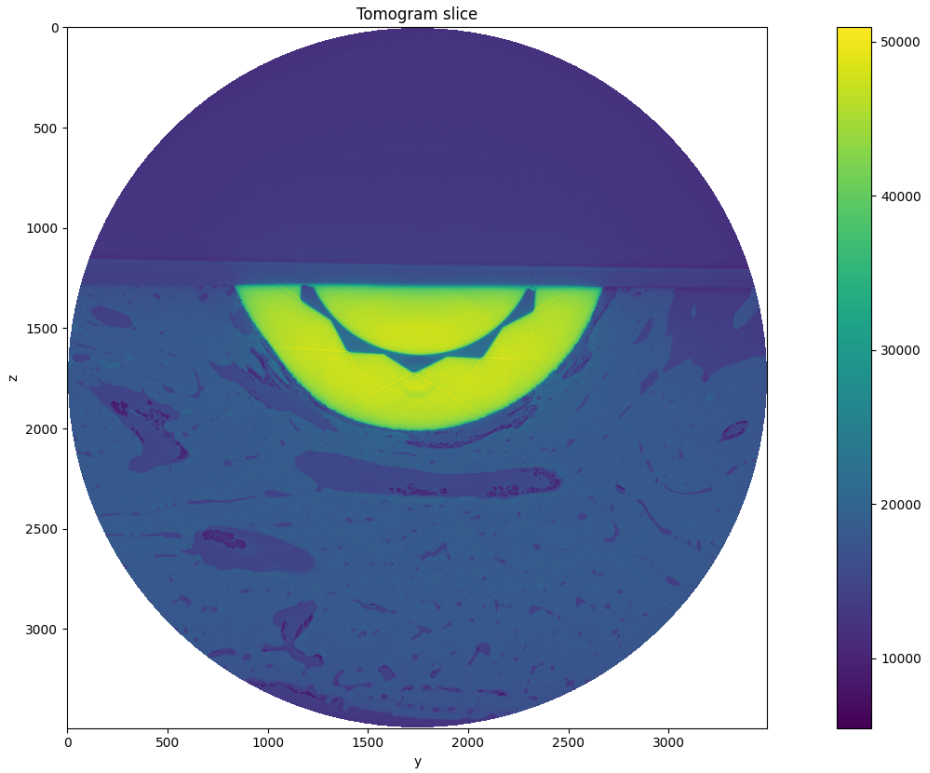


Figure 4.1: An example of a tomogram slice, labelled 770-slice-1500-1505 in the dataset. This slice is a good representation of a typical tomogram in the dataset, as it contains all the materials were are searching to identify. Note: the bright region in the centre of the slice, this is the titanium implant, the area above the bright centre, the air, and the area surrounding and below the slice; bone, blood vessels and resin.

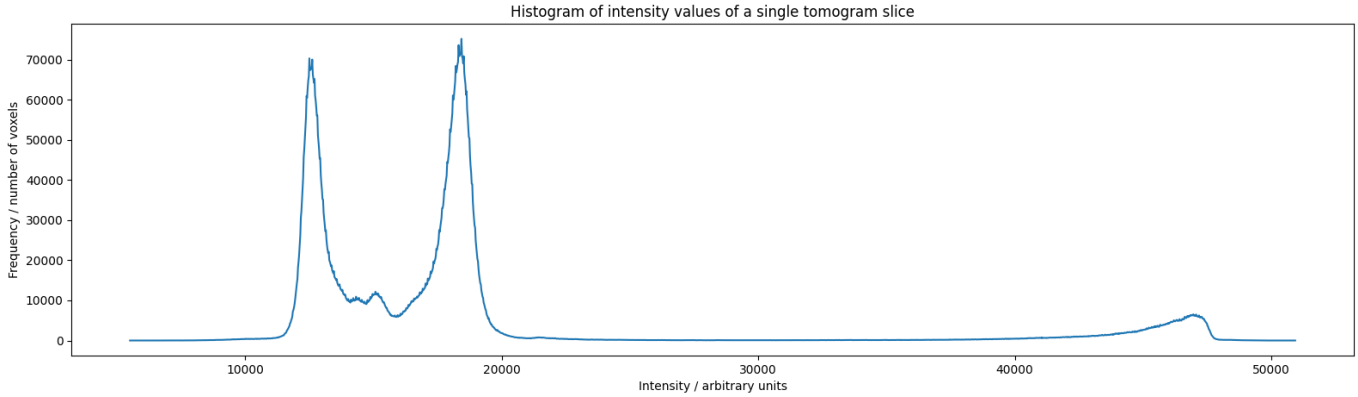


Figure 4.2: A histogram of the intensities present in the example slice. Note the peaks corresponding to the various materials present in the slice. There are two large peaks, which likely are due to the air and bone present in the image, and between them a series of smaller peaks, due to blood vessels and other impurities. Then the highest intensity peak is the titanium implant, which can be seen as a low, non-gaussian peak at the rightmost point of the histogram.

The intensity histogram has a series of peaks. Each peak should represent a material present in the sample. To each of these peaks a gaussian can be fitted so that the sum of these gaussians fits well to the entire histogram. This sum of gaussians was fitted to the histogram using a non-linear least squares method. Also, using multiple gaussians to fit the same peak was tried for an even more accurate curve fitting which may give a better fitting for some peaks as compared to only using a single gaussian for each peak.

In order to fit this sum of gaussians to the histogram, bounding was also needed so that each peak would accurately be fitted with a gaussian. The bounding was such that it forced a gaussian to be present in the required location for each peak in the histogram. A good initial guess for the parameters of each gaussian, along with the bounding, helped by reducing the time taken for the computation.

Once there was a fitted approximation for the curve the probability that a single voxel was a member of a specific gaussian in the fitted curve, and thus a specific material, could be ascertained. This was done based on the equation $\frac{h_1}{H+\epsilon}$. At the edges of each gaussian the height falls to very low values, but continues to be nonzero for a long time. In order to account for this, and not predict large probabilities of a voxel in these extremities belonging to the wrong material group, a value ϵ was also added to the sum H.

Using this for each gaussian the probability that a single voxel contained a specific material could be found and then applied to the tomogram slice to separate the different materials present in the tomogram. Then it was possible to present only the various elements of the tomogram that correspond to a specific material to qualitatively analyse the results of this process.

5

ENTIRE TOMOGRAM

Due to the effects of beam hardening and reflection due to high refraction at very high contrast interfaces, such as tissue - titanium, the position of a voxel must also be used to determine the material to which that specific voxel belongs. By looking at all the slices in each plane (x-y, x-z, y-z) as well as the displacement from the centre of the tomogram, r, we can ascertain the relationship between position, intensity and the material. This should allow the effects of beam hardening and reflection to be mitigated.

First I looked at all the slices in the x-y plane, the same plane as the single tomogram slice analysed earlier. Similarly to that process the first step was to make histograms for every slice. These can be plotted as an image where the x axis is intensity of the slice and the y axis is each slice through the other coordinate, in the case of the x-y plane the z coordinate. The brightness of the image represents the frequency of the histogram for a particular intensity.

Using the initial work of Carl Johnson, each histogram in this range the locations of each peak was found automatically. The progression of a peak corresponding to a certain material throughout the numerous histograms is referred to as a line, and each line of peaks was labelled. As the series of histograms can be viewed as an image it was possible to perform image analysis techniques on these histograms. Morphological transformations were performed with a small kernel size. First the lines present in the image were eroded, then they were dilated. Erosion removed the edges of the lines in the image, thinning them as all pixels near the boundary of a line were discarded. This removed small background noise in the line detection. Dilation is the opposite process to erosion, where erosion reduces the size of the line in the image dilation will increase the size of the lines and as noise should be reduced we are then only enlarging the true presence of a line. This was useful to join discontinuities in the lines. Following that the contours of the image were found, in order to find which components of the image, which lines, were connected. These were then labelled, and it was assumed that each line represented a single material. The kernels used during this process were elongated in the vertical direction, the direction parallel to the lines. This was done as the lines of peaks needed to be continuously connected and a line is much longer in the direction of its length than its width.

Using this information it was possible to begin automatically bounding and setting initial guesses for the fitting function to use. New curves were fitted for each slice. By having the peak locations already found the centre point of the peak can be estimated which enables automatic bounding.

The initial guess for the height value, a, is set as being the highest point of a specific peak. The bounds are then set for height between 0.2 and 1.01 of max height value previously found. The initial guess for displacement, b, is the intensity of the highest point of the frequency peak. On the histogram this corresponds to the value of x at the initial guess for height.

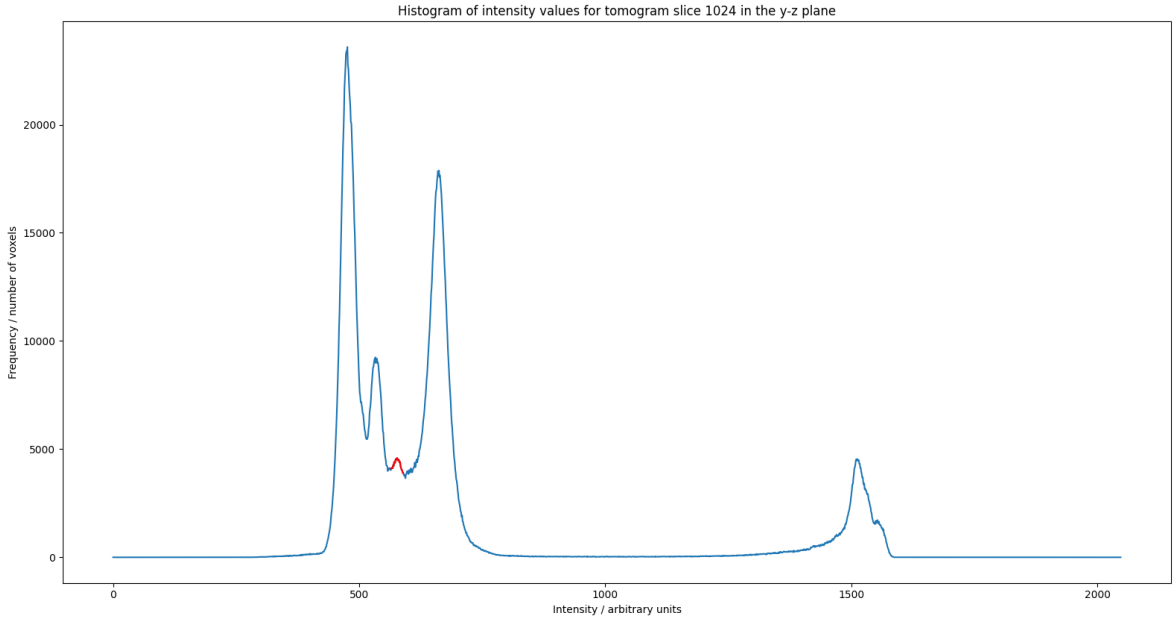


Figure 5.1: This is a histogram of the intensity values for the tomogram slice at 1024 located in the y-z plane. The part of the histogram coloured in red is a peak that has been identified. Notice the height, size and shape of that peak relative to the peaks around it. This large difference in height caused issues when trying to automatically bound the exponential which was intended to fit to that peak.

The width, c, of the fitted curve was set to be the width at the point where the height of the peak has fallen by half, for both sides of a given peak. This doesn't always work, as some peaks are nestled next to much larger peaks[Example peak], so if the ratio of height to width is greater than 0.001 instead a more dynamic fitting of the initial width of a curve was needed. Instead of setting width to be equal to the fall off at half peak height, I tried where the height falls off to 0.9 of the maximum. It is important to note that the displacement of a peak where this smaller width is used must not be above 1200, as this would be the metal implant, which is very short, wide and outside the range of the 'normal' peaks due to bone, blood, etc. This means I could exclude this high intensity peak when taking the comparative ratio mentioned before. If the ratio was greater than 0.01 something even more drastic was needed. Here I set width to be at the location of height fall off to 0.99 of the maximum. As this must be a peak nestled inside, or very close to other peaks as shown in Fig. 5.1 The bounds

for the width of the curve was set to 1.3 times larger and smaller than the initial guess for width, c.

In order to obtain a better fit for curve it was necessary to drop pure gaussians and instead have the exponent of the gaussian be alterable.To this end the possible range for this was set between 1.5 and 2, with an initial guess of 2, which would make that component of the overall curve a gaussian. This made the equation of the curve a sum of exponentials like this:

$$\sum_{i}^{N} A_i^2 \exp\left(\frac{\left(X - B_i^2\right)^{D_i^2}}{2C_i^2}\right) \tag{5.1}$$

For each histogram a curve made up of exponentials was fitted using a non-linear least squares method to fit my function to the data. Though this could cause a problem, with fitting leaving the curve higher than the data it is fitting. To counter this problem an energy function that must be minimised is used instead. Then a function with a minimise method can be used to fit a new curve to the data. This curve was tuned with a weighting penalty that penalised any result which was higher, or larger, than the data.

6 INTERPOLATION

Having found the values that govern the curves that have been fitted to histogram across a series of slices of the tomogram, from a certain orientation. These values need to be smoothed. To do this I used a series of low degree polynomials with a small number of segments, six. These polynomials took the form:

$$A + Bx + Cx^2 + Dx^3 \tag{6.1}$$

These were used to interpolate the values that govern the curves that have been fitted to histograms, as over the range of histograms the values were not continuous. In order to do this the interpolation must be continuous and differentiable (at all points and at the connections between segments). A matrix of equations for each point in a curve value was created, across the space of histograms, i.e. a across z. This means x is the displacement, z, of the tomogram slice and y, or the result of the above equation, is the curve value at that point.

There were two conditions used to fit an N-segment piecewise cubic polynomial to a set of points with linear least squares:

(1) Continuity at the borders: $f_n(X_n) = f_{n-1}(X_n)$

(2) Differentiability at the borders: $f'_n(X_n) = f'_{n-1}(X_n)$

Due to the constraints on the interpolation, after the first segment, I did not need to solve for A and B, as I could use the f(x) and f'(x) to solve for future values of A and B beyond A_1 and B_1. This did mean I needed to continue having the first set of A, B, C, D values in each equation. By doing this it also tied every equation in my matrix to every part of the interpolated line.

Taking this new, smooth line, I used it to set the displacement of each exponential to this continuous set of values from the interpolation. This, along with the other interpolated values were passed into the minimise function to be a new set of bounding conditions for a second fitting. This gave better, more continuous results.

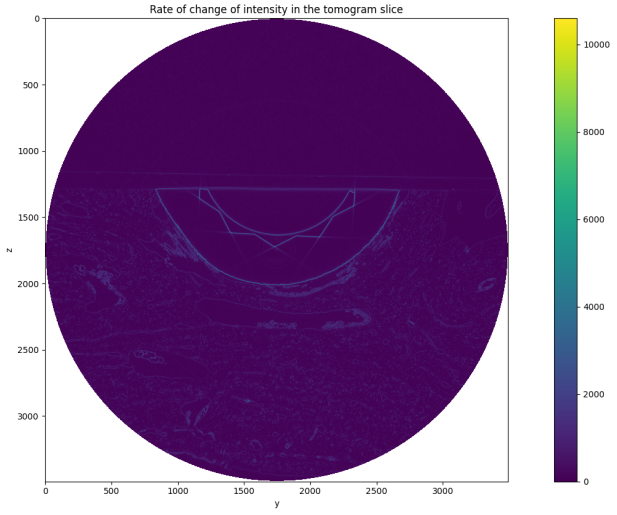
Once the spatial dependence of the intensity in each plane was ascertained, by using the method described above, it was then possible to automatically and systematically predict to which material a given voxel belonged. This enabled the removal of the disconnectedness effects.

Part III

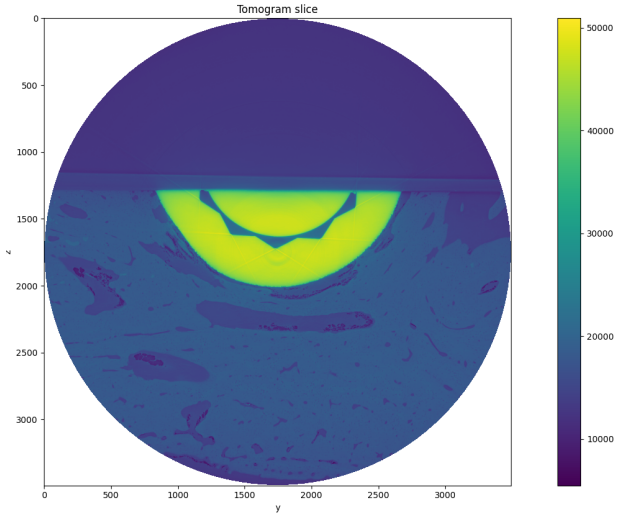
RESULTS AND DISCUSSION

7 SINGLE TOMOGRAM SLICE

A tomogram slice was chosen at a midpoint to be mostly representative of the overall structure of the bone; this was done by qualitatively analysing many slices. This slice contained all of the materials that I was looking to separate and distinguish. It is important to note that not all of the slices contained the metal implant, which has a very high intensity compared to the other materials present and is the cause of X-ray reflection.



(a) Image showing the rate of change of the intensity of the voxels in a single tomogram slice, labelled 770-slice-1500-1505 in the dataset. Note the bright lines around the location of the titanium implant, and at the edges of blood vessels. These areas are the primary locations for reflection, a phenomenon discussed earlier.



(b) The tomogram slice used for this part of the analysis, 770-slice-1500-1505. A good representative slice to use to begin a curve fitting attempt.

A brighter voxel on the tomogram indicates a higher intensity of the transmitted X-ray beam and as such should mean that the material at the specific voxel is more dense. Conversely the darker regions of the tomogram are indicative of lower intensities, and thus lower density materials. From a qualitative analysis of the tomogram slice it was clear there are four main regions that need to be distinguished. After taking a histogram of the intensities across this slice this then confirmed my initial qualitative analysis.

There are three main, large peaks present on the histogram. The leftmost peak, present at around 12,000 corresponds to the air at the top of the tomogram, which has many voxels with a similar low value. This can be confirmed by the darker colour in the tomogram slice. It is important to note that the units used for intensity here are arbitrary and at later points these same peaks will appear at different intensities. The next noticeable peak is at around 18,000. This peak

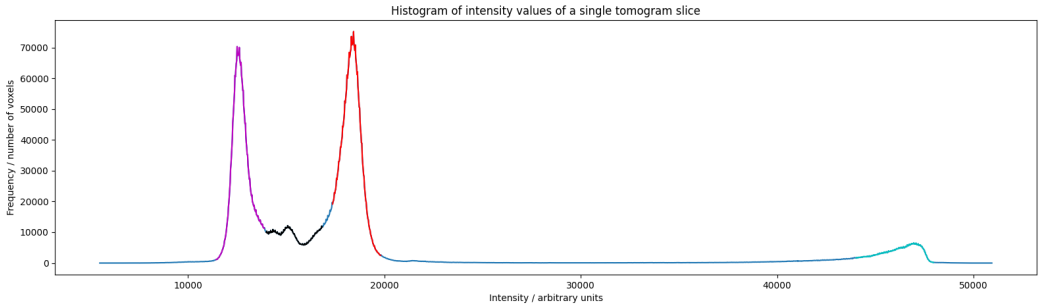


Figure 7.1: A histogram of the intensities present in tomogram 770-slice-1500-1505. Each region, which is caused by the presence of a specific material, has been coloured and labelled.

corresponds to the bone present in the sample, as a large number of voxels are bone in the tomogram. Between these two peaks with a high frequency of voxels sharing similar intensity values there are a number of smaller peaks which don't quite correspond to a specific material, though the main material present in this range should be the blood vessels in the bone sample. Otherwise this region contains a lot of noise, impurities and other non-uniform materials that exist across the tomogram. Finally at the far, rightmost end of the histogram there is a very low, wide peak; centred at around 47,000. This peak corresponds to the titanium implant present in the centre of the sample. It is the brightest region of the tomogram and the densest part of the sample being analysed.

As there are four main regions in the sample, and the histogram, to be separated. A curve composed of four gaussians was fitted to the histogram. Using my previous qualitative analysis of the peaks present in the histogram I could then give the fitting function reasonable bounding to use. I also supplied some initial guesses to the fitting function by reading off the heights, guessing a rough width and displacement for each gaussian peak present in the overall curve that was fitted.

This fitting gave close fits for the two largest peaks. This heavily supports the decision to use gaussians to represent the material peaks. It also supports the hypothesis that the materials in the bone sample have some mean value that the intensities are centred around, with a gaussian distribution. However, as the central region between those peaks was not comprised of a single gaussian, this peak did not have as great a fit as the others. This is likely due to the impurities in the sample being mainly in this intensity range.

In addition the titanium peak, which does not look particularly gaussian in nature, was also not as well fitting. Though, as there are no other peaks in the vicinity of this peak, an imperfect fitting of the curve in this region shouldn't cause an issue when using the fitted curve to distinguish the various regions of the tomogram.

Next I tried improving the fit of the curve to the data. To do this I tried using multiple gaussians for some of the peaks. Specifically the

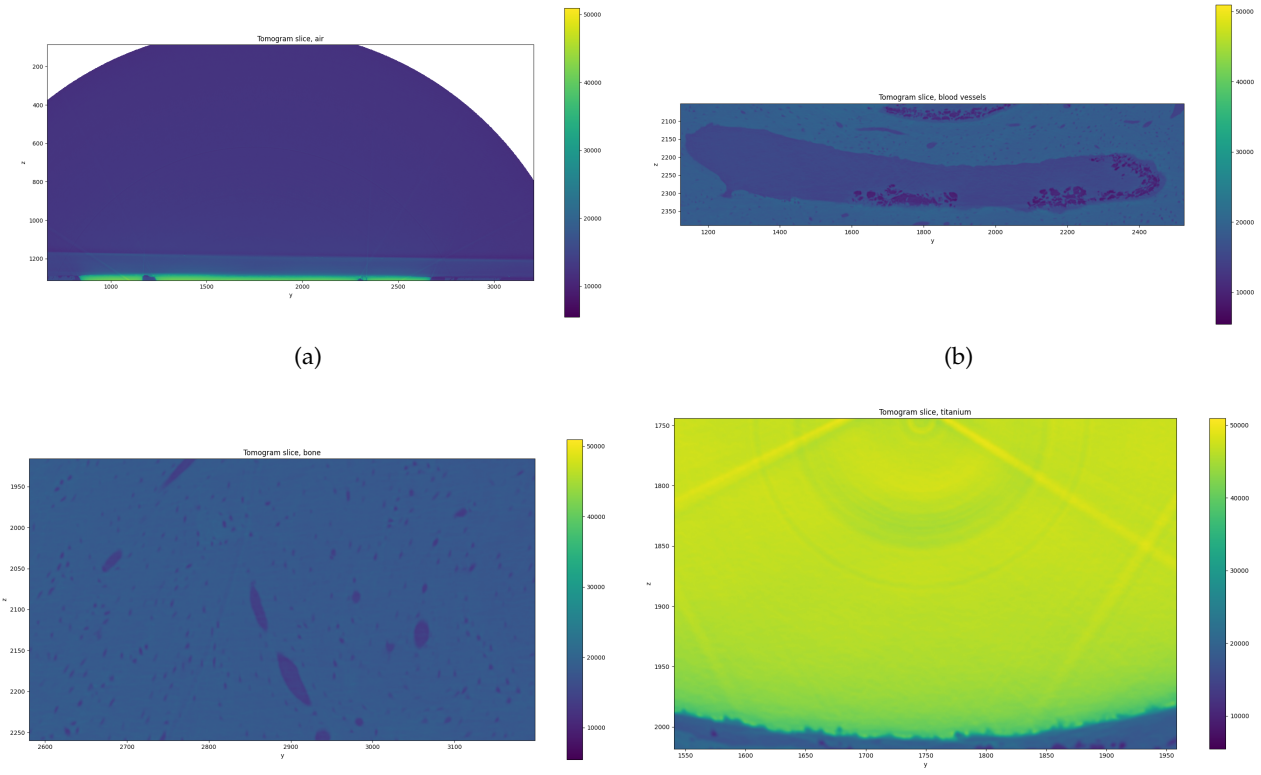
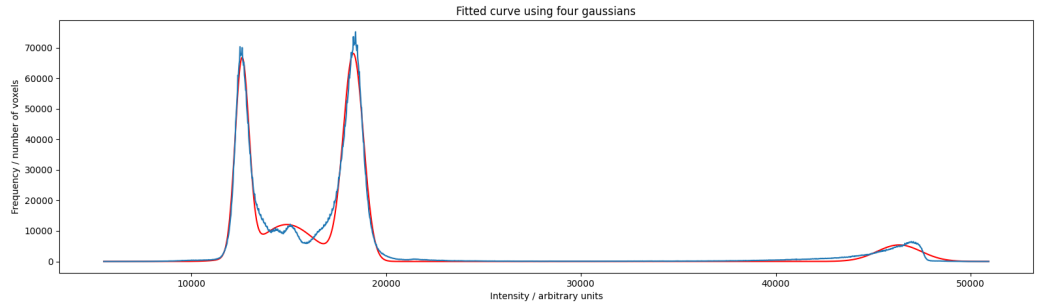


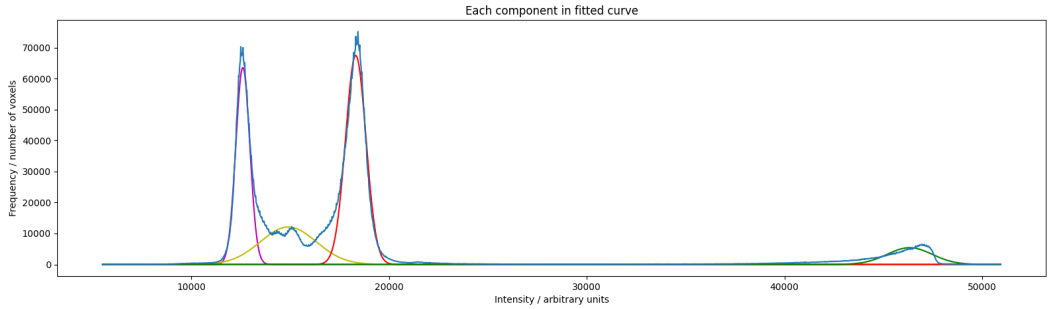
Figure 7.2: These are representative components of the tomogram slice, 770-slice-1500-1505. Each image highlights a material present in the sample. (a) is the air present, it tends to be a uniform area in the top part of the tomogram slice in the y - z plane. (b) the blood vessels present in the bone. Centred in the image, is a large blood vessel, but note the smaller blood vessels present throughout the main bone structure surrounding the blood vessel, identifiable only as dark flecks. (c) is an example of the bone present in the sample. Again, there are blood vessels present throughout the bone, visible as small dark spots of varying shapes and sizes. Finally (d) is an example of the titanium alloy present in the sample, it is a very bright, and therefore dense, region of the tomogram.

two largest peaks. This gave a better fit for the data than the previous, single gaussian per peak that was used first. The most noticeable improvement occurred in the central region between the two large peaks. This was as expected, as in that range of intensity there are a number of other impurities amongst the blood vessels.

However, these improvements do not cause any noticeable improvement in the probabilities calculated from this fitted curve. Whilst it does give a closer fit to the data represented in the histogram it does not seem to provide much additional benefit to the clarity of the distinction. As can be seen, from a qualitative standpoint, both sets of material classification probability representations of the tomogram are nearly indistinguishable. They both provide a reasonable classification of the different materials for which we are searching.



(a)

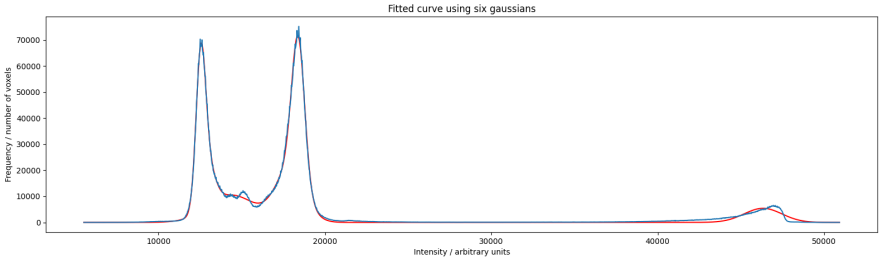


(b)

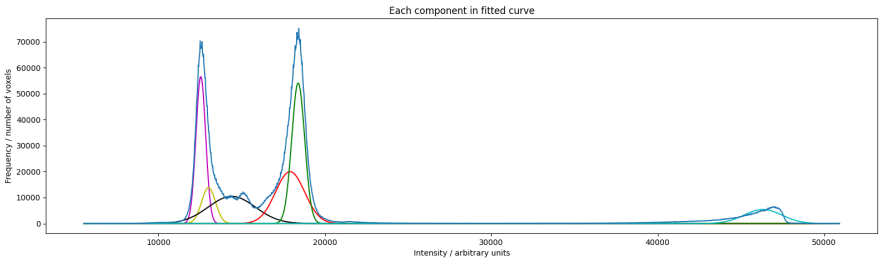
Figure 7.3: The first graph (a) shows the fitted curve next to the data. Note that this is using the least squares method and there are a number of regions where the fit raises above the data. The second graph (b) is the individual components of the fitted curve in (a).

With regard to the material classification probability representations of the tomogram, the intensity of the image represents the likelihood that the particular voxel in question is of a specific material. As can be seen close to the titanium-bone interface the intensity of those voxels are higher than other blood vessel voxels or bone voxels in other areas of the image. This is a great example of the reflection phenomenon described earlier. This also means that areas which, upon a qualitative examination, would appear to be blood vessels are being distinguished as bone. They do not exist in the blood vessel classification probability image, but clearly are a part of the bone classification probability image. Near the edges of the image there are also great examples of beam hardening, with a similar but opposite effect to the visual representation of reflection observable in the possibility representation images.

While this approach was a good start to distinguish the various material regions of the tomogram, there are clearly issues. As discussed previously, the reflection phenomenon present at the high density to low density material interfaces causes issues with the automatic classification of the tomogram. In addition the beam hardening causes issues around the edges. In order to address these issues a spatial component, as well as the intensity, of the voxels in the tomogram across a number of spatial dimensions needs to be investigated. This was then used to account for these phenomena.



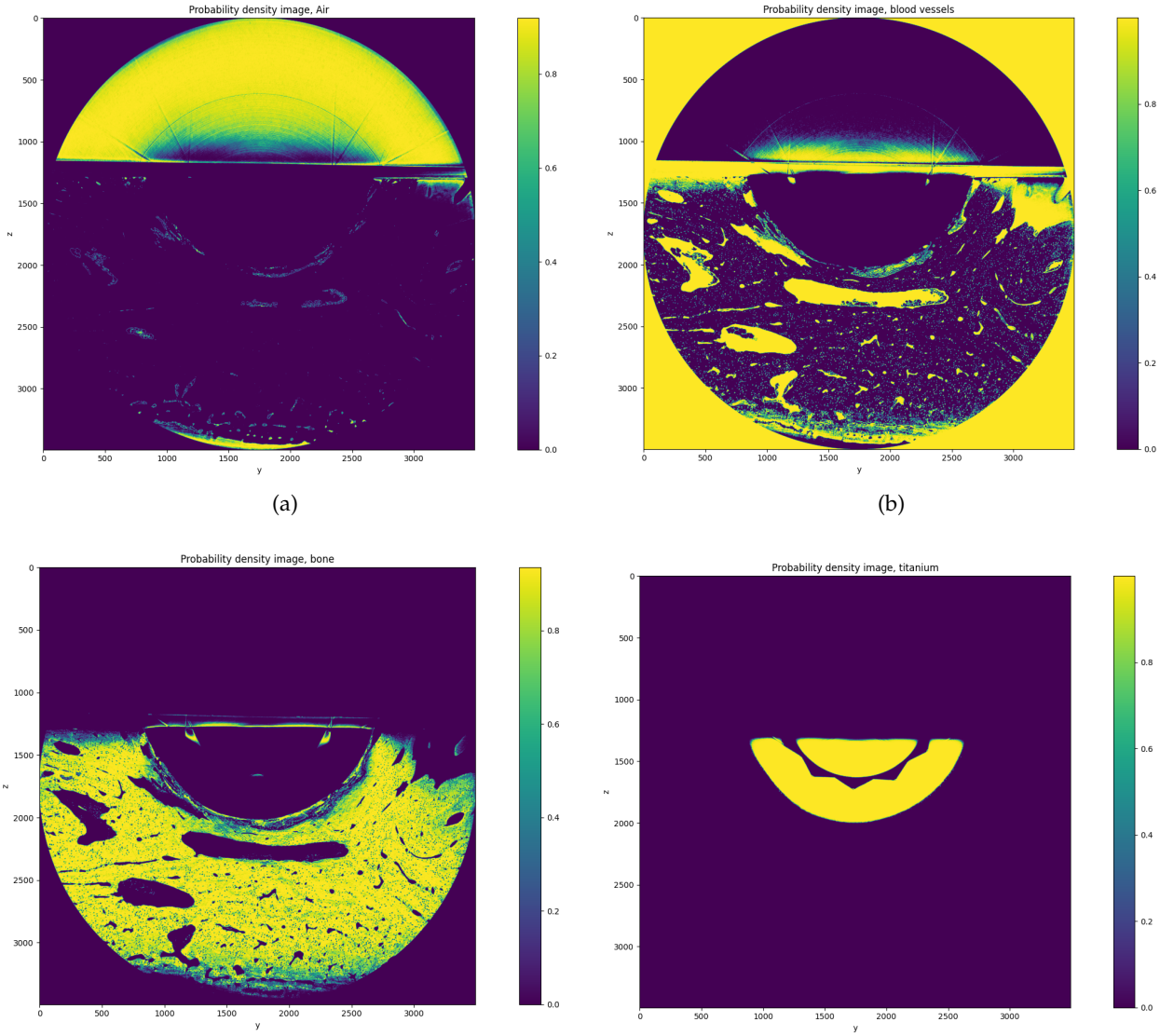
(a)



(b)

Figure 7.4: The first graph (a) shows the fitted curve next to the data. Note that this is using the least squares method and there are a number of regions where the fit raises above the data. The second graph (b) is the individual components of the fitted curve in (a). There are six components of the curve that are fitting to four peaks in this regime. Note the improved fit around the central region, blood vessels and impurities, this is due to the addition of more components in the fitted curve.

This implementation did not have any way to work out the initial guesses and bounds of the curves in an automatic way. Without some automatic generation of these initial conditions for the fitting function the process used to segment the materials of this tomogram slice cannot be extended to other slices. This was not very useful, and needed to be addressed. Additionally the implementation did not have an automatic way of finding the general locations of the peaks present, which would of course be the prerequisite to an automatic solution for bounding and generating initial guesses. These issues were addressed as the process was refined.



(a) (b)

(c) (d)

Figure 7.5: The material classification probability representations of the tomogram for one slice. These represent (a) air, (b) blood vessels, (c) bone, (d) titanium alloy. They were calculated using only four exponentials in the curve, one for each peak present in the data. The colour of the image represents the probability that a specific voxel is of the material titled. The important features to note are the blood vessels near to the titanium implant which have been misclassified as bone with this analysis, and the resin which has been classified as blood vessel, due to its similar position in the intensity.

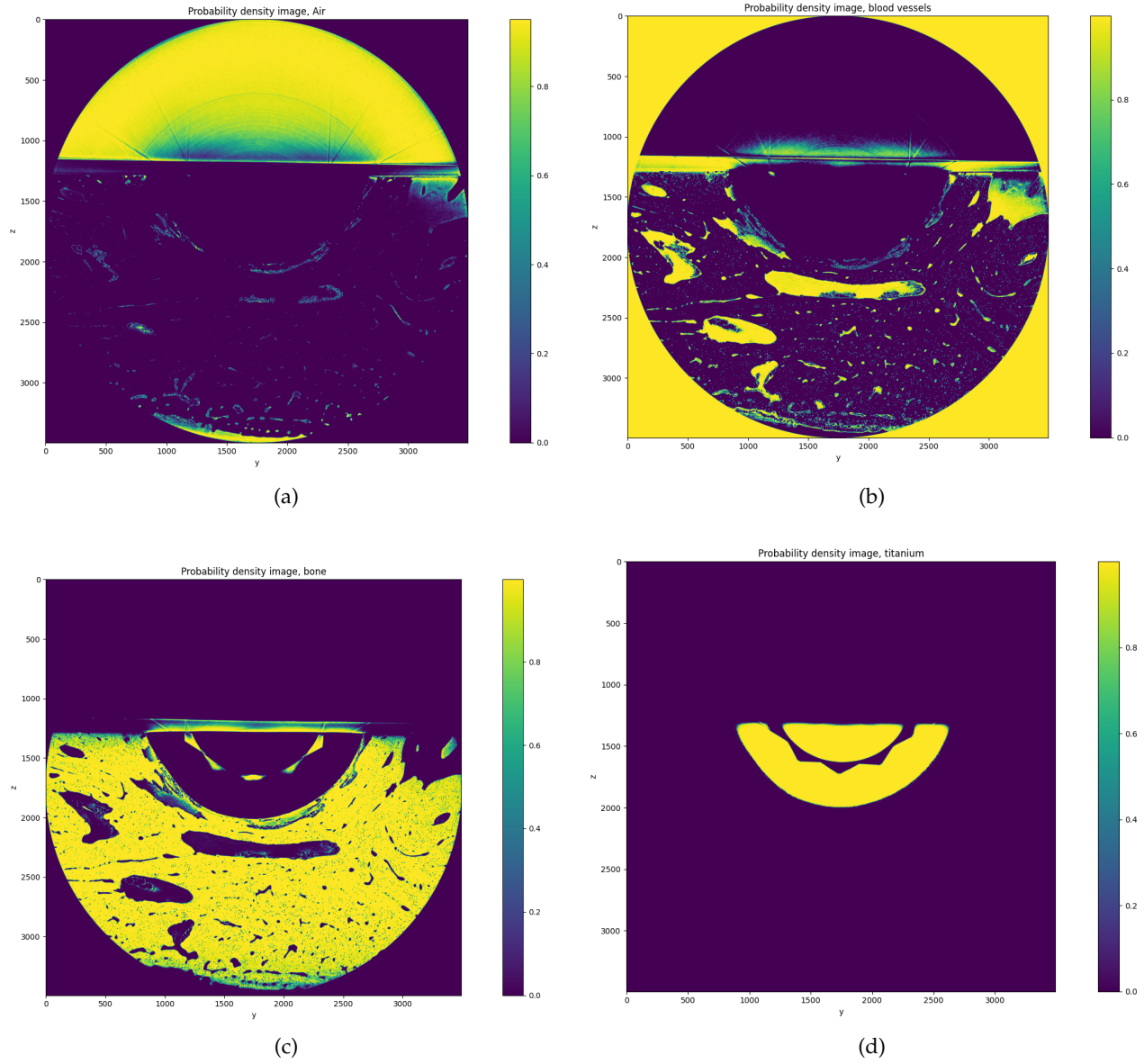
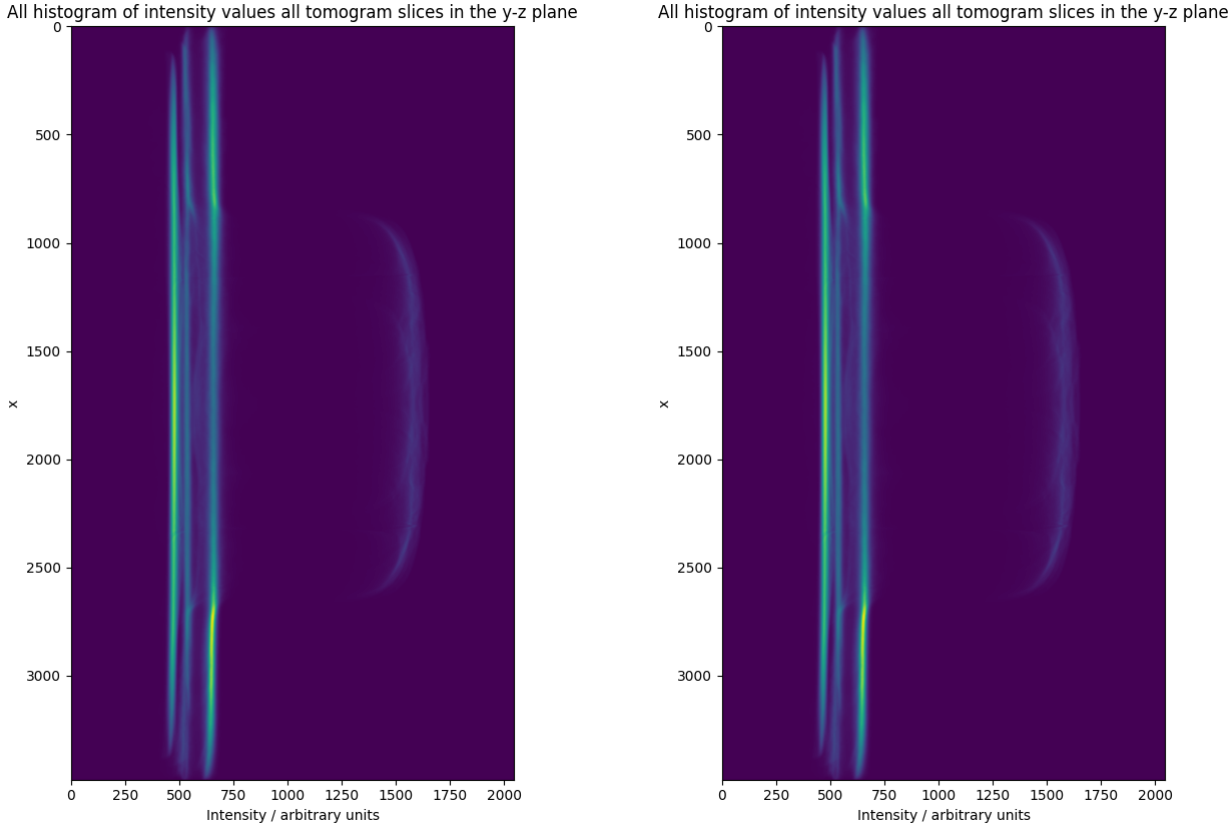


Figure 7.6: The material classification probability representations of the tomogram for one slice. These represent (a) air, (b) blood vessels, (c) bone, (d) titanium alloy. They were calculated using only six exponentials in the curve, two for the two largest peaks and one for each other peak present in the data. They are qualitatively very similar to the probability distributions produced with four exponentials making up the fitted curve.

8

ENTIRE TOMOGRAM

First, I started by examining the tomogram slices on the y-z plane, such that each slice was layered down along the x axis. The tomogram slice's intensities were binned to generate histograms for each layer. This series of histograms was plotted in 2 dimensions to perform a qualitative analysis of the histograms. The continuity of the peaks, visible on a single histogram of a single slice, along the x direction will be referred to as lines. If the X-ray tomography.



(a) (b)

Figure 8.1: These images represent the peaks in the histograms through x, where each histogram is a slice in the y-z plane. (a) is the image of these histograms, with the height of the peak, or the intensity, represented by brighter colours. Note the titanium implant midway through the slices and at a high intensity. It is also easy to see the effect of spatial position from this image, as the peaks curve at their start and end points, usually near interfaces. This will be due to the beam hardening phenomenon. (b) shows the lines detected automatically across the space, with each line having a different colour to signify it's individuality.

Then using the method outlined previously these curves, made of sums of exponentials, were fitted to every histogram. This was done with both a least squares fitting and a minimising fitting. Then the computed curves were used to calculate the probability that a voxel at a certain intensity and displacement, x, was a member of a specific material. When looking at some of the individual histograms, from an individual slice, there is little difference between the curve fitted using the least squares method and the curve fitted using the minimise method. The main noticeable difference is that the least squares method does not penalise overshooting, and as such that curve tends to rise above the data on many occasions. In contrast, due to the penalty incorporated in the energy function, the minimise curve never exceeds the data.

When we move on to viewing the probability distributions across the entire range of histograms for each material there is a qualitative difference between the two methods. It is quite clear that the least squares method, which overshoots, does not produce probability distributions that are as contiguous as the minimise method. Furthermore the exponentials fitted with least squares have a higher tendency to be of a lower height but greater width than those fitted using the other method. This lower connectedness of probability distributions and sporadic flattening of individual exponentials in the overall curve leads to a higher incidence of splits along individual lines. These splits cause discontinuity problems for both sets of probability distributions, but are clearly more pronounced amongst the curves calculated with a least squares method.

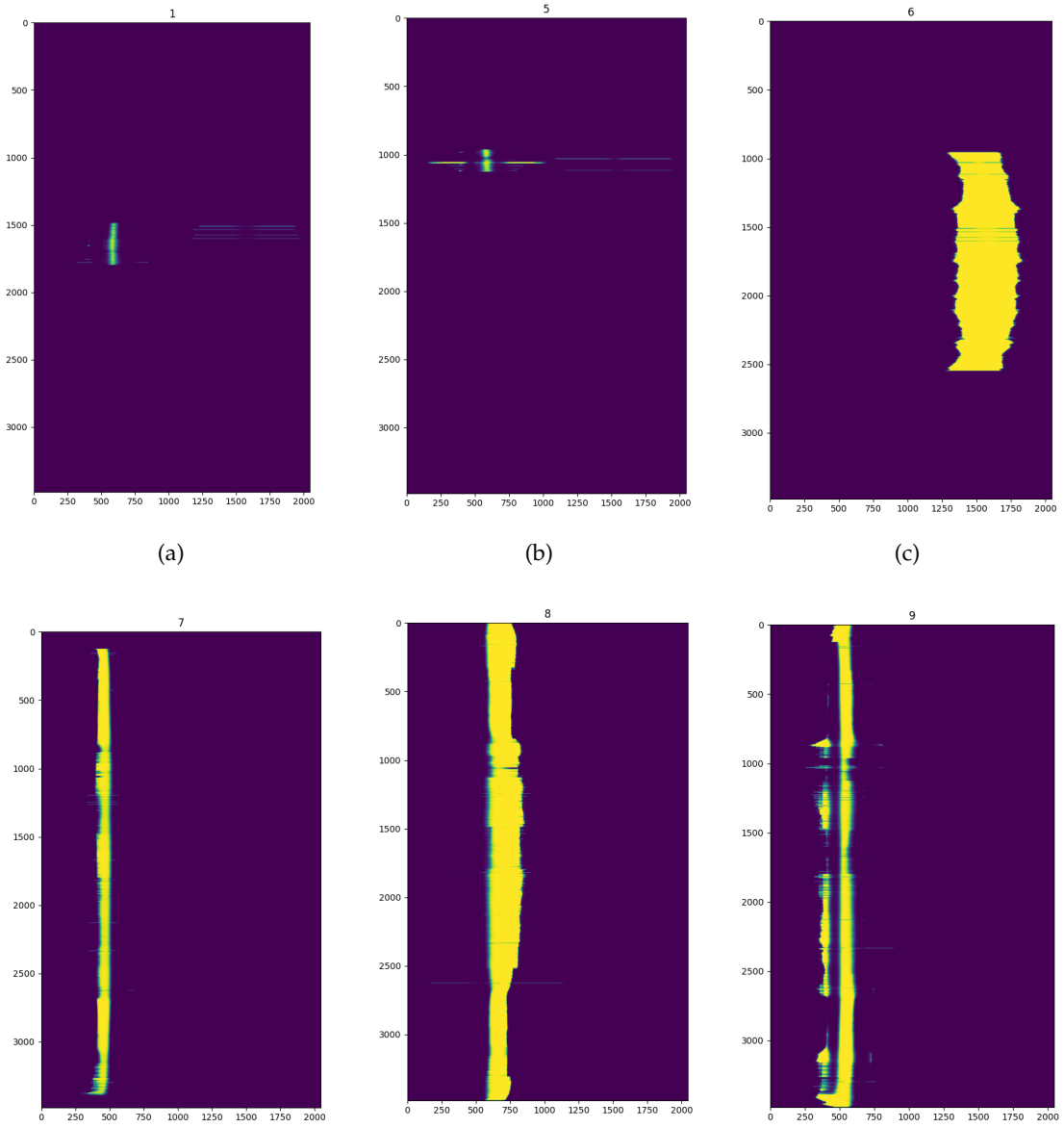
Materials labelled '1' and '5' are interesting to investigate. These two peaks are quite likely both part of the same smaller line which exists nestled amongst the larger, and more obvious, lines surrounding them on either side. On close inspection, they are noticeably shallow and difficult to define. It is highly likely that the material these peaks represent is an impurity in the sample, or possibly the resin component. They are noticeable in the lines representing materials '8' and '9'.

By comparing the histograms calculated for a single tomogram slice with the lines present in his entire histogramme space, it is possible to quantify the materials each line represents. The easiest to distinguish is the titanium implant, as that has a very high intensity. That is material '6'. Next is the air part of the tomogram, that is of the lowest intensity as it is the least dense and therefore will attenuate the X-ray beam the least. This is the material labelled as '7'. Following this, by comparing a single histogram from a single slice with the single slice analysed earlier it is possible to discern that material '8' is bone and material '9' is blood vessels.

Upon inspection, it was clear that some smoothing is needed of the variables that compose each exponential across the histogram space. This is where interpolation comesIt is important to note that to keep the time of the computation down, but still maintain accuracy, the maximum number of function evaluations for the minimise method was set to forty. This number was chosen during testing as it seemed to

be an effective number of evaluations. This is due to the fact that if the energy function is minimised by this point, forty evaluations, it doesn't really improve much more. At the extreme end of the evaluations this is because the minimisation is stuck oscillating between a set of values until it reaches the function evaluation limit.

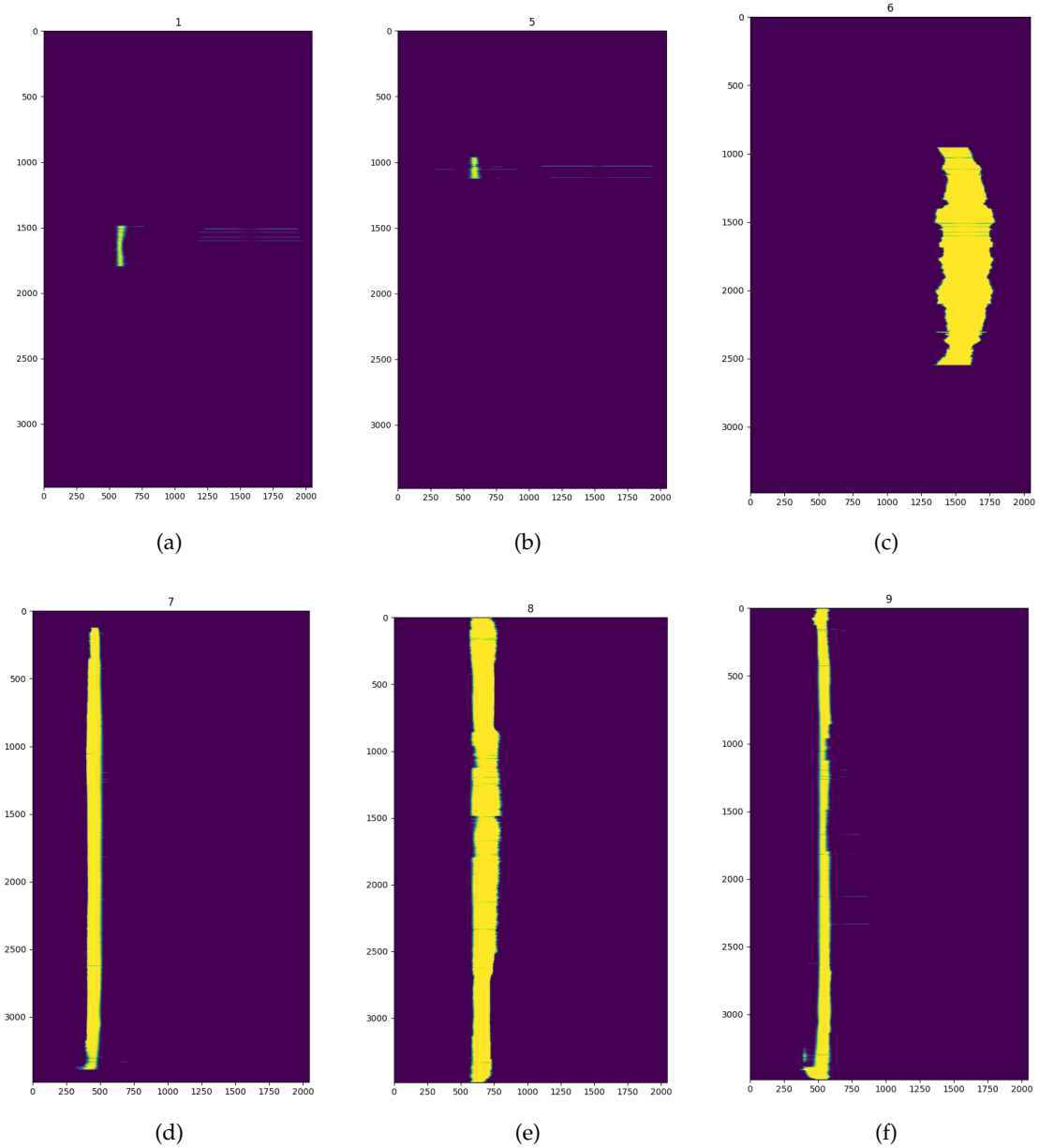
Upon inspection, it was clear that some smoothing is needed of the variables that compose each exponential across the histogram space, as the probability density is not the smooth function we would like it to be. This is where interpolation of the variables can be used. It is also clear that using the minimise function produces better results, and for that reason the least squares fitting method will be dropped in future iterations of this process.



(a) (b) (c)

(d) (e) (f)

Figure 8.2: These are the probability distributions for each material across the entire range of tomograms. These curves were fitted using a least squares method. The colour represents the probability that a voxel in that spatial location and at that intensity is a part of the material. The lines that correspond to definite materials are (6) titanium, (7) air, (8) bone and (9) blood vessels. The minor lines (1) and (5) represent some impurities in the sample. Material (9) is a great example of poor continuity which is not penalised by the least squares method.



(a) (b) (c)

(d) (e) (f)

Figure 8.3: These are the probability distributions for each material across the entire range of tomograms. These curves were fitted by using a minimise method on an energy function. The colour represents the probability that a voxel in that spatial location and at that intensity is a part of the material. The lines that correspond to definite materials are (6) titanium, (7) air, (8) bone and (9) blood vessels. The minor lines (1) and (5) represent some impurities in the sample. In comparison to the least squares method material (9) has far better continuity as this method penalises discontinuity.

9 INTERPOLATION

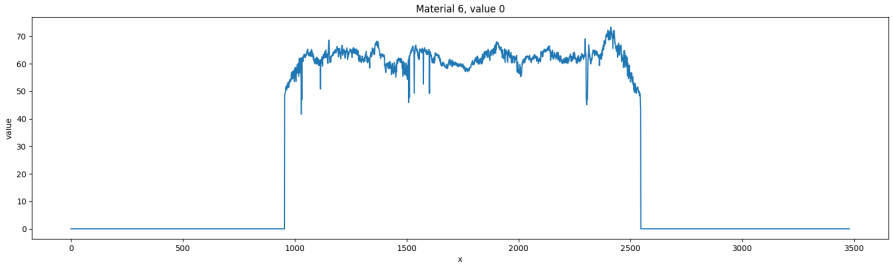
In order to interpolate the individual values of each exponential component in the fitted curves it was necessary to plot them across the numerous histograms. This showed that some values were much more continuous than others, and likely not the reason for the discontinuities in the probability distributions. However, other values did follow a smooth function, these values included a number of anomalous points, far outside the observed range of the other values. It is important to note that all of these values are squared in the exponential function, as they cannot be negative in these exponentials. This also means that they differ from the bounds specified earlier, as the bounds are the bounding of the square of these values. For instance the value of D in a number of places seems to be constant at a value just above 1.4. Yet this is actually due to the fact that it has reached the edge of the bounding, it is at SQRT(2).

There are a large number of slices where some materials are not present. This can be seen for instance in the titanium line. Outside the presence of a peak the values for the exponential of course drop to zero. In order to interpolate the values correctly any values outside the existence of a peak were dropped, where they were zero. To interpolate 6 segments were used. In addition the mean and standard deviation of each value, ignoring values of zero, was calculated. And entries for a value that were more than three standard deviations from the mean were excluded, this addressed the anomalous entries for certain values across the space.

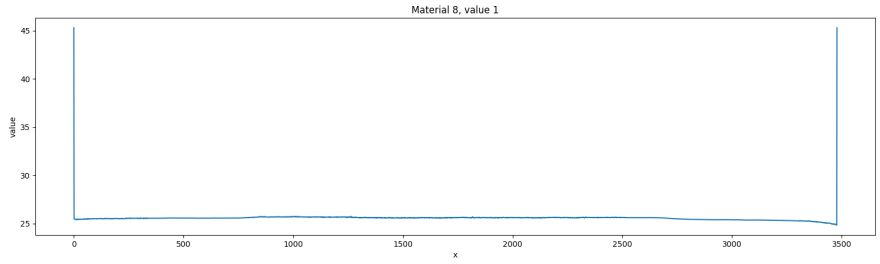
Using these new interpolations for the values of the exponentials three regimes were tried. First I tried setting all of the values to those gained through interpolation, without attempting to use the minimise function a second time. Whilst this did provide very smooth probability densities throughout the space, there were definite issues which were quite striking upon qualitative investigation.

The next method tried was to use the minimise method a second time but to set the initial guess as the interpolated value at each point, x. Then set the boundary conditions to be one standard deviation from the interpolated line. This gave a better result than the previous method tried, without using the minising function. Though there are clearly still issues with discontinuity and lines split by odd slices.

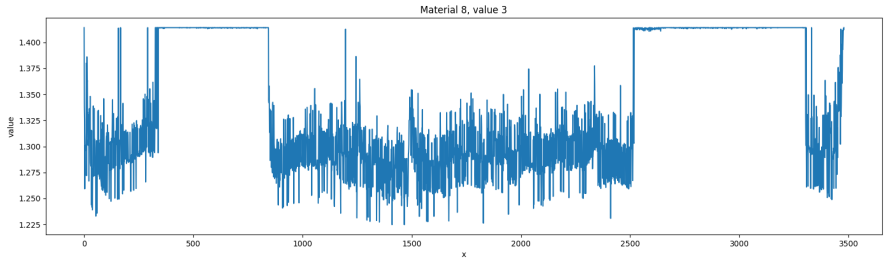
The final method used was to set the value of B, the displacement of each exponential component, using the interpolated values. This gave a smooth and continuous central point of each peak. This was the best result yet for the probability distributions across the spatial coordinate. Once this was completed for the y-z plane it was then used on the x-y, x-z and r planes to produce probability distributions across the space.



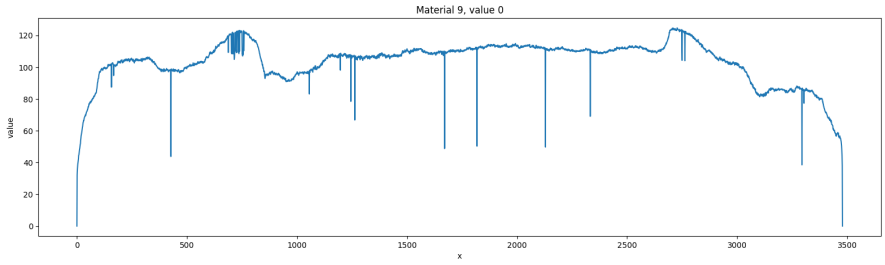
(a)



(b)

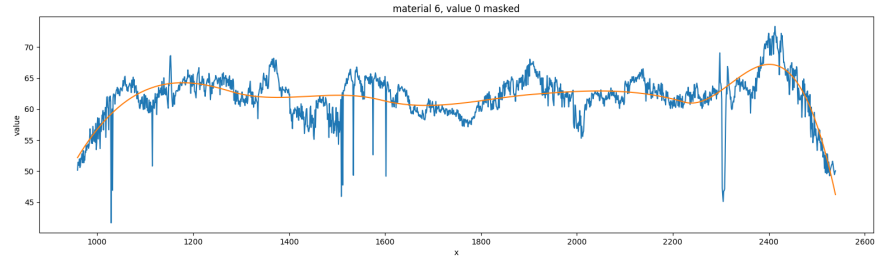


(c)

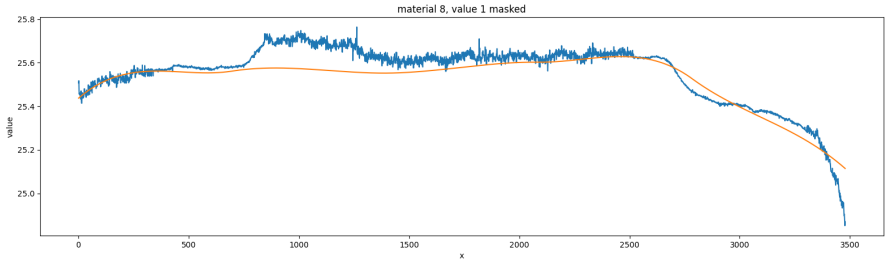


(d)

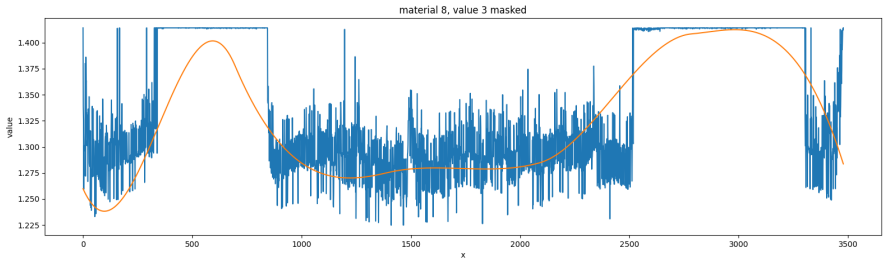
Figure 9.1: These graphs are a representative example of the disposition of the values that compose the exponentials in each fitted curve. These graphs represent: (a) material 6 value A, (b) material 8 value B, (c) material 8 value D, (d) material 9 value A. Note the varied nature between each graph. Some values have a very discontinuous form (c), whilst others, apart from some odd anomalous numbers, are quite continuous (b). It is interesting as well to note that (c) on two occasions is equal to the upper bound of it's limits.



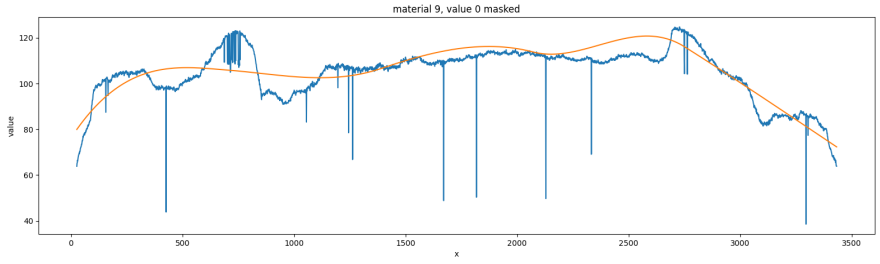
(a)



(b)



(c)



(d)

Figure 9.2: These are the interpolated results for the graphs shown in the previous figure. These graphs represent: (a) material 6 value A, (b) material 8 value B, (c) material 8 value D, (d) material 9 value A.

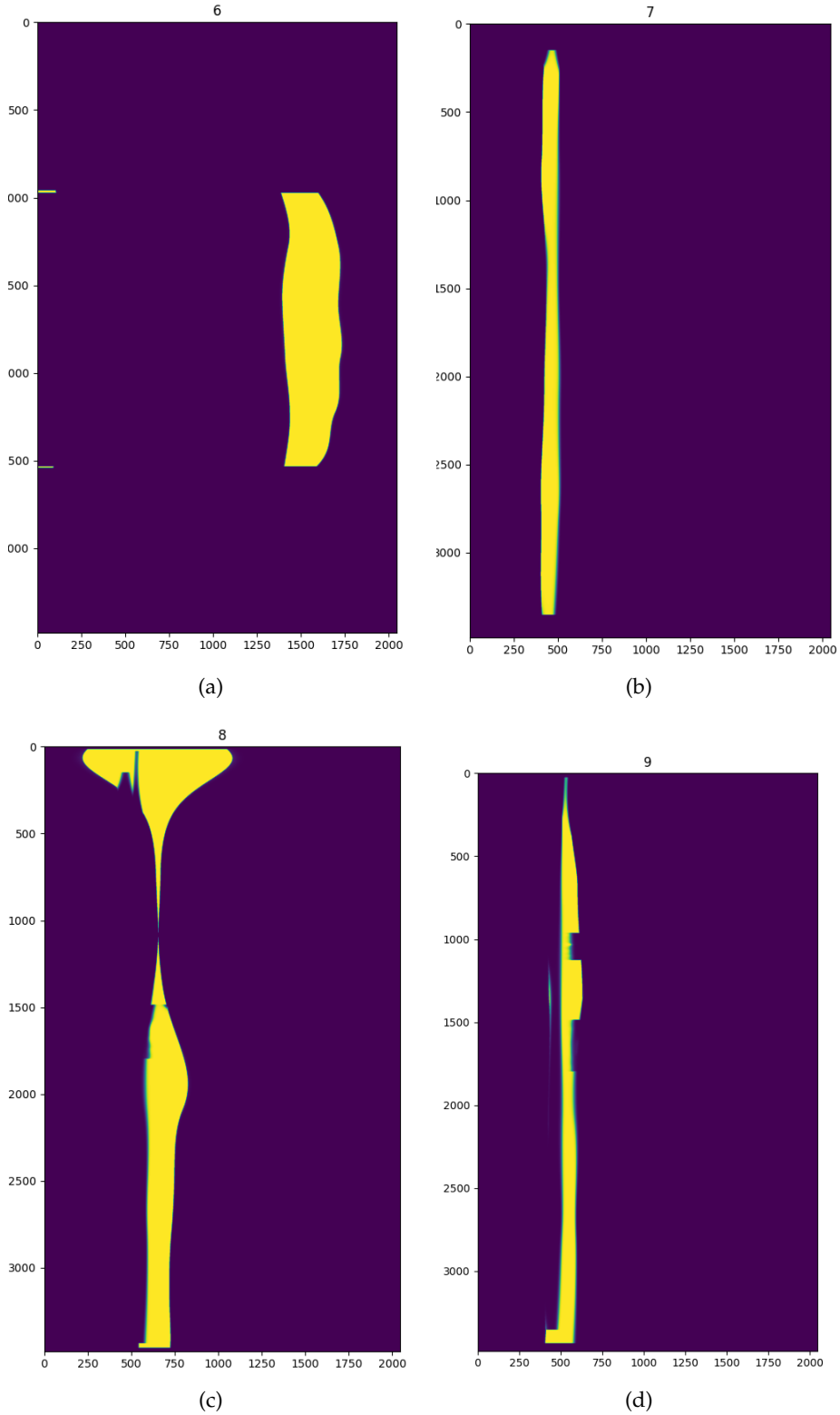
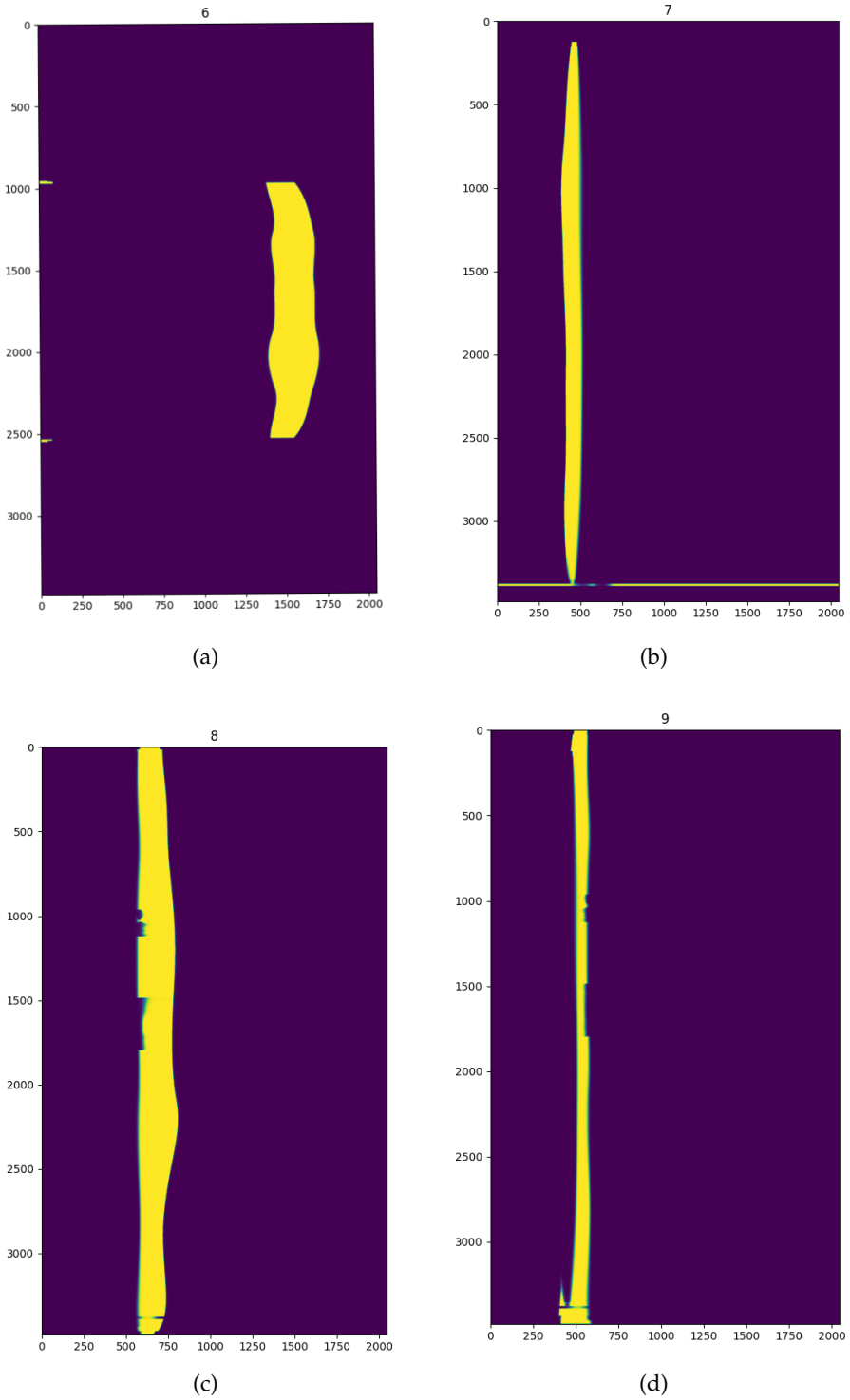


Figure 9.3: These graphs represent probability densities when using purely the smooth and continuous interpolated values to determine the probability. I have only included the plots for the four main components: (a) titanium, (b) air, (c) bone and (d) blood vessels.



(a) (b)

(c) (d)

Figure 9.4: These graphs represent probability densities when using the minimise method a second time, but setting the value B for every exponential component equal to that of the interpolated values for B. Otherwise the other values were given the same boundary conditions as the initial pass of the mininise method. I have only included the plots for the four main components: (a) titanium, (b) air, (c) bone and (d) blood vessels. This has given the best probability distributions across the spatial coordinate to date.

Part IV

CONCLUSION

10

CONCLUSION AND FURTHER WORK

Whilst on the surface some of the methods employed seem rather straightforward. The process of bounding the methods was quite challenging. In order to generate accurate probability spaces the bounding required quite fine tuning. Then setting these boundary conditions across the spatial coordinates, for instance x, was an even greater challenge.

Whilst the initial material classification probability representations of the tomogram for one slice seem to be quite accurate, outside the areas where beam hardening and reflection have affected them, the areas that they are inaccurate are some of the most interesting to study. Therefore without this automatic reversal of physical distortion effects by combining spatially dependent probability distributions over several axes, the tomograms are less useful for further work and research.

Due to time constraints the final combination of all planes to be reinterpreted with probability distributions on a single tomogram slice, using both intensity and spatial coordinates, was not completed. This would be a key piece of further work to round off this investigation. All of the components have been completed, as the probability densities were completed in every plan, and across the 'r' coordinate or distance from the centre of the tomogram. Though this recombination is non-trivial.

Following from that it should then be possible to use this process to systematically analyse and interpret many tomograms in order to automatically analyse the segmented tomograms to quantify bone-contact and blood flow to the implants around which the new bone grows. This would improve and hasten other work in this field, the MAXIBONE project.

BIBLIOGRAPHY

Bishop, Christopher M. (2006). *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag. ISBN: 0387310738.

Jähne, Bernd (2005). *Digital Image Processing 6th Edition*. Berlin [u.a.]: Springer. ISBN: 3540240357 9783540240358. URL: http://www.amazon.com/Digital-Image-Processing-Bernd-J%C3%A4hne/dp/3540240357/ref=sr_1_1?s=books&ie=UTF8&qid=1328626244&sr=1-1.

Peter, Zsolt-Andrei and Françoise PEYRIN (Apr. 2011). "Synchrotron Radiation Micro-CT Imaging of Bone Tissue." In: *Theory and Applications of CT Imaging and Analysis*. 233-254. InTech. URL: https://hal.archives-ouvertes.fr/hal-02268606.

Schaffler, M. B. and O. D. Kennedy (2012). "Osteocyte signaling in bone." In: *Current osteoporosis reports* 10.2, pp. 118–125. URL: https://doi.org/10.1007/s11914-012-0105-4.

Part V

APPENDIX

APPENDIX: SCRIPTS

Listing A.1: Script "single_slice_4_gaussians.py"

```
import numpy as np
import numpy.ma as ma;
import scipy.ndimage as ndi
import skimage.morphology as morf
import skimage.filters as filt
import skimage.feature as feature
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
from scipy.optimize import curve_fit
from scipy.optimize import minimize
#from scipy import polyfit

tomo_data = np.load("770-slice-1500_1505.npy")

(Nz, Ny, Nx) = tomo_data.shape
R = Nx / 2
xs = np.linspace(-R, R, Nx)
ys = np.linspace(-R, R, Ny)

rs = np.sqrt(xs[None, :] ** 2 + ys[:, None] ** 2)  # rs[i,j] =
    sqrt(xs[j]^2 + ys[i]^2)
mask = rs >= R  # mask[i,j] == rs[i,j] >= R, dtype==bool

tomo_slice = ma.masked_array(tomo_data[3], mask=mask)

plt.figure(figsize=(15,15))
plt.imshow(tomo_slice)
plt.show()

plt.figure(figsize=(15,15))
plt.hist(tomo_slice.compressed(), bins=2000)
plt.show()

#(vmin, vmax) = (tomo_slice.min(), tomo_slice.max())
(vmin, vmax) = (-4,12)
#print(vmin, vmax)

nbins = 2 ** 16 +1
bin_edges = np.linspace(vmin, vmax, nbins)
values = (bin_edges[1:] + bin_edges[:-1]) / 2

#counts, _ = np.histogram(tomo_slice.compressed(), bins=bin_edges
    )

counts, _ = np.histogram(values[tomo_slice.compressed()], bins=
    bin_edges)
```

```
#convolve with 1D gaussian

counts = gaussian_filter1d(counts, 10, mode='constant')

#print(len(counts))

plt.subplots(figsize=(20, 5))

plt.plot(values, counts)
plt.show()

x = np.array([2,5,7,0,1,3,6,3,3,7,0,9,1,3])
y = np.linspace(0,1,10)

z = y[x]
print(z)

test = values[tomo_slice]
print(len(test))
print(len(tomo_slice))

def gaussian(x, a, b, c):
return a * np.exp(-((x-b)**2) / (2*c)**2)

def testfunc3(xs, a1, a2, a3, b1, b2, b3, c1, c2, c3):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3)

def testfunc4(xs, a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4)
    :
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4)

def testfunc5(xs, a1, a2, a3, a4, a5, b1, b2, b3, b4, b5, c1, c2,
     c3, c4, c5):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4) +
    gaussian(xs, a5, b5, c5)

def testfunc6(xs, a1, a2, a3, a4, a5, a6, b1, b2, b3, b4, b5, b6,
     c1, c2, c3, c4, c5, c6):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4) +
    gaussian(xs, a5, b5, c5) + gaussian(xs, a6, b6, c6)

print('almost')
popt2, pcov2 = curve_fit(testfunc4, values, counts, bounds
    =([1000, 10, 500, 100, -1, -0.8, 0, 6, 0, 0, 0, 0], [10000,
    1000, 10000, 10000, -0.6, 0, 2, 8, 1, 1, 1, 10]))
print('alomst...')
probability1 = gaussian(values,popt2[0],popt2[4],popt2[8]) / (
    testfunc4(values, *popt2) +0.01)
probability2 = gaussian(values,popt2[1],popt2[5],popt2[9]) / (
    testfunc4(values, *popt2) +0.01)
probability3 = gaussian(values,popt2[2],popt2[6],popt2[10]) / (
    testfunc4(values, *popt2) +0.01)
```

```
probability4 = gaussian(values,popt2[3],popt2[7],popt2[11]) / (
    testfunc4(values, *popt2) +0.01)

min2 = np.max(np.where(probability1 > 0.5)[0])
max2 = np.min(np.where(probability3 > 0.5)[0])
where2 = np.logical_and(tomo_slice >= values[min2], tomo_slice <=
     values[max2])

probmap2 = probability2[tomo_slice]
plt.figure(figsize=(15,15))
plt.imshow(probmap2)
plt.colorbar()
plt.show()

print('here')
print(min2)
print(max2)
probability2[0:min2] = 0 * probability2[0:min2]
probability2[max2:len(probability2)] = 0 * probability2[max2:len(
    probability2)]

plt.subplots(figsize=(20, 5))
plt.plot(probability1)
plt.plot(probability2)
plt.plot(probability3)
plt.plot(probability4)
plt.show()

test1 = values[tomo_slice]
print('test1 length =', len(test1))
print('test1 shape =', test1.shape)
test1comp = values[tomo_slice.compressed()]
index = np.unique(test1comp)
index = np.sort(index)
print('index length =', len(index))
print('test1comp length =', len(test1comp))


indicies = np.arange(0, len(values))

np.min(index)
count1 = 0
initposit = np.where(test1 == index[5])
x = test1.shape
print(x)
new_tomo = np.zeros(x)
print(initposit)
print(np.where(values == index[5]))
print(values[np.where(values == index[5])])
print(index[6])
#test2 = values[index]
#print(np.max(test2))
#print(len(test2))

plt.subplots(figsize=(20, 5))
```

```
plt.title('Histogram of intensity values of a single tomogram
    slice')
plt.xlabel('Intensity / arbitrary units')
plt.ylabel('Frequency / number of voxels')
plt.plot(values, gaussian(values,popt2[0],popt2[4],popt2[8]),
    color='m')
plt.plot(values, gaussian(values,popt2[1],popt2[5],popt2[9]),
    color='y')
plt.plot(values, gaussian(values,popt2[2],popt2[6],popt2[10]),
    color='r')
plt.plot(values, gaussian(values,popt2[3],popt2[7],popt2[11]),
    color='g')
plt.plot(values, counts)
plt.show()



#new_tomo1 = values[tomo_slice]
print('max of tomo_slice', np.max(tomo_slice))
print('min of tomo_slice', np.min(tomo_slice))
print(nbins)

zone1 = probability1[tomo_slice]
zone2 = probability2[tomo_slice]
zone3 = probability3[tomo_slice]
zone4 = probability4[tomo_slice]

plt.figure(figsize=(20,20))
plt.imshow(zone1)
plt.title('Probability density image, Air')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone2)
plt.title('Probability density image, blood vessels')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone3)
plt.title('Probability density image, bone')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone4)
plt.title('Probability density image, titanium')
plt.xlabel('y')
plt.ylabel('z')
```

```
plt.colorbar()
plt.show()
```

Listing A.2: Script "single_slice_6_gaussians.py"

```
import numpy as np
import numpy.ma as ma;
import scipy.ndimage as ndi
import skimage.morphology as morf
import skimage.filters as filt
import skimage.feature as feature
import matplotlib.pyplot as plt
from scipy.ndimage import gaussian_filter1d
from scipy.optimize import curve_fit
from scipy.optimize import minimize
#from scipy import polyfit

tomo_data = np.load("770-slice-1500_1505.npy")

(Nz, Ny, Nx) = tomo_data.shape
R = Nx / 2
xs = np.linspace(-R, R, Nx)
ys = np.linspace(-R, R, Ny)

rs = np.sqrt(xs[None, :] ** 2 + ys[:, None] ** 2)  # rs[i,j] =
    sqrt(xs[j]^2 + ys[i]^2)
mask = rs >= R  # mask[i,j] == rs[i,j] >= R, dtype==bool

tomo_slice = ma.masked_array(tomo_data[3], mask=mask)

plt.figure(figsize=(15,15))
plt.imshow(tomo_slice)
plt.show()

plt.figure(figsize=(15,15))
plt.hist(tomo_slice.compressed(), bins=2000)
plt.show()

#(vmin, vmax) = (tomo_slice.min(), tomo_slice.max())
(vmin, vmax) = (-4,12)
#print(vmin, vmax)

nbins = 2 ** 16 +1
bin_edges = np.linspace(vmin, vmax, nbins)
values = (bin_edges[1:] + bin_edges[:-1]) / 2

#counts, _ = np.histogram(tomo_slice.compressed(), bins=bin_edges
    )

counts, _ = np.histogram(values[tomo_slice.compressed()], bins=
    bin_edges)

#convolve with 1D gaussian

counts = gaussian_filter1d(counts, 10, mode='constant')

#print(len(counts))

plt.subplots(figsize=(20, 5))
```

```
plt.plot(values, counts)
plt.show()

x = np.array([2,5,7,0,1,3,6,3,3,7,0,9,1,3])
y = np.linspace(0,1,10)

z = y[x]
print(z)

test = values[tomo_slice]
print(len(test))
print(len(tomo_slice))

def gaussian(x, a, b, c):
return a * np.exp(-((x-b)**2) / (2*c)**2)

def testfunc3(xs, a1, a2, a3, b1, b2, b3, c1, c2, c3):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3)

def testfunc4(xs, a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4)
    :
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4)

def testfunc5(xs, a1, a2, a3, a4, a5, b1, b2, b3, b4, b5, c1, c2,
     c3, c4, c5):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4) +
    gaussian(xs, a5, b5, c5)

def testfunc6(xs, a1, a2, a3, a4, a5, a6, b1, b2, b3, b4, b5, b6,
     c1, c2, c3, c4, c5, c6):
return gaussian(xs, a1, b1, c1) + gaussian(xs, a2, b2, c2) +
    gaussian(xs, a3, b3, c3) + gaussian(xs, a4, b4, c4) +
    gaussian(xs, a5, b5, c5) + gaussian(xs, a6, b6, c6)

print('almost')
popt2, pcov2 = curve_fit(testfunc6, values, counts, bounds
    =([1000, 500, 10, 500, 500, 100, -1, -1, -0.8, 0, 0, 6, 0, 0,
     0, 0, 0, 0], [10000, 10000, 1000, 10000, 10000, 10000, -0.6,
     -0.6, 0, 2, 2, 8, 1, 1, 1, 1, 1, 10]))
print('alomst...')
probability1 = (gaussian(values, popt2[0], popt2[6], popt2[12]) +
     gaussian(values, popt2[1], popt2[7], popt2[13])) / (
    testfunc6(values, *popt2) + 0.01)
probability2 = gaussian(values,popt2[2],popt2[8],popt2[14]) / (
    testfunc6(values, *popt2) + 0.01)
probability3 = (gaussian(values,popt2[3],popt2[9],popt2[15]) +
    gaussian(values,popt2[4],popt2[10],popt2[16])) / (testfunc6(
    values, *popt2) + 0.01)
probability4 = gaussian(values,popt2[5],popt2[11],popt2[17]) / (
    testfunc6(values, *popt2) + 0.01)

min2 = np.max(np.where(probability1 > 0.5)[0])
```

```
max2 = np.min(np.where(probability3 > 0.5)[0])
where2 = np.logical_and(tomo_slice >= values[min2], tomo_slice <=
     values[max2])

probmap2 = probability2[tomo_slice]
plt.figure(figsize=(15,15))
plt.imshow(probmap2)
plt.colorbar()
plt.show()

print('here')
print(min2)
print(max2)
probability2[0:min2] = 0 * probability2[0:min2]
probability2[max2:len(probability2)] = 0 * probability2[max2:len(
    probability2)]

plt.subplots(figsize=(20, 5))
plt.plot(probability1)
plt.plot(probability2)
plt.plot(probability3)
plt.plot(probability4)
plt.show()

test1 = values[tomo_slice]
print('test1 length =', len(test1))
print('test1 shape =', test1.shape)
test1comp = values[tomo_slice.compressed()]
index = np.unique(test1comp)
index = np.sort(index)
print('index length =', len(index))
print('test1comp length =', len(test1comp))


indicies = np.arange(0, len(values))

np.min(index)
count1 = 0
initposit = np.where(test1 == index[5])
x = test1.shape
print(x)
new_tomo = np.zeros(x)
print(initposit)
print(np.where(values == index[5]))
print(values[np.where(values == index[5])])
print(index[6])
#test2 = values[index]
#print(np.max(test2))
#print(len(test2))

plt.subplots(figsize=(20, 5))
plt.plot(values, gaussian(values,popt2[0],popt2[6],popt2[12]),
    color='m')
plt.plot(values, gaussian(values,popt2[1],popt2[7],popt2[13]),
    color='y')
```

```
plt.plot(values, gaussian(values,popt2[2],popt2[8],popt2[14]),
    color='k')
plt.plot(values, gaussian(values,popt2[3],popt2[9],popt2[15]),
    color='r')
plt.plot(values, gaussian(values,popt2[4],popt2[10],popt2[16]),
    color='g')
plt.plot(values, gaussian(values,popt2[5],popt2[11],popt2[17]),
    color='c')
plt.plot(values, counts)
plt.show()



#new_tomo1 = values[tomo_slice]
print('max of tomo_slice', np.max(tomo_slice))
print('min of tomo_slice', np.min(tomo_slice))
print(nbins)

zone1 = probability1[tomo_slice]
zone2 = probability2[tomo_slice]
zone3 = probability3[tomo_slice]
zone4 = probability4[tomo_slice]

plt.figure(figsize=(20,20))
plt.imshow(zone1)
plt.title('Probability density image, Air')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone2)
plt.title('Probability density image, blood vessels')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone3)
plt.title('Probability density image, bone')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()

plt.figure(figsize=(20,20))
plt.imshow(zone4)
plt.title('Probability density image, titanium')
plt.xlabel('y')
plt.ylabel('z')
plt.colorbar()
plt.show()
```

Listing A.3: Script "find_po_and_bounds.py"

```
import numpy as np


def find_bounds_and_init_guess(local_peakcount, local_peaks, p0,
    bounds, bins, bintype, lines, i):

for j in range(local_peakcount):

lineslice = (lines[bintype][i] == local_peaks[j])
height = np.max(lineslice * bins[i])
height_index = np.max(np.where(bins[i] == height))

if local_peakcount == 1:
gap = 500
if j < (local_peakcount - 1) and j > 0:
gap = np.min([np.min(np.where(lines[bintype][i] == local_peaks[j
    + 1])) - np.max(np.where(lineslice)),
np.min(np.where(lineslice)) - np.max(np.where(lines[bintype][i]
    == local_peaks[j - 1]))])
elif j < (local_peakcount - 1):
gap = np.min(np.where(lines[bintype][i] == local_peaks[j + 1])) -
     np.max(np.where(lineslice))
elif j > 0:
gap = np.min(np.where(lineslice)) - np.max(np.where(lines[bintype
    ][i] == local_peaks[j - 1]))

width = float(gap / 2)

p0[j] = height
                                                                #set
     height initial guess, A
bounds[0][j] = height * 0.2
                                                #set min height
    bound, A
bounds[1][j] = height * 1.01 + 0.0001
                                          #set max height bound, A
p0[j + local_peakcount] = height_index
                                        #set displacement initial
    guess, B
bounds[0][j + local_peakcount] = height_index - width / 8
                  #set min displacement bound, B
bounds[1][j + local_peakcount] = height_index + width / 8
                  # #set max displacement bound, B

right_peakedge = np.max(np.where(lineslice))
left_peakedge = np.min(np.where(lineslice))

left_side_peak = np.where(bins[i] > 0) * (np.where(bins[i] > 0) <
     left_peakedge)
left_side_above_value = np.where(bins[i] < height / 2)
left_half_peak_gapindex = 1
try:
left_half_peak_gapindex = np.max(np.intersect1d(
    left_side_above_value, left_side_peak))
```

```
except ValueError:
pass

right_side_peak = np.where(bins[i] > 0) * (np.where(bins[i] > 0)
    > right_peakedge)
right_side_above_value = np.where(bins[i] < height / 2)

right_half_peak_gapindex = 1
try:
right_half_peak_gapindex = np.min(np.intersect1d(
    right_side_above_value, right_side_peak)[
np.nonzero(np.intersect1d(right_side_above_value, right_side_peak
    ))])
except ValueError:
pass

leftwidth = np.abs(left_half_peak_gapindex - np.max(np.where(
    lineslice * bins[i] == height)))
rightwidth = np.abs(right_half_peak_gapindex - np.max(np.where(
    lineslice * bins[i] == height)))

#print(((leftwidth + rightwidth) / 2) /(height+0.0000001))
if (((leftwidth + rightwidth) / 2) /(height+0.0000001) > 0.001):
           #tries to work out if the previous has found the wrong
     location for width, at half peak height.
if height_index < 1200:
                                                #checks that it
    is NOT the metal at a position above 1200. IMPORTANT, only
    tested on x, may break for y, z or r


left_side_above_value = np.where(bins[i] < height * 9 / 10)     #
    choses what new peak height to use to determine the width
left_half_peak_gapindex = 1
try:
left_half_peak_gapindex = np.max(np.intersect1d(
    left_side_above_value, left_side_peak))
except ValueError:
pass

right_side_above_value = np.where(bins[i] < height * 9 / 10)    #
    choses what new peak height to use to determine the width
right_half_peak_gapindex = 1
try:
right_half_peak_gapindex = np.min(
np.intersect1d(right_side_above_value, right_side_peak)[np.
    nonzero(
np.intersect1d(right_side_above_value, right_side_peak))])
except ValueError:
pass

if (((leftwidth + rightwidth) / 2) / (height + 0.0000001) > 0.01)
    :
if height_index < 1200:  # checks that it is NOT the metal at a
    position above 1200. IMPORTANT, only tested on x, may break
    for y, z or r
```

```
left_side_above_value = np.where(
bins[i] < height * 99 / 100)  # choses what new peak height to
    use to determine the width
left_half_peak_gapindex = 1
try:
left_half_peak_gapindex = np.max(np.intersect1d(
    left_side_above_value, left_side_peak))
except ValueError:
pass

right_side_above_value = np.where(
bins[i] < height * 99 / 100)  # choses what new peak height to
    use to determine the width
right_half_peak_gapindex = 1
try:
right_half_peak_gapindex = np.min(
np.intersect1d(right_side_above_value, right_side_peak)[np.
    nonzero(
np.intersect1d(right_side_above_value, right_side_peak))])
except ValueError:
pass


leftwidth = np.abs(left_half_peak_gapindex - np.max(np.where(
    lineslice * bins[i] == height)))
rightwidth = np.abs(right_half_peak_gapindex - np.max(np.where(
    lineslice * bins[i] == height)))

bounds[0][j + local_peakcount * 2] = (np.min([leftwidth,
    rightwidth]) / 1.3) ** 2           #set min width bound, C
bounds[1][j + local_peakcount * 2] = (np.max([leftwidth,
    rightwidth]) * 1.3) ** 2 + 0.0001  #set min width vound, C

bounds[0][j + local_peakcount * 3] = 1.5
                                #set min exponent bound, D
bounds[1][j + local_peakcount * 3] = 2
                                  #set max exponent bound, D

p0[j + local_peakcount * 2] = ((leftwidth + rightwidth) / 2) ** 2
       #set width initial guess, C
p0[j + local_peakcount * 3] = 2
                                         #set exponent initial
    guess, C

return np.sqrt(bounds), np.sqrt(p0)     #return bounds and intial
     guess
```

Listing A.4: Script "fit_curve.py"

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit
from find_p0_and_bounds import find_bounds_and_init_guess


# define exponential functions
def gaussian(x, a, b, c, d):
return a ** 2 * np.exp(-np.abs((x - b ** 2)) ** (d ** 2) / (2 * c
     ** 2))


def gaussians2(xs, a1, a2, b1, b2, c1, c2, d1, d2):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    )


def gaussians3(xs, a1, a2, a3, b1, b2, b3, c1, c2, c3, d1, d2, d3
    ):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3)


def gaussians4(xs, a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4
    , d1, d2, d3, d4):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4)


def gaussians5(xs, a1, a2, a3, a4, a5, b1, b2, b3, b4, b5, c1, c2
    , c3, c4, c5, d1, d2, d3, d4, d5):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4) + gaussian(xs, a5, b5, c5, d5)


def gaussians6(xs, a1, a2, a3, a4, a5, a6, b1, b2, b3, b4, b5, b6
    , c1, c2, c3, c4, c5, c6, d1, d2, d3, d4, d5, d6):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4) + gaussian(xs, a5, b5, c5, d5) + gaussian(xs, a6, b6, c6,
     d6)


def fit_curve(data, bintype):

#define a dictionary of the functions
fndict = {"A": gaussian, "B": gaussians2, "C": gaussians3, "D":
    gaussians4, "E": gaussians5, "F": gaussians6}
keynames = ["A", "B", "C", "D", "E", "F"]

bins1, bins1_lines = data
```

```
lines = bins1_lines[()]
bins = bins1[bintype]
shapedepth, shapecount = bins.shape #shapedepth is the number of
    histograms, shapecount is the frequencies of the histograms

peakcount = np.size(np.unique(lines[bintype]) ) -1      # the
    number of materials in the whole space
peakmax = np.max(lines[bintype])                        # the
    highest numerical index of the materials

# totalcurves = np.zeros((shapedepth, shapecount))   #commented
    out as not used
individual_curves = np.zeros((peakmax, shapedepth, shapecount)) #
    these are the individual peaks for each material, acorss the
    whole space
curve_values = np.zeros((peakmax, shapedepth, 4))   #these are
    the ABCD values for each material, across the whole space
probabilities = np.zeros((peakmax, shapedepth, shapecount))     #
    these are the values for the probability that this is the
    material, at each point in the space.
counts = np.arange(shapecount)
counts = counts.astype(float)

for i in range( shapedepth):

# a personal check to see if it is working still. Currently
    Disabled
if i% 250 == 0:
print(i)

local_peaks_bad, ind = np.unique(lines[bintype][i], return_index=
    True)
local_peaks = local_peaks_bad[np.argsort(ind)]
local_peaks = np.delete(local_peaks, 0)
local_peakcount = np.size(local_peaks)

if local_peakcount == 0:
continue

bounds = (np.zeros(local_peakcount * 4), np.zeros(local_peakcount
     * 4))
p0 = np.zeros(local_peakcount * 4)

bounds, p0 = find_bounds_and_init_guess(local_peakcount,
    local_peaks, p0, bounds, bins, bintype, lines, i)  # generate
     bounds and initial guess

popt2, pcov2 = curve_fit(fndict[keynames[local_peakcount - 1]],
    counts, bins[i], maxfev=200000, p0=p0, bounds=bounds)  # fits
     the curve

# calculates and saves the probabilities
for m in range(local_peakcount):

# saves the values of ABCD in the correct index for each material
curve_values[local_peaks[m] - 1, i, 0] = popt2[m]
```

```
curve_values[local_peaks[m] - 1, i, 1] = popt2[m +
    local_peakcount]
curve_values[local_peaks[m] - 1, i, 2] = popt2[m +
    local_peakcount * 2]
curve_values[local_peaks[m] - 1, i, 3] = popt2[m +
    local_peakcount * 3]

# gets the intensity values for each material at a this slice of
    the data
individual_curves[local_peaks[m] - 1, i, :] = gaussian(counts,
    popt2[m], popt2[m + local_peakcount],
popt2[m + local_peakcount * 2],
popt2[m + 3 * local_peakcount])[:]

# works out probabilities as h_i / (H + epsilon)
probabilities[local_peaks[m] - 1, i] = gaussian(counts, popt2[m],
     popt2[m + local_peakcount],
popt2[m + local_peakcount * 2],
popt2[m + 3 * local_peakcount]) / (
fndict[keynames[local_peakcount - 1]](counts,
*popt2) + np.abs(
fndict[keynames[local_peakcount - 1]](counts, *popt2) < 0.1))

# '''
# prints out the graphs periodically, if you want to do that!
# Change the condition to choose how often it prints out a
    histogram
# '''
if i % 250 == 100000:

plt.subplots(figsize=(20, 5))
plt.title(i)
plt.plot(counts, fndict[keynames[local_peakcount - 1]](counts, *
    popt2), color='c')
plt.plot(counts, bins[i], color='r', lw=2, ls='—')
for k in range(local_peakcount):
plt.plot(counts, gaussian(counts, popt2[k], popt2[k +
    local_peakcount], popt2[k + local_peakcount * 2],
popt2[k + 3 * local_peakcount]))
plt.show()

plt.subplots(figsize=(20, 5))
plt.title(i)
plt.plot(counts, fndict[keynames[local_peakcount - 1]](counts, *
    popt2))
plt.plot(counts, bins[i])
plt.show()
# '''

# this plots the probabilities after all data has been fitted
'''
peakindex = np.delete(np.unique(lines[bintype]), 0)

for i in range(peakcount):
plt.title(peakindex[i] - 1)
plt.imshow(probabilities[peakindex[i] - 1])
```

```
plt.show()
'''

return probabilities, curve_values, individual_curves
```

Listing A.5: Script "optimise_fit.py"

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import minimize
from find_p0_and_bounds import find_bounds_and_init_guess


NA = np.newaxis


# define exponential functions
def gaussian(x, a, b, c, d):
return a ** 2 * np.exp(-np.abs((x - b ** 2)) ** (d ** 2) / (2 * c
     ** 2))


def gaussians2(xs, a1, a2, b1, b2, c1, c2, d1, d2):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    )


def gaussians3(xs, a1, a2, a3, b1, b2, b3, c1, c2, c3, d1, d2, d3
    ):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3)


def gaussians4(xs, a1, a2, a3, a4, b1, b2, b3, b4, c1, c2, c3, c4
    , d1, d2, d3, d4):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4)


def gaussians5(xs, a1, a2, a3, a4, a5, b1, b2, b3, b4, b5, c1, c2
    , c3, c4, c5, d1, d2, d3, d4, d5):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4) + gaussian(xs, a5, b5, c5, d5)


def gaussians6(xs, a1, a2, a3, a4, a5, a6, b1, b2, b3, b4, b5, b6
    , c1, c2, c3, c4, c5, c6, d1, d2, d3, d4, d5, d6):
return gaussian(xs, a1, b1, c1, d1) + gaussian(xs, a2, b2, c2, d2
    ) + gaussian(xs, a3, b3, c3, d3) + gaussian(xs, a4, b4, c4,
    d4) + gaussian(xs, a5, b5, c5, d5) + gaussian(xs, a6, b6, c6,
     d6)


# distribution
def distribution(X, ABCD):

ABCD = np.reshape(ABCD, (4, -1))
A, B, C, D = ABCD  # len(A) = len(B) = ... = Nmaterials
```

```
return A[:, NA] ** 2 * np.exp(-np.abs((X[NA, :] - B[:, NA] ** 2))
     ** (D[:, NA] ** 2) / (
2 * C[:, NA] ** 2))  # A,B,C,D along rows, XS along columns


# energy function
def energy(ABCD, *options):

(penalty, histogram) = options

hist_length = len(histogram)
X = np.arange(hist_length)

dist_values = distribution(X, ABCD)

hist_approx = np.sum(dist_values, axis=0)
difference = histogram - hist_approx
negative_mask = difference < 0

pen = penalty * np.abs(difference * negative_mask)

return np.sum(np.abs(difference) + pen)


def optimise_fit(data, bintype, penalty=1, maxfuncev=50, ex_p0=np
    .zeros(1), ex_bounds=np.zeros(1), set_b = False):

# define a dictionary of the functions
fndict = {"A": gaussian, "B": gaussians2, "C": gaussians3, "D":
    gaussians4, "E": gaussians5, "F": gaussians6}
keynames = ["A", "B", "C", "D", "E", "F"]

bins1, bins1_lines = data

lines = bins1_lines[()]
bins = bins1[bintype]
shapedepth, shapecount = bins.shape  # shapedepth is the number
    of histograms, shapecount is the frequencies of the
    histograms

peakcount = np.size(np.unique(lines[bintype])) - 1  # the number
    of materials in the whole space
peakmax = np.max(lines[bintype])  # the highest numerical index
    of the materials

# totalcurves = np.zeros((shapedepth, shapecount))   #commented
    out as not used
individual_curves = np.zeros(
(peakmax, shapedepth, shapecount))  # these are the individual
    peaks for each material, acorss the whole space
curve_values = np.zeros(
(peakmax, shapedepth, 4))  # these are the ABCD values for each
    material, across the whole space
probabilities = np.zeros((peakmax, shapedepth,
shapecount))  # these are the values for the probability that
    this is the material, at each point in the space.
```

```
counts = np.arange(shapecount)
counts = counts.astype(float)

for i in range(shapedepth):

# a personal check to see if it is working still. Currently
    Disabled
#print(i)
if i% 250 == 0:
print(i)

local_peaks_bad, ind = np.unique(lines[bintype][i], return_index=
    True)
local_peaks = local_peaks_bad[np.argsort(ind)]
local_peaks = np.delete(local_peaks, 0)
local_peakcount = np.size(local_peaks)

bounds = (np.zeros(local_peakcount * 4), np.zeros(local_peakcount
     * 4))
p0 = np.zeros(local_peakcount * 4)


if np.sum(ex_p0**2) < 0.01:
bounds, p0 = find_bounds_and_init_guess(local_peakcount,
    local_peaks, p0, bounds, bins, bintype, lines,
i)  # generate bounds and initial guess
bounds_low = bounds[0]
bounds_high = bounds[1]
else:
#print('here!')

if set_b == False:

bounds_low = bounds[0]
bounds_high = bounds[1]
for m in range(local_peakcount):
for n in range(4):
p0[m + n*local_peakcount] = ex_p0[local_peaks[m] - 1,i,n]
#print(local_peaks[m]-1,i,n)
#print(ex_bounds[local_peaks[m] - 1,i,n])
bounds_low[m + n * local_peakcount] = ex_p0[local_peaks[m] - 1,i,
    n] - ex_bounds[local_peaks[m] - 1,i,n]
bounds_high[m + n * local_peakcount] = ex_p0[local_peaks[m] - 1,
    i, n] - ex_bounds[local_peaks[m] - 1, i, n]
#print(p0)
#print(bounds_high)
#print(bounds_low)


elif set_b == True:
bounds, p0 = find_bounds_and_init_guess(local_peakcount,
    local_peaks, p0, bounds, bins, bintype, lines,
i)  # generate bounds and initial guess
#print(bounds[0])
#print(bounds[1])
bounds_low = bounds[0]
```

```
bounds_high = bounds[1]
for m in range(local_peakcount):
#if i == 2:
#print('2 before this')
p0[m + local_peakcount]= ex_p0[local_peaks[m] - 1,i,1]
#print('here', m)
bounds_low[m + local_peakcount] = ex_p0[local_peaks[m] - 1, i, 1]
     - 0.0001
bounds_high[m + local_peakcount] = ex_p0[local_peaks[m] - 1, i,
    1] + 0.0001


#print(bounds)
#restructure the bounds to work for optimise.minimise
pass_bounds = np.zeros((len(bounds_low), 2))
#print(pass_bounds)
for p in range(len(bounds_low)):

pass_bounds[p,0] = bounds_low[p]
pass_bounds[p,1] = bounds_high[p]

#print(pass_bounds)
# bounds=pass_bounds

#print(p0)
#print(pass_bounds)
#print(bounds_low)
res = minimize(energy, p0, (penalty, bins[i]), method='Powell',
    bounds=pass_bounds, options={'maxfev': maxfuncev}) #fits by
    minimising the energy
popt2 = res.x

#print(res.x)
# calculates and saves the probabilities
for m in range(local_peakcount):
#print(m)

# saves the values of ABCD in the correct index for each material
curve_values[local_peaks[m] - 1, i, 0] = popt2[m]
curve_values[local_peaks[m] - 1, i, 1] = popt2[m +
    local_peakcount]
curve_values[local_peaks[m] - 1, i, 2] = popt2[m +
    local_peakcount * 2]
curve_values[local_peaks[m] - 1, i, 3] = popt2[m +
    local_peakcount * 3]

# gets the intensity values for each material at this slice of
    the data
individual_curves[local_peaks[m] - 1, i, :] = gaussian(counts,
    popt2[m], popt2[m + local_peakcount],
popt2[m + local_peakcount * 2],
popt2[m + 3 * local_peakcount])[:]

# works out probabilities as h_i / (H + epsilon)
probabilities[local_peaks[m] - 1, i] = gaussian(counts, popt2[m],
     popt2[m + local_peakcount],
```

```
popt2[m + local_peakcount * 2],
popt2[m + 3 * local_peakcount]) / (
fndict[keynames[local_peakcount - 1]](counts,
*popt2) + np.abs(
fndict[keynames[local_peakcount - 1]](counts, *popt2) < 0.1))

# '''
# prints out the graphs periodically, if you want to do that!
    Change the condition to choose how often it prints out a
    histogram
# '''

if i == 100000: #% 250 == 0:


plt.subplots(figsize=(20, 5))
plt.title(i)
plt.plot(counts, fndict[keynames[local_peakcount - 1]](counts, *
    popt2), color='c')
plt.plot(counts, bins[i], color='r', lw=2, ls='-')
for k in range(local_peakcount):
plt.plot(counts, gaussian(counts, popt2[k], popt2[k +
    local_peakcount], popt2[k + local_peakcount * 2],
popt2[k + 3 * local_peakcount]))
plt.show()


title_string = str(penalty) + ', ' + str(i)

plt.subplots(figsize=(20, 5))
plt.title(title_string)
plt.plot(counts, fndict[keynames[local_peakcount - 1]](counts, *
    popt2))
plt.plot(counts, bins[i])
plt.show()

'''
# this plots the probabilities after all data has been fitted
peakindex = np.delete(np.unique(lines[bintype]), 0)

for i in range(peakcount):
continue
title_string = str(penalty) + ', ' + str(peakindex[i] - 1)
plt.title(title_string)
plt.imshow(probabilities[peakindex[i] - 1])
plt.show()
'''

return probabilities, curve_values, individual_curves
```

Listing A.6: Script "cubic.py"

```
from numpy import array, linspace, sin, empty, zeros, linalg,
    random, pad, concatenate
import numpy as np

# Scheme:
#  Fit an N-segment piecewise cubic polynomial to a set of points
     with linear least squares with
#  two exact conditions:
#   (1) Continuity at the borders:        f_n (X_n) = f_{n-1} (
    X_n)
#   (2) Differentiability at the borders: f_n'(X_n) = f'_{n-1}(
    X_n)
#
#  Input: A list of M data points [(x1,y1),....(x_M,y_M)] and
#         A list of N+1 borders   [X0,X1,....,X_N]
#
#  This file generalizes the 2-segment method given in cubic2.py
    to an arbitrary
#  number of segments.
#          / f1(x) = A1 + B1*(x-X0) + C1*(x-X0)**2 + D1*(x-X0)
    **3          if x>=X0 & x<=X1
#    f(x) = |
#          | f2(x) = f1(X1) + f1'(X1)*(x-X2) + C2*(x-X2)**2 + D2
    *(x-X2)**3 if x>=X1 & x<=X2
#          |
#          | f3(x) = f2(X2) + f2'(X2)*(x-X3) + C2*(x-X3)**2 + D2
    *(x-X3)**3 if x>=X2 & x<=X3
#          |   ...
#          \ f_N(x) = f_{N-1}(X_{N-1}) + f'_{N-1}(X_{N-1})*(x-
    X_N) + C_N*(x-X_N)**2 + D_N*(x-X_N)**3
#

    if x>=X_{N-1} & x<= X_N
#
# We still want to set up a linear least squares system of
    equations that implicitly
# obeys the two exact conditions (so they are not weakened by the
     least squares
# approximation to the over-determined system).

# Eq. (1) and (2) determine A_n (the function value at X_n) and
    B_n (the derivative at X_n).
# So we have 4+2*(N-1) = 2*(N+1) variables to determine: A1, B1,
    C1, D1, and C2, D2, C3, D3, ..., C_N, D_N

# Each data point (x_i,y_i)  defines a row with the coefficients
    for A1,B1,C1,D1,C2,D2,...,C_N,D_N in the matrix, and y_i on
    the RHS.
def mxrow_f (x, borders):
if(len(borders) < 2):
printf(f"No segments: borders = {borders}")
return

Xleft, Xright = borders[-2:] # x >= Xleft & x <= Xright
```

```
n  = len(borders)-2         # n is the current segment number
#    print(f"f_{n}({x}): Xleft,Xright = {np.round([Xleft,Xright
    ],2)}; borders = {borders}, {len(borders)-1} segments")

if n==0:
X = x-Xleft
return array([ 1, X, X**2,  X**3])
else:
# We recursively define the matrix row
# f_n(x) = f_{n-1}(Xn) + f'_{n-1}(Xn)*(x-Xn) + Cn*(x-Xn)**2 + Dn
    *(x-Xn)**3
#        print(f"Recursing down from level {n} to level {n-1},
    evaluated at {Xleft}")
row               = zeros((4+2*n,), dtype=float)
row[:(4+2*(n-1))] = mxrow_f(Xleft,borders[:-1]) + mxrow_df(Xleft,
    borders[:-1])*(x-Xleft);
row[-2:]          = [(x-Xleft)**2, (x-Xleft)**3]
#        print(f"returning length = {len(row)} row at level {n}")
return row

def mxrow_df(x,borders):
if len(borders) < 2:
print(f"No segments:  borders={borders}")
return

Xleft, Xright = borders[-2:] # x >= Xleft & x <= Xright
n  = len(borders)-2         # n is the current segment number
#    print(f"f'_{n}({x}): Xleft,Xright = {np.round([Xleft,Xright
    ],2)}; borders = {borders}, {len(borders)-1} segments")

if n==0:
X0 = borders[0]
return array([ 0,    1, 2*(x-X0) ,3*(x-X0)**2])
else:
# We recursively define the matrix row: f'_n(x) = f'_{n-1}(Xn) +
    Cn*2*(x-Xn) + Dn*3*(x-Xn)
row               = zeros((4+2*n,), dtype=float)
row[:(4+2*(n-1))] = mxrow_df(Xleft,borders[:-1])
row[-2:]          = [2*(x-Xleft), 3*((x-Xleft)**2)]
return row


# We can now put these together to construct the matrix and RHS
    row by row:
def piecewisecubic_matrix(xs,ys, Xs):
M = len(xs)                 # M data points
N = len(Xs)-1               # N segments, i.e. N+1 borders

A = empty((M,2*(N+1)),dtype=float)
b = empty((M,1),dtype=float)

n = 0                        # Start in first region
Xleft, Xright = Xs[0], Xs[1]

for i in range(len(xs)):
```

```
#        print(f"Xleft, Xright = {Xleft, Xright}, Xs = {Xs}, n =
    {n}")
if(xs[i] > Xright):
n += 1
Xleft, Xright = Xs[n], Xs[n+1]

#        print(f"A[{i}]: n = {n}, 4+2n = {4+2*n}, n+2 = {n+2}")
A[i,:(4+2*n)] = mxrow_f(xs[i],Xs[:(n+2)])
b[i] = ys[i]

return A,b

# A function that takes an N-segment piecewise cubic polynomial
# produced by fit_piecewisecubic() and evaluates it on an
    arbitrary
# set of coordinates xs (not necessarily the same as the data
    points
# it was fitted to):
def piecewisecubic(pc,all_xs):
coefs, Xs = pc          # Polynomial coefficients A1,B1,C1,D1,C2,
    D2,C3,D3,... and borders
N = len(Xs)-1           # N segments have N+1 borders: |seg1|seg2
    |...|segN|

ys = []                 # List of function values for segments
A,B = coefs[:2]         # A and B coefficients are only defined
    for 0-segment

for n in range(N):
C, D          = coefs[(2+2*n):(2+2*n+2)] #

Xleft, Xright = Xs[n], Xs[n+1]
xs_segment    = all_xs[(all_xs>=Xleft) & (all_xs< Xright)]


# f_n(x) = A + B*(x-Xn) + C*(x-Xn)**2 + D*(x-Xn)**3
xs   = xs_segment - Xleft                # Segment coordinate x-
    Xn
ys += [A + B*xs + C*(xs**2) + D*(xs**3)]  # f_n(xs)

# Calculate next A and B coefficients:
X = Xright-Xleft
A = A + B*X + C*(X**2) + D*(X**3)   # A_{n+1} = f_n (X_{n+1})
B =     B   + C*2*X    + D*3*(X**2) # B_{n+1} = f'_n(X_{n+1})

return concatenate(ys)



# ...and a function to construct the overdetermined linear system
     of
# equations and find the least squares optimal approximate
    solution:
def fit_piecewisecubic(xs,ys, Xs):
A, b = piecewisecubic_matrix(xs,ys,Xs)
```

```
coefs, residuals, rank, sing = linalg.lstsq(A,b,rcond=None)

return (coefs,Xs)

def cubic_interpolation(curve_values, segments=5, plot=False):

new_curve_vals = np.zeros_like(curve_values)
for i in range(len(curve_values)):
for j in range(4):

print(i,j)
if np.sum(curve_values[i, :, j]**2) < 0.01:
print('ignored')
continue

vals = curve_values[i, :, j].T

mask_abovezero = (vals > 0.000001)

val_mean = np.mean(vals[vals> 0.0001])
val_std = np.std(vals[vals> 0.0001])

if i == 6:

print(f"mean, std = {val_mean, val_std}")

mask = (vals > 0.000001) & (np.abs(vals-val_mean) < 3*val_std)

xs = np.argwhere(mask).astype(float).flatten()
ys = vals[mask]
borders = np.linspace(xs.min(), xs.max() + 1, segments + 1)

pc = fit_piecewisecubic(xs, ys, borders)
Xs = np.arange(len(curve_values[i,:]))
Ys = piecewisecubic(pc, xs)

YXs = piecewisecubic(pc, Xs)


for k in range(int(np.min(borders)),int(np.max(borders)), 1):

new_curve_vals[i, k, j] = YXs[k-int(np.min(borders))]

return  new_curve_vals
```

Listing A.7: Script "find_lines.py"

```
import cv2
import numpy as np
from numpy.lib.function_base import disp
from scipy import signal
from scipy.ndimage import gaussian_filter1d
from moviepy.video.io.bindings import mplfig_to_npimage
import matplotlib.pyplot as plt
from skimage.morphology import skeletonize
import json
import argparse
import os

def batch():
global args

if not os.path.exists(args.output):
os.mkdir(args.output)

for histogram in args.histogram:
sample = ''.join(os.path.basename(histogram).split('.')[:-1])
f = np.load(histogram)
for name, bins in f.items():
rng = _range(0,bins.shape[1],0,bins.shape[0])
px, py = scatter_peaks(bins)
mask = np.zeros(bins.shape, dtype=np.uint8)
mask[py, px] = 255
dilated, eroded = process_closing(mask)
labeled = process_contours(eroded, rng)

np.save(f'{args.output}/{sample}_{name}_labeled', labeled)

if args.verbose:
print (f'Processed {sample}')

def load_config(filename):
if os.path.exists(filename):
with open(filename, 'r') as f:
config = json.load(f)
else:
config = {
'min peak height': 1,
'close kernel y': 10,
'close kernel x': 2,
'line smooth': 7,
'iter dilate': 10,
'iter erode': 5,
'min contour size': 2000,
'joint kernel size': 2
}
return config

def load_hists(filename):
hists = np.load(filename)
results = {}
```

```
for name, hist in hists.items():
hist_sum = np.sum(hist, axis=1)
hist_sum[hist_sum==0] = 1
results[name] = hist / hist_sum[:,np.newaxis]
return hists

class _range:
class inner_range:
start = 0
stop = 0

x = inner_range()
y = inner_range()

def update(self):
self.x.start = cv2.getTrackbarPos('range start x', 'Histogram
    lines')
self.x.stop = cv2.getTrackbarPos('range stop x', 'Histogram lines
    ')
self.y.start = cv2.getTrackbarPos('range start y', 'Histogram
    lines')
self.y.stop = cv2.getTrackbarPos('range stop y', 'Histogram lines
    ')
return self

def __init__(self, range_x_start=0, range_x_stop=0, range_y_start
    =0, range_y_stop=0):
self.x.start = range_x_start
self.x.stop = range_x_stop
self.y.start = range_y_start
self.y.stop = range_y_stop

def parse_args():
parser = argparse.ArgumentParser(description="Computes the
    connected lines in a 2D histogram. It can either be run in
    GUI mode, where one tries to find the optimal configuration
    parameters, or batch mode, where it processes one or more
    histograms into images. In GUI mode, one can specify the
    bounding box by dragging a box on one of the images, specify
    the line to highlight by left clicking and reset the bounding
     box by middle clicking.")

# E.g. histogram: /mnt/data/MAXIBONE/Goats/tomograms/processed/
    histograms/770c_pag/bins1.npz
# TODO glob support / -r --recursive
parser.add_argument('histogram', nargs='+',
help='Specifies one or more histogram files (usually *.npz) to
    process. If in GUI mode, only the first will be processed.
    Glob is currently not supported.')
parser.add_argument('-b', '--batch', action='store_true',
help='Toggles whether the script should be run in batch mode. In
    this mode, the GUI isn\'t launched, but the provided
    histograms will be stored in the specified output folder.')
parser.add_argument('-c', '--config', default='config.json', type
    =str,
```

```
help='The configuration file storing the parameters. If in GUI
    mode, this will be overwritten with the values on the
    trackbars (unless the -b flag is provided). If it doesn\'t
    exist, the default values inside this script will be used.
    NOTE: there\'s an error in libxkbcommon.so, which crashes
    OpenCV whenever any other key than escape is used. So to save
     it, close the GUI with the escape button.')
parser.add_argument('-d', '--dry_run', action='store_true',
help='Toggles whether the configuration should be saved, when
    running in GUI mode.')
parser.add_argument('-o', '--output', default='output', type=str,
help='Specifies the folder to put the resulting images in.')
parser.add_argument('-p', '--peaks', action='store_true',
help="Toggles whether to plot peaks on the cutout (1st row, 3rd
    image). Turned off by default, since it is computationally
    heavy. Only applicable in GUI mode.")
parser.add_argument('-v', '--verbose', action='store_true',
help='Toggles whether debug printing should be enabled.')

args = parser.parse_args()
return args

def plot_line(line, rng: _range):
global config

meaned, peaks = process_line(line[rng.x.start:rng.x.stop])
fig, ax = plt.subplots()
ax.plot(line[rng.x.start:rng.x.stop])
ax.plot(meaned)
ax.scatter(peaks, meaned[peaks], c='red')
line_plot = mplfig_to_npimage(fig)
line_plot = cv2.cvtColor(line_plot, cv2.COLOR_RGB2BGR)
plt.close()

return line_plot

def process_closing(mask):
global config

close_kernel = np.ones((config['close kernel y'], config['close
    kernel x']))
dilated = cv2.dilate(mask, close_kernel, iterations=config['iter
    dilate'])
eroded = cv2.erode(dilated, close_kernel, iterations=config['iter
     erode'])

return dilated, eroded

def process_contours(hist, rng: _range):
global config

contours, _ = cv2.findContours(hist, cv2.RETR_TREE, cv2.
    CHAIN_APPROX_SIMPLE)
contours = [c for c in contours if cv2.contourArea(c) > config['
    min contour size']]
```

```
result = np.zeros((rng.y.stop-rng.y.start, rng.x.stop-rng.x.start
    ), np.uint8)
bounding_boxes = [cv2.boundingRect(c) for c in contours]
(contours, _) = zip(*sorted(zip(contours, bounding_boxes), key=
    lambda b:b[1][0]))
for i in np.arange(len(contours)):
result = cv2.drawContours(result, contours, i, int(i+1), -1)

return result, [i+1 for i in range(len(contours))]

def process_joints(hist):
global config

skeleton = skeletonize(hist)
skeleton_gray = cv2.cvtColor(skeleton, cv2.COLOR_BGR2GRAY)
skeleton_gray = cv2.threshold(skeleton_gray, 1, 255, cv2.
    THRESH_BINARY)[1]
skeleton_gray = cv2.dilate(skeleton_gray, (3,3), 10)
joint_kernel_size = config['joint kernel size']
horizontal_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (
    joint_kernel_size,1))
horizontal = cv2.morphologyEx(skeleton_gray, cv2.MORPH_OPEN,
    horizontal_kernel, iterations=2)
vertical_kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (1,
    joint_kernel_size))
vertical = cv2.morphologyEx(skeleton_gray, cv2.MORPH_OPEN,
    vertical_kernel, iterations=2)
joints = cv2.bitwise_and(horizontal, vertical)
joints = cv2.dilate(joints, (10,10), iterations=5)

return joints

def process_line(line):
global config

meaned = gaussian_filter1d(line, config['line smooth'])
peaks, _ = signal.find_peaks(meaned, .01*config['min peak height'
    ]*line.max())

return meaned, peaks

def process_scatter_peaks(hist, rng: _range):
global config, args

# Show the peaks scattered on the cutout of the original
px, py = scatter_peaks(hist[rng.y.start:rng.y.stop,rng.x.start:
    rng.x.stop])
fig, ax = plt.subplots()
ax.imshow(hist[rng.y.start:rng.y.stop,rng.x.start:rng.x.stop],
    cmap='jet')
if args.peaks:
ax.scatter(px, py, color='red', alpha=.008)
scatter_plot = mplfig_to_npimage(fig)
scatter_plot = cv2.cvtColor(scatter_plot, cv2.COLOR_RGB2BGR)
plt.close()
```

```
return scatter_plot, py, px

def process_with_box(hist, rng: _range, selected_line, scale_x,
    scale_y, partial_size):
display = ((hist.astype(np.float32) / hist.max()) * 255.0).astype
    (np.uint8)
box_x_start = int(rng.x.start * scale_x)
box_y_start = int(rng.y.start * scale_y)
box_x_stop = int(rng.x.stop * scale_x)
box_y_stop = int(rng.y.stop * scale_y)
resized = cv2.resize(display, partial_size)
colored = cv2.cvtColor(resized, cv2.COLOR_GRAY2BGR)
colored = cv2.rectangle(colored, (box_x_start, box_y_start), (
    box_x_stop, box_y_stop), (0, 255, 0), 1)
colored[int(selected_line*scale_y),:] = (0,0,255)

return colored

def save_config():
global config, args

with open(args.config, 'w') as f:
json.dump(config, f)

def scatter_peaks(hist):
peaks_x = []
peaks_y = []
for i in range(len(hist)):
_, peaks = process_line(hist[i,:])
line_y = [i for _ in peaks]
peaks_x += list(peaks)
peaks_y += line_y

return peaks_x, peaks_y

def gui():
global last_bin, args
def update_image(_):
hist_shape = f[keys[cv2.getTrackbarPos('bins', 'Histogram lines')
    ]].shape
cv2.setTrackbarMax('range start x', 'Histogram lines', hist_shape
    [1]-1)
cv2.setTrackbarPos('range start x', 'Histogram lines', min(cv2.
    getTrackbarPos('range start x', 'Histogram lines'),
    hist_shape[1]-1))
cv2.setTrackbarMax('range stop x', 'Histogram lines', hist_shape
    [1]-1)
cv2.setTrackbarPos('range stop x', 'Histogram lines', min(cv2.
    getTrackbarPos('range stop x', 'Histogram lines'), hist_shape
    [1]-1))
cv2.setTrackbarMax('range start y', 'Histogram lines', hist_shape
    [0]-1)
cv2.setTrackbarPos('range start y', 'Histogram lines', min(cv2.
    getTrackbarPos('range start y', 'Histogram lines'),
    hist_shape[0]-1))
```

```
cv2.setTrackbarMax('range stop y', 'Histogram lines', hist_shape
    [0]-1)
cv2.setTrackbarPos('range stop y', 'Histogram lines', min(cv2.
    getTrackbarPos('range stop y', 'Histogram lines'), hist_shape
    [0]-1))
update(42)

def update_line(event, x, y, flags, param):
global mx, my, scale_x, scale_y, last_bin

sizex = cv2.getTrackbarPos('size x', 'Histogram lines')
sizey = cv2.getTrackbarPos('size y', 'Histogram lines')
rng = _range().update()
rwidth = rng.x.stop - rng.x.start
rheight = rng.y.stop - rng.y.start
pwidth = sizex // 3
pheight = sizey // 2
dead_zone = 100 # This is in global scale
if y > sizey // 2 or x < sizex // 3:
lx = x % pwidth
ly = y % pheight
gx = int(lx * (1./scale_x)) if y < sizey // 2 else int(rng.x.
    start + lx * (1./(pwidth / rwidth)))
gy = int(ly * (1./scale_y)) if y < sizey // 2 else int(rng.y.
    start + ly * (1./(pheight / rheight)))
if (event == cv2.EVENT_LBUTTONDOWN):
print ('down', gx, gy)
mx = gx
my = gy
elif (event == cv2.EVENT_LBUTTONUP):
print ('up', gx, gy)
if abs(mx-gx) < dead_zone and abs(my-gy) < dead_zone:
cv2.setTrackbarPos('line', 'Histogram lines', gy)
print ('Set line', y, ly, gy)
update(42)
else:
mx, gx = sorted((mx,gx))
my, gy = sorted((my,gy))
cv2.setTrackbarPos('range start x', 'Histogram lines', mx)
cv2.setTrackbarPos('range stop x', 'Histogram lines', gx)
cv2.setTrackbarPos('range start y', 'Histogram lines', my)
cv2.setTrackbarPos('range stop y', 'Histogram lines', gy)
print ('Set bounds', mx, my, gy, gy)
update(42)
elif (event == cv2.EVENT_MBUTTONDOWN):
print ('reset bounding box')
hist_shape = f[keys[cv2.getTrackbarPos('bins', 'Histogram lines')
    ]].shape
cv2.setTrackbarPos('range start x', 'Histogram lines', 0)
cv2.setTrackbarPos('range stop x', 'Histogram lines', hist_shape
    [1]-1)
cv2.setTrackbarPos('range start y', 'Histogram lines', 0)
cv2.setTrackbarPos('range stop y', 'Histogram lines', hist_shape
    [0]-1)
update(42)
```

```
# TODO Only do recomputation, whenever parameters that have an
    effect are changed.
def update(_): # Note: all colors are in BGR format, as this is
    what OpenCV uses
global last_bin, config, scale_x, scale_y

update_config()

# Check if trackbar ranges should be updated
bin = cv2.getTrackbarPos('bins', 'Histogram lines')
hist = f[keys[bin]]
if bin != last_bin:
last_bin = bin
# Set trackbar max values
cv2.setTrackbarMax('range start x', 'Histogram lines', hist.shape
    [1]-1)
cv2.setTrackbarMax('range stop x', 'Histogram lines', hist.shape
    [1]-1)
cv2.setTrackbarMax('range start y', 'Histogram lines', hist.shape
    [0]-1)
cv2.setTrackbarMax('range stop y', 'Histogram lines', hist.shape
    [0]-1)
cv2.setTrackbarMax('line', 'Histogram lines', hist.shape[0]-1)

# Set trackbar starting values
cv2.setTrackbarPos('range start x', 'Histogram lines', 0)
cv2.setTrackbarPos('range stop x', 'Histogram lines', hist.shape
    [1]-1)
cv2.setTrackbarPos('range start y', 'Histogram lines', 0)
cv2.setTrackbarPos('range stop y', 'Histogram lines', hist.shape
    [0]-1)
cv2.setTrackbarPos('line', 'Histogram lines', 0)

# Show the original image, along with the line
rng = _range().update()
selected_line = cv2.getTrackbarPos('line', 'Histogram lines')

partial_width = cv2.getTrackbarPos('size x', 'Histogram lines')
    // 3
partial_height = cv2.getTrackbarPos('size y', 'Histogram lines')
    // 2
partial_size = (partial_width, partial_height)
scale_x = partial_width / hist.shape[1]
scale_y = partial_height / hist.shape[0]

colored = process_with_box(hist, rng, selected_line, scale_x,
    scale_y, partial_size)

# Show the plot of a single line
line_plot = plot_line(hist[selected_line, :], rng)
lp_resized = cv2.resize(line_plot, partial_size)

scatter_plot, py, px = process_scatter_peaks(hist, rng)
sp_resized = cv2.resize(scatter_plot, partial_size)
```

```
first_row = np.concatenate((colored, lp_resized, sp_resized),
    axis=1)

local_scale_y = partial_height / (rng.y.stop - rng.y.start)
mask = np.zeros((rng.y.stop-rng.y.start,rng.x.stop-rng.x.start),
    dtype=np.uint8)
mask[py, px] = 255

dilated, eroded = process_closing(mask)
display_eroded = cv2.resize(eroded, partial_size)
display_eroded = cv2.cvtColor(display_eroded, cv2.COLOR_GRAY2BGR)
if selected_line > rng.y.start and selected_line < rng.y.stop:
display_eroded[int((selected_line-rng.y.start)*local_scale_y),:]
    = (0,0,255)

# Find lines
labeled, labels = process_contours(eroded, rng)
label_colours = [(0,0,0)] + [(np.random.randint(0,255), np.random
    .randint(0,255), np.random.randint(0,255)) for _ in labels]
labeled_colour = np.zeros((labeled.shape[0], labeled.shape[1], 3)
    , dtype=np.uint8)
for y in range(labeled.shape[0]):
for x in range(labeled.shape[1]):
labeled_colour[y,x,:] = label_colours[labeled[y,x]]
display_contours = cv2.resize(labeled_colour, partial_size)
if selected_line > rng.y.start and selected_line < rng.y.stop:
display_contours[int((selected_line-rng.y.start)*local_scale_y)
    ,:] = (0,0,255)

# Find joints
joints = process_joints(labeled_colour)
display_joints = cv2.resize(joints, partial_size)
display_joints = cv2.cvtColor(display_joints, cv2.COLOR_GRAY2BGR)
if selected_line > rng.y.start and selected_line < rng.y.stop:
display_joints[int((selected_line-rng.y.start)*local_scale_y),:]
    = (0,0,255)

second_row = np.concatenate((display_eroded, display_contours,
    display_joints), axis=1)

cv2.imshow('Histogram lines', np.concatenate((first_row,
    second_row)))

def update_config():
global config

for entry in config.keys():
config[entry] = cv2.getTrackbarPos(entry, 'Histogram lines')

f = np.load(args.histogram[0])
keys = [key for key in f.keys()]
first_hist = f[keys[0]]
last_bin = 0
cv2.namedWindow('Histogram lines')
cv2.createTrackbar('range start x', 'Histogram lines', 0,
    first_hist.shape[1]-1, update)
```

```
cv2.createTrackbar('range stop x', 'Histogram lines', first_hist.
    shape[1]-1, first_hist.shape[1]-1, update)
cv2.createTrackbar('range start y', 'Histogram lines', 0,
    first_hist.shape[0]-1, update)
cv2.createTrackbar('range stop y', 'Histogram lines', first_hist.
    shape[0]-1, first_hist.shape[0]-1, update)
cv2.createTrackbar('size x', 'Histogram lines', 1024, 1920,
    update)
cv2.createTrackbar('size y', 'Histogram lines', 512, 1080, update
    )
cv2.createTrackbar('bins', 'Histogram lines', 0, len(keys)-1,
    update_image)
cv2.createTrackbar('line', 'Histogram lines', 0, first_hist.shape
    [0]-1, update)
cv2.createTrackbar('line smooth', 'Histogram lines', config['line
     smooth'], 50, update)
cv2.createTrackbar('min peak height', 'Histogram lines', config['
    min peak height'], 100, update)
cv2.createTrackbar('close kernel x', 'Histogram lines', config['
    close kernel x'], 100, update)
cv2.createTrackbar('close kernel y', 'Histogram lines', config['
    close kernel y'], 100, update)
cv2.createTrackbar('iter dilate', 'Histogram lines', config['iter
     dilate'], 100, update)
cv2.createTrackbar('iter erode', 'Histogram lines', config['iter
    erode'], 100, update)
cv2.createTrackbar('min contour size', 'Histogram lines', config[
    'min contour size'], 10000, update)
cv2.createTrackbar('joint kernel size', 'Histogram lines', config
    ['joint kernel size'], 100, update)
cv2.setMouseCallback('Histogram lines', update_line)
update(42)
cv2.waitKey(0) # Segfaults for some reason, when the key isn't
    escape :) Looks like it's some wayland / ubuntu 20.04 error
#while True:
# key = cv2.waitKey(16)
# key &= 0xFF
# if key == 113:
# break
cv2.destroyAllWindows()

if __name__ == '__main__':
global args

args = parse_args()
config = load_config(args.config)
if args.batch:
batch()
else:
gui()
if not args.dry_run:
save_config()
```

Listing A.8: Script "test_bones.py"

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from scipy.optimize import least_squares
from scipy.interpolate import CubicSpline
from optimise_fit import optimise_fit
from fit_curve import fit_curve
from cubic import *

np.set_printoptions(suppress=True)

bins_linesdata = np.load("bins1_lines.npz", allow_pickle=True)
binsdata = np.load("bins1.npz", allow_pickle=True)

bintype = 'x_bins'  #choose which data to look at, x, y, z or r

data = [binsdata, bins_linesdata]

bins1, bins1_lines = data

#a = np.arange(10)
#np.save('a_output_test', a)

lines = bins1_lines[()]
bins = bins1[bintype]

line_10 = bins[:, 500]

#plt.plot(line_10, np.arange(len(line_10)))
#plt.show()

'''
for i in range(10):
plt.plot(bins[:,200*i], np.arange(len(bins[:,200*i])))
plt.show()
'''
#optimise_fit(data, bintype, -100)
#optimise_fit(data, bintype, -10)
#optimise_fit(data, bintype, -5)
#optimise_fit(data, bintype, -2)
#optimise_fit(data, bintype, -1)
#optimise_fit(data, bintype, 0)


#probabilities, curve_values, individual_curves = optimise_fit(
    data, bintype, 2, 50)
#probabilities, curve_values, individual_curves = fit_curve(data,
    bintype)
'''
np.save('output_probs_x_leastsq', probabilities)
np.save('output_curve_values_x_leastsq', curve_values)
np.save('output_individual_curves_x_leastsq', individual_curves)
'''

probabilities = np.load('output_probs.npy')
curve_values = np.load('output_curve_values.npy')
individual_curves = np.load('output_individual_curves.npy')
```

```
new_p0 = cubic_interpolation(curve_values)

np.save('new_po', new_p0)
new_bounds = np.zeros_like(new_p0)
mask = new_p0 > 0.0001
new_p0_masked = new_p0[mask]
print('here')
for i in range(len(new_p0)):
for j in range(len(new_p0[0,0])):

if np.sum(curve_values[i, :, j] ** 2) < 0.01:
#print('ignored')
continue

#print(i,j)
new_p0_here = new_p0[i,:,j]
mask = new_p0_here > 0.0001
new_p0_masked = new_p0_here[mask]
new_bounds[i,:,j] = 0* np.std(new_p0_masked) + 0.0001
#if i == 5



new_probs, new_curve, new_individual = optimise_fit(data, bintype
    , 2, 50, new_p0, new_bounds, False)

#new_new_p0 = cubic_interpolation(new_curve) # ,plot = True)

#probabilities, curve_values, individual_curves = optimise_fit(
    data, bintype, 2, 50)


xs = np.arange(len(curve_values[i]))
for i in range(10):
for j in range(4):

plt.plot(xs, new_curve[i,:,j])
plt.plot(xs, curve_values[i,:,j])
plt.show()


np.save('new_output_probs_nochange', new_probs)
np.save('new_output_curve_values_nochange', new_curve)
np.save('new_output_individual_curves_nochange', new_individual)
'''
```