NTNU – Trondheim
Norwegian University of
Science and Technology

Fracture detection and analysis from image log raw data

Jan Cornet

Petroleum Geosciences
Submission date: June 2013
Supervisor: Helge Langeland, IPT

Norwegian University of Science and Technology
Department of Petroleum Engineering and Applied Geophysics

NTNU
Norwegian University of Science and Technology

Quantitative Geology from Petrophysical Measurements Project

Fracture detection and analysis on wellbore images

Applied Petroleum Geophysics
Master Thesis

Student: Jan Cornet

Supervisor: Helge Langeland

Trondheim, June, 2013

Abstract

This report presents the results obtained during the master thesis done at NTNU in the department of applied geophysics and is part of the: “Quantitative geology from petrophysical measurment” project.

The goal of this thesis was to develop a tool in Matlab to automatically interpret fractures on images logs. Since many fractures show as sinusoids on images logs this work focuses on detecting sinusoidal events in images. The programs were tested on resistivity and acoustic images provided respectively by Geostress and Statoil.

Image processing is usually made of two main steps: edge detection and feature extraction. Two different kinds of edge mapping were tested on real data: gradient based and Fourier transform based. Finally the Fourier based edge mapping was kept because it could enhance the signal of fractures and reduce the impact of lithology. Some pre-processing steps like the interpolation of images had however to be added to enable the use of Fourier. In a second part of the thesis, three different feature extraction methods found in the literature where tested on synthetic data: the correlation method, the Hough transform and the Radon transform.

The understanding of the limits of each of the three extraction methods tested previously led to the creation of an interpretation program of the “edge-following” type that uses a partitioning algorithm. An “edge-following” type was chosen for its ability to interpret fractures with imperfect sinusoidal shapes. The program was finally tested on electric and sonic image logs. The chances of properly interpreting a fracture on electric images were found to be of two over three and the processing speed of 200m/min.

The tests on real data showed that the program has troubles interpreting low contrast or high dip fractures. There are also some issues when applied on images with break outs or wash outs.

In a word, a fracture interpretation program was built using Matlab but it can still be improved.

Acknowledgements

I would like to thank the Norwegian University of Science and Technology for giving me the opportunity to come to study in Trondheim. I would especially like to express my sincere thanks to Helge Langeland, my supervisor at NTNU, for his advices during this thesis.

I place on record, my sincere gratitude to my father who provided me with data and material from Gesotress. I would also like to thank Statoil, and especially Harry Brandsen, for their commitment to this project by providing data and feedbacks.

To Daniel Quesada, who showed me that it was possible to solve my problem, I would like to thank him for the hope he gave me.

I take this opportunity to record my sincere thanks to all the faculty members of IPT for their help and especially Alexis Stovas who explained the Radon transform to me.

I would also like to thank all the people that made this semester unforgettable: Carlos Flores Zamora, Eric Ameyaw Debrah and all the others.

Finally to my family and to my friends back in France.

Contents

Introduction

With the continuous demand for hydrocarbons and the depletion of reservoirs, it is critical to extract as much hydrocarbons from reservoir as possible. This means increasing the knowledge we have of reservoirs in order to design the best production strategy and achieve a high recovery factor.

At NTNU this issue is addressed from different point of views and especially through the project called *Quantitative Geology from Petrophysical Measurements*, which this work is a part of. The idea is to extract more information from high resolution logs than is done today.

The work presented here focuses on studying and implementing automatic methods to interpret fractures on image logs.

Borehole images are acquired by lowering a tool in the well and measuring a physical quantity (resistivity, travel time, wave amplitude) when pulling it up. This gives an image of the wellbore which contains a lot of information about fractures, sedimentary structures, layer properties, etc. Although it does not give as much information as cores, image logs can be a good substitute when some cores are missing or when the information provided by the image logs is sufficient. These images can even have information that the core has lost since the logs are acquired at reservoir conditions.

It has been decided to focus on fractures for the reason that they are of major importance to understand the behavior of reservoirs from a geomechanical and from a production point of view. Moreover, image logs are the only method giving direct measurements of fracture properties at reservoir conditions. Other indirect methods exist which involve inversion of seismic and production data or the use of outcrop analogs.

Still, image logs do not have only advantages. One of their drawbacks is their processing which might prove to be very time consuming if done manually, hence our purpose to make it automatic. Even if the ultimate goal would be to have a software interpreting as a geologist, for the moment someone needs to check that the proposed interpretation is coherent. To sum up, one can think of this work as the development of a tool in Matlab which makes the geologist's work faster and easier when it comes to fracture interpretation on image logs.

I) Image logs

Historically, the ancestors of image logs are dip meters which have been used in the industry since the 1930's. They were introduced to evaluate the dip magnitude and direction of rock strata. They were made of 4 or 8 electrodes measuring resistivity in equally distributed directions along the wellbore circumference. Hence, they could only give an idea of what the wellbore looked like. New imaging tools have up to 192 traces and provide a high resolution image, however one should not forget that dip meters were precursors and because of this many of the processing steps were initially designed to deal with them.

In the following, all the image logs that are presented have been oriented to give a proper representation of the wellbore so you cannot consider them as being raw data. Other corrections like tool velocity correction for example have been applied depending on the tool used for the acquisition.

1) Concept

Image logs are high resolution images of the borehole wall. They contain a lot of information about sedimentary structures, layer properties, and also fractures. There are usually two types of image logs: micro resistivity and acoustic images. Micro resistivity images are obtained thanks to many electrodes measuring how much current flow into the formation while acoustic images are obtained by measuring the travel time and amplitude of an ultrasonic wave propagating in the borehole. Even though micro resistivity and acoustic images do not measure the same physical property they are often very similar so it is interesting to compare them. Figure 1 shows an example of FMI (Full-bore Formation Microloger) and UBI (Ultrasonic Borehole Imager) images put side by side. The two images look very alike with the UBI image showing maybe more clearly the fractures. Even if electric and acoustic images are very similar, they have differences. One of them being that acoustic images can be acquired in any type of muds whereas resistivity logs need to be run in water based muds.

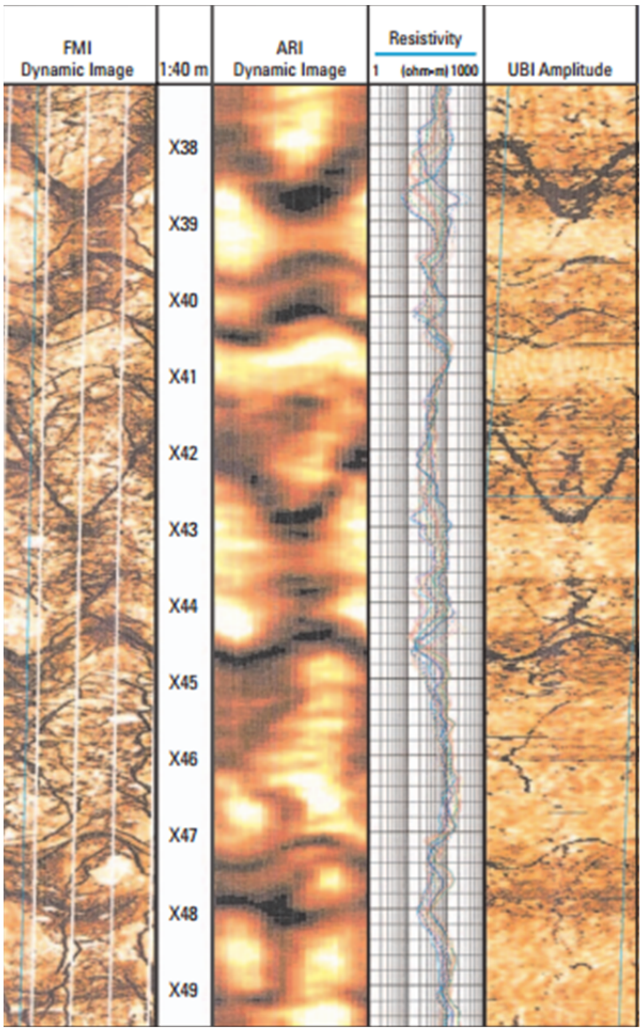


Figure 1 Comparison between acoustic (UBI) and electric (FMI) images. The fractures are more clearly seen on the UBI image because the background of the image is more homogeneous. [8]

In this study both micro resistivity and acoustic images have been considered. Thanks to the interest of Statoil for this project acoustic data acquired with a UBI was available. The micro resistivity images are a courtesy of Geostress, acquired by their tool: the HTPF (Hydraulic Testing on Pre-existing Fractures). Fortunately the two sets of data cover the whole well so the images present a complete coverage and the processing is made easier thanks to it.

A UBI is made of a rotating transducer (Figure 2) sending an ultrasonic wave and recording the reflected wave from the borehole wall for different channels. The UBI data presented in the following sections has been acquired with 180 channels meaning that the image is made of 180 traces. The amplitude of the wave gives some information about the texture of the surface and the transit time shows changes in the borehole radius. Although the UBI may not always reflect lithology changes accurately it is very effective to detect fractures. Figure 3 shows the UBI measurement specifications.

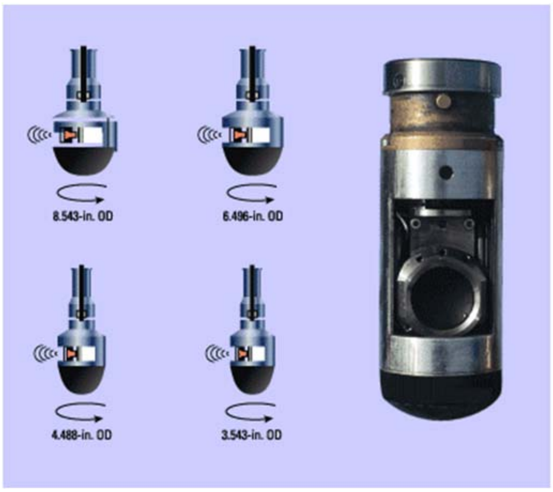
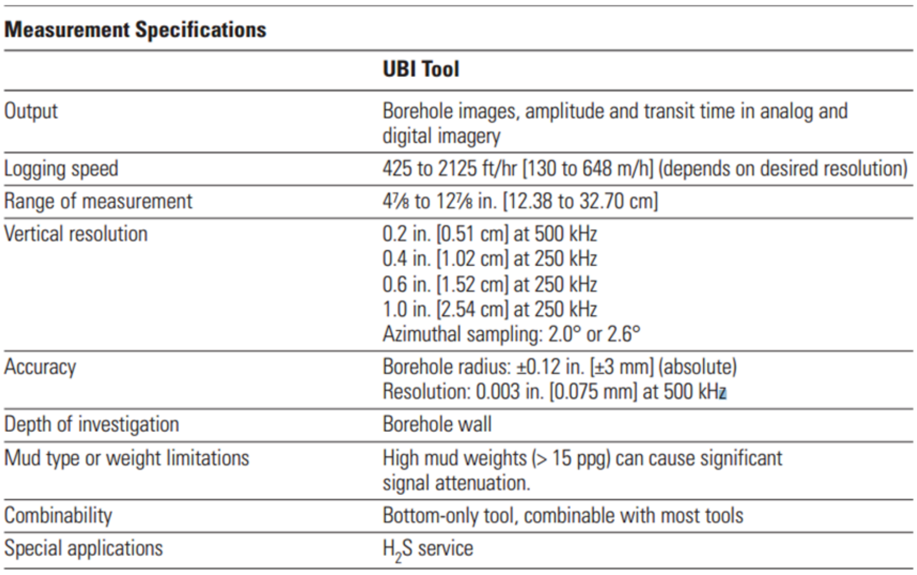


Figure 2 Transducer used in the UBI. The transducer can turn at different frequencies depending on the resolution of the image (I)



Measurement Specifications

	UBI Tool
Output	Borehole images, amplitude and transit time in analog and digital imagery
Logging speed	425 to 2125 ft/hr [130 to 648 m/h] (depends on desired resolution)
Range of measurement	4⅞ to 12⅞ in. [12.38 to 32.70 cm]
Vertical resolution	0.2 in. [0.51 cm] at 500 kHz 0.4 in. [1.02 cm] at 250 kHz 0.6 in. [1.52 cm] at 250 kHz 1.0 in. [2.54 cm] at 250 kHz Azimuthal sampling: 2.0° or 2.6°
Accuracy	Borehole radius: ±0.12 in. [±3 mm] (absolute) Resolution: 0.003 in. [0.075 mm] at 500 kHz
Depth of investigation	Borehole wall
Mud type or weight limitations	High mud weights (> 15 ppg) can cause significant signal attenuation.
Combinability	Bottom-only tool, combinable with most tools
Special applications	H_2S service

Figure 3 Measurement specification of a UBI tool (I)

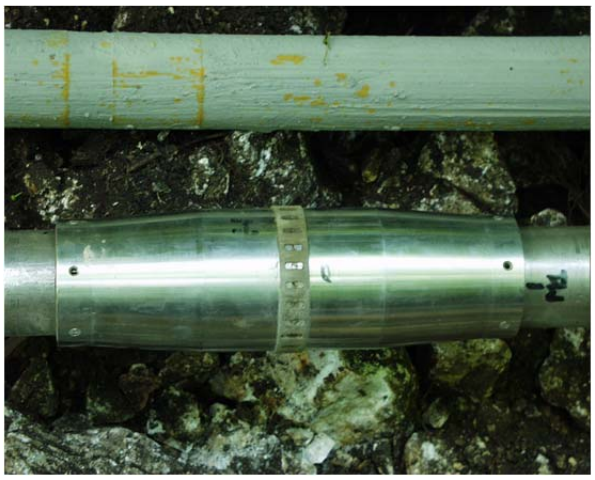


Figure 4 View of the imaging part of the HTPF. It is made of 24 equally spaced electrodes

The HTPF is used to calculate in-situ stress by reactivating pre-existing fractures [2]. Because the dip direction and dip of pre-existing fractures must be known to apply the method an imaging tool has been added to measure these. The imaging part of the HTPF is made of 24 equally spaced electrodes covering the whole borehole circumference. The electric current received by each of the electrodes is proportional to the conductance of the part of the borehole wall it is facing. Some focalization electrodes have been added to ensure that the direction of the current is normal to the wall.

2) Fractures on image logs

Fractures are a key parameter to understand the behavior of the underground from a geomechanical or hydrodynamical point of view. They can help you understand the stress situation, they can transform tight formations into producible ones, they can seal some parts of the reservoir, etc. Still, the main issue with them is that they are difficult to characterize. Many methods have been developed to characterize the fracture sets involving seismics, AVO analysis or reservoir engineering but image logs are yet the only way that allows a direct measure of fractures. Image logs are even more effective than cores for fracture analysis because there can be some drilling induced factures in the cores.

If one assumes fractures to be planar features and the wellbore to be cylindrical then the unwrapping of the cylindrical image will show fracture crossing the wellbore as sinusoids (Figure 5). The dip direction of a fracture can be directly read from the image by looking at the x axis of the minimum while the dip angle can be simply derived from:

$$\Theta = tan^{-1}\frac{A}{2R} \qquad (1)$$

Where Θ is the dip angle, R is the borehole radius, and A is defined in the figure below.

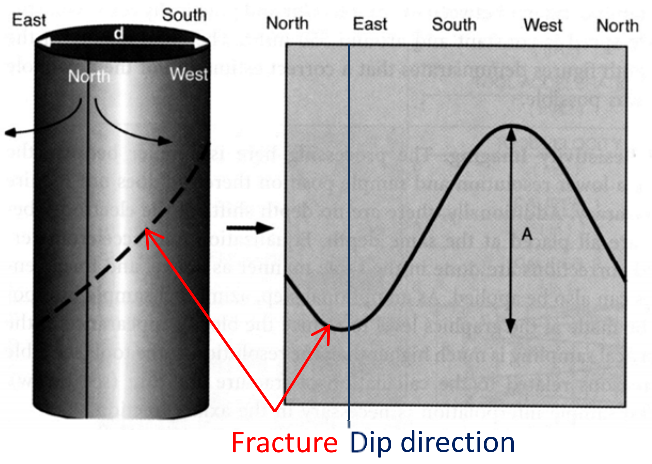


Figure 5 Sketch showing that a fracture intersecting a well shows on an unwrapped image as a sinusoid. [8]

However if the well is not a straight cylinder anymore or if the fracture is not planar (Figure 6) it is more difficult to identify fractures on the images because they are not showing as sinusoids anymore (Figure 7).

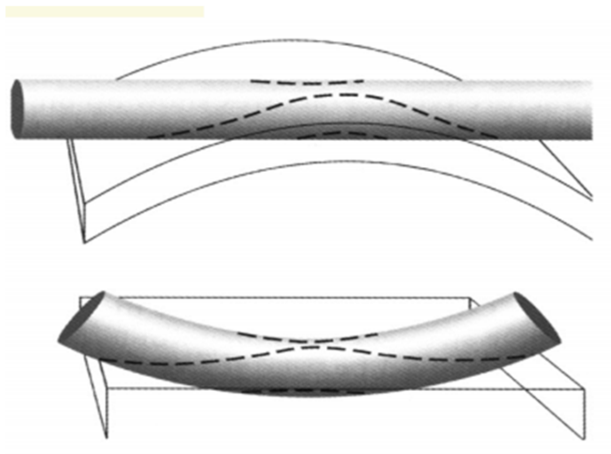


Figure 6 Schematic depicting an apparent overturned fold in a horizontal well and in a slightly undulating well with a horizontal fracture

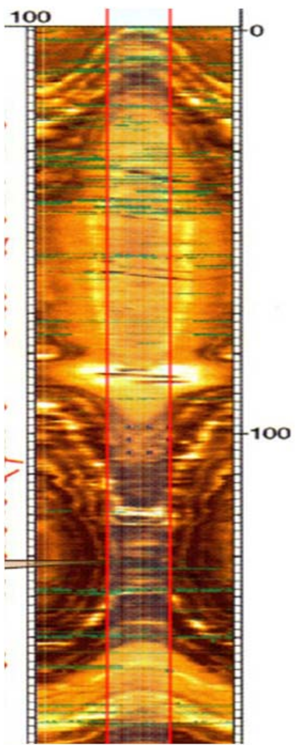


Figure 7 Example of an image acquired in one of the situations presented in Figure 6 [8]

This shows that sometimes fractures do not show as sinusoids on the image logs. Nevertheless, the following work is focusing on finding sinusoids in images so one should keep in mind that not all of the fractures might be detected by the methods presented here.

Now that the issue of interpreting fractures in image logs has been narrowed to the search of sinusoids in images, it is possible to move on to the actual processing of images. This processing is made of two major steps: edge mapping and reconstruction of sinusoids. Different methods have been considered and are thus being presented hereafter. Different synthetic cases are also shown as well as the results given by the latest version of the code.

II) Pre-Processing & Edge Mapping

In the following the images shown are resistivity images acquired with Geostress' HTPF tool. The images were acquired thanks to 24 equally spaced electrodes in a vertical well which was crossing some fractures. To compare every step of the workflow the same data is presented at different steps of the workflow.

1) Pre-processing

Before the fracture interpretation workflow can start, images need to be pre-processed in order to remove some undesired effects related to the acquisition or to the tool. Unfortunately not all of the noise has been corrected for and some issues remain when dealing with images containing breakouts or washouts.

The first step is to perform an interpolation of the data. When images are acquired the spacing between two measurements is often different because the tool can have a stick/slip behavior which is often due to varying well diameter. To correct for this, an interpolation is performed on every trace using a simple linear approximation. This step is compulsory if one wants to perform a Fourier transform so it is usually the first one in the processing workflow. In order to keep a similar image one usually chooses the theoretical sampling interval used during acquisition; hence it corrects the image without changing its resolution.

Figure 8 shows the original image used throughout the report to illustrate the effects of the different processing steps. Figure 9 shows how the image looks like once interpolated at regularly spaced depths. One can see that there is no major difference between the two images because the original data had already been interpolated but not at equally spaced depths.

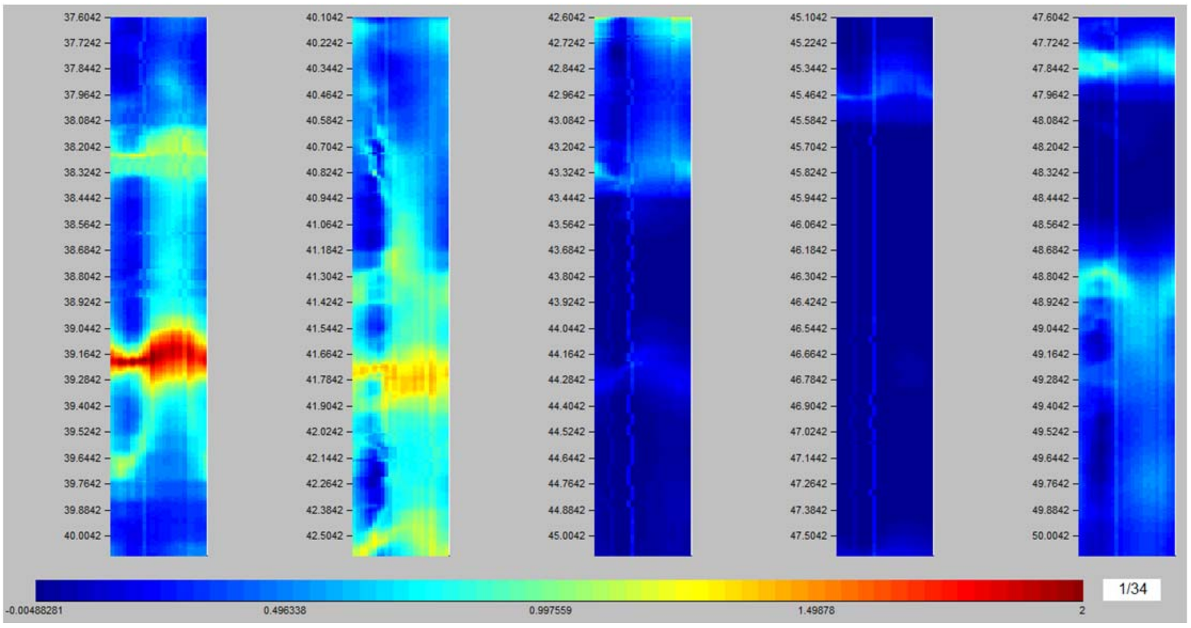


Figure 8 Original image of the HTPF data considered throughout this report. Some fractures are clearly visible for example there is one at 38.2042 m, 39.1642 m …

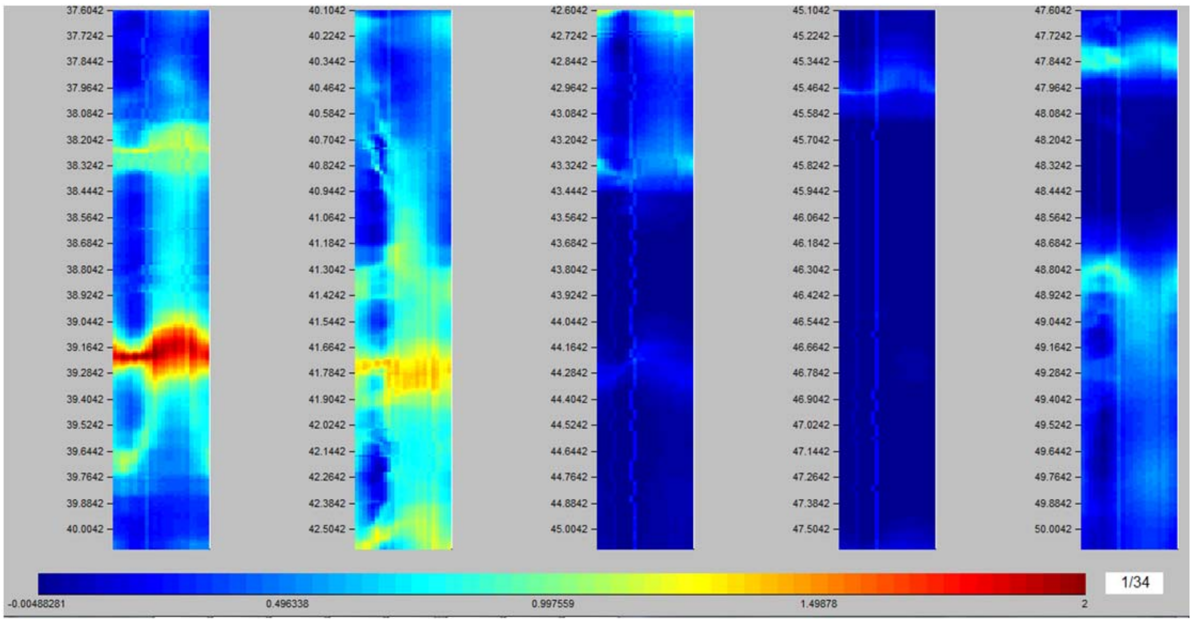


Figure 9 Figure 8 once it has been interpolated at regular depths. The depth interval here is equal to 1 cm.

The second pre-processing step is to filter the images horizontally. As seen from the two previous figures there can be some miscalibrated or broken electrodes that do not read the same values as their neighbors. Horizontal filtering allows to minimize the impact of the broken electrodes and to make features like fractures more homogeneous. This is very important when it comes to fracture interpretation because one looks for continuous events and if there is some heterogeneity the edge mapping and the interpretation of sinusoids can go wrong. This is especially true for UBI images which have 180 electrodes but it can actually be neglected in our reference case because it has only 24 electrodes. A horizontal filtering has still been applied using a 3*1 window (Figure 10).

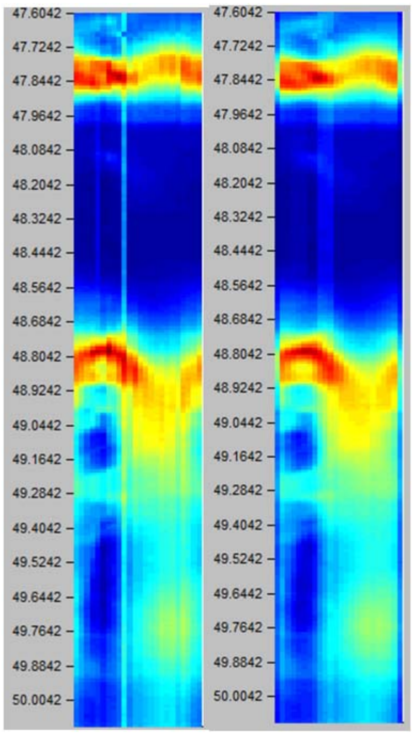


Figure 10 Reference image (left) and the same image when it has been filtered horizontally with a 3*1 window (right).

Now that some of the undesired signal has been removed one can move on to the edge mapping step. Edge mapping is the process that finds changes in images and creates new images containing only the contour of features. It simplifies images by only considering the features in them hence it is the first step in every feature extraction algorithm. This step is critical because a lot of information is lost so one has to be careful when doing it. If information is lost at this stage it won't be possible to retrieve it at a later one. In the following, two different edge mapping methods will be presented: one based on gradient calculation and a second one using Fourier transform.

2) Gradient based edge mapping

There are many different gradient based edge mapping methods but they all have the same skeleton:

- compute the gradient of the image

- find and keep the local maxima and minima of the gradients bigger than a pre-defined threshold

The difference between all gradient based methods is the way the gradients are approximated. There are many different methods to approximate the gradients like: the Sobel, Prewitt, or Canny approximation. It is also possible to calculate a simplistic gradient by subtracting a line to the following one.

If one assumes a fracture to look like a roof edge (Figure 11), then the gradient based edge mapping will draw the contours of the fracture where the gradient is either maximum or minimum. One of the drawbacks of this method is that it will also find many other contours that do not characterize fractures; basically it will find any events that are sharp enough.

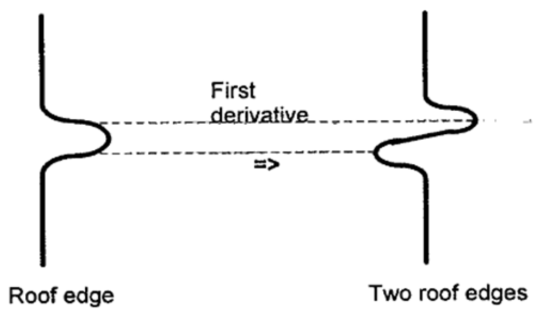


Figure 11 Roof edge representation of a fracture in a single trace [6]

To calculate the gradient of the image two methods have been tested. The first one is very simplistic because it calculates the difference line by line of an image. The results of these simplistic gradient calculations are presented in Figure 12. The second one is using the Canny filter to create an edge map (Figure 13). Canny is applying a 2D filter to process the whole image and is not identifying edges trace by trace; luckily it was already coded in Matlab (2) so it was quite easy to apply. Unfortunately the results given by the two gradient based edge mappings were not satisfying enough so it has been decided to focus more on the one using the Fourier transform.

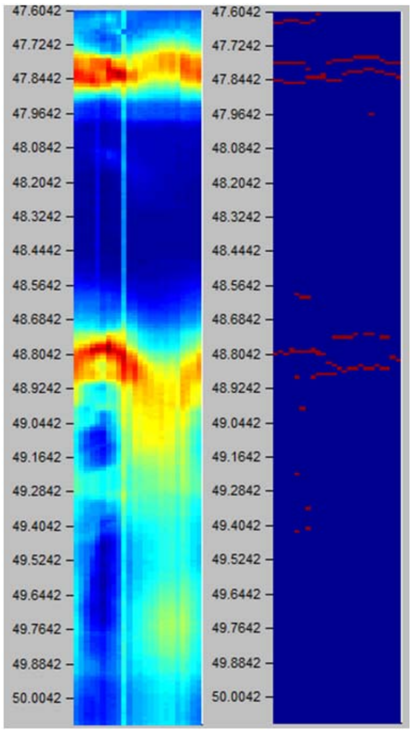


Figure 12 Gradient based edge map (right) of the reference image (left).Here the gradient was calculated by subtracting two consecutive lines.

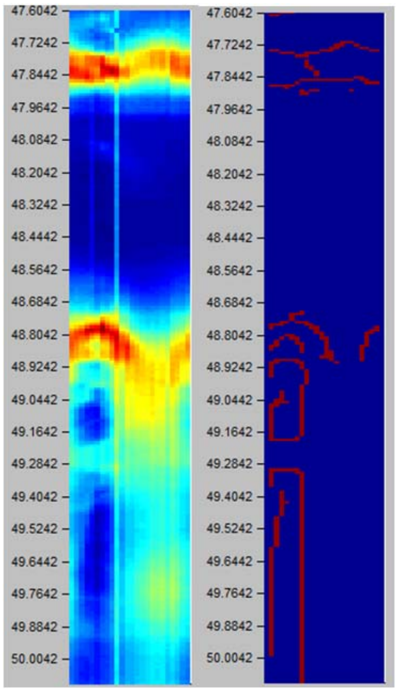


Figure 13 Gradient based edge map (right) of the reference image (left).Here the edge map was calculated using a Canny filter.

3) Fourier based edge mapping

The second edge mapping considered is based on the Fourier transform [6]. The idea is to increase the signal from the fractures while reducing the one from the lithology. To accomplish this, the images are first Fourier transformed, then the data is filtered in the frequency domain and finally a back transformation is applied.

Fourier allows looking at the images from a different perspective, instead of looking at it in the space domain one can see the spatial frequency content of the data. Thanks to this, one can choose the spatial frequency content one wants to keep in every trace. In the case of fracture detection in images, it is well known that some of the signal is coming from the lithology. This latter has often a larger thickness than fracture aperture. If a pre-defined band pass filter is applied, one should be able to keep the sine and cosine functions constructing the fractures and remove the one involved in the representation of thick events like lithology.

The Fast Fourier transform was used to calculate the Fourier coefficients f_j of every trace using the following equation:

$$f_j = \sum_{k=0}^{n-1} x_k e^{-\frac{2\pi i}{n} jk} \tag{2}$$

Where n is the number of data points, $x_1, x_2, \dots, x_{n-1}$ are the data and j=1,…,n-1.

The coefficients are then usually plotted as a function of spatial frequency to see how the spectrum of the signal looks like. It is then possible to design a filter that will only keep the frequencies of interest.

To get back to the space domain one applies an inverse Fast Fourier transform:

$$x_k = \sum_{j=0}^{n-1} f_j e^{\frac{2\pi i}{n} jk} \tag{3}$$

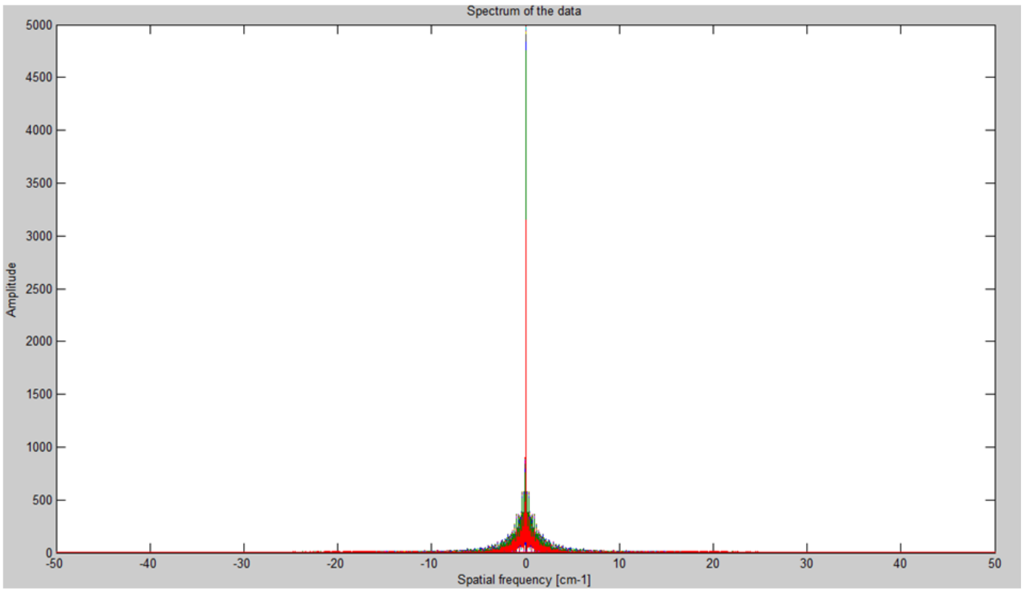


Figure 14 Spectrum of the 24 traces for the complete image. The spectrum shows very high coefficients in the low spatial frequency part meaning that the dominant part of the signal is coming from constants or from signals with a big spatial period.

Figure 14 shows the spectrum of the complete image. It shows that the traces are mostly dominated by low spatial frequency signals. If one keeps only the spatial frequencies between 0.8 and 8 cm-1 and make an inverse Fourier transform one gets Figure 15. It is possible to see from this figure that fractures are better defined after filtering because the background has been made more homogeneous. Also the impact of the miscalibrated electrodes has been removed. The band pass filter used here keeps spatial frequencies between 0.8 and 8 cm-1 or equivalently spatial periods between 1.25 cm and 0.125 cm.

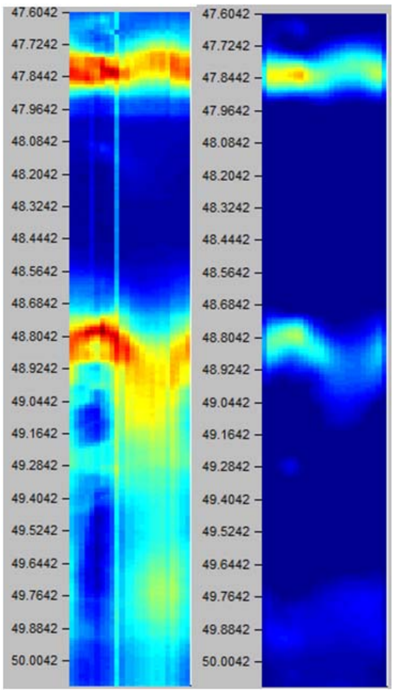


Figure 15 Reference image (left) and the same image when it has been filtered with a band pass filter (0.8-8 cm-1) (right).

Now that the image has been filtered so that the signal from fractures has been enhanced, a simple edge method has been applied. This edge mapping is based on finding all the local maxima in the filtered images that are bigger than a pre-defined threshold. One advantage of this edge mapping method is that it will plot an edge in the middle of fractures (Figure 16) compared to the gradient based edge mapping that are plotting the contours of fractures.

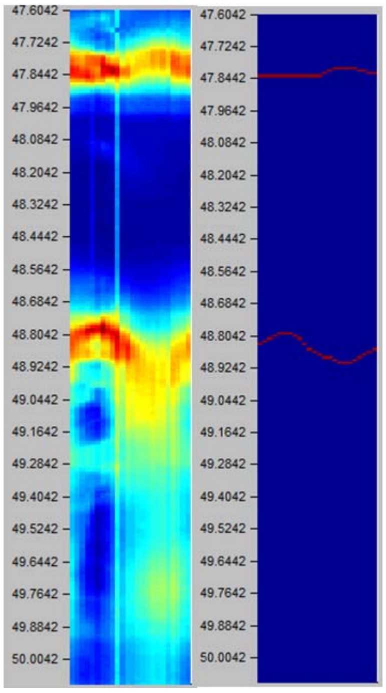


Figure 16 Reference image (left) and its edge map when using a Fourier based edge mapping method (right).

Even though this Fourier transform based edge mapping is powerful it cannot make the difference between fractures and layers with a thickness close to the aperture of fractures so one should keep in mind that all the detected events are not necessarily fractures.

Once this edge mapping has been applied to detect fractures with lower resistivity than their surroundings it needs to be applied again to detect cemented fractures with higher resistivity than their surroundings.

III) Sinusoid recognition

Now that an edge map has been created the sinusoid recognition may begin. Three different ways to reconstruct sinusoids have been implemented:

- Correlation calculation [4]
- Hough transform [3]
- Radon transform [1]

For each method a synthetic case will be presented to show how they work in theory.

1) Correlation calculation

The first method used to extract sinusoids is involving the calculation of correlations and is of the "edge-following" type. In this method one tries to regroup points belonging to the same sinusoid, once this is done one fits a sinusoids through the group of points to get the arguments of the sinusoid. This method does not need to be applied on an edge map and it could actually use directly the pre-processed images.

To calculate the correlation R between two vectors X and Y having the same size n, one computes:

$$R_{XY} = \frac{\sum_{i=1}^{n}(X_i - \bar{\bar{X}})(Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2 \sum_{i=1}^{n}(Y_i - \bar{Y})^2}} \tag{4}$$

Let's assume one found a fracture in a trace and one wants to regroup all the points belonging to this fracture. Once the aperture of the fracture has been determined, one can look at the next trace to see where the fracture is going by calculating the correlations in a window with a pre-defined size and by choosing the position that maximizes the correlation (Figure 17).

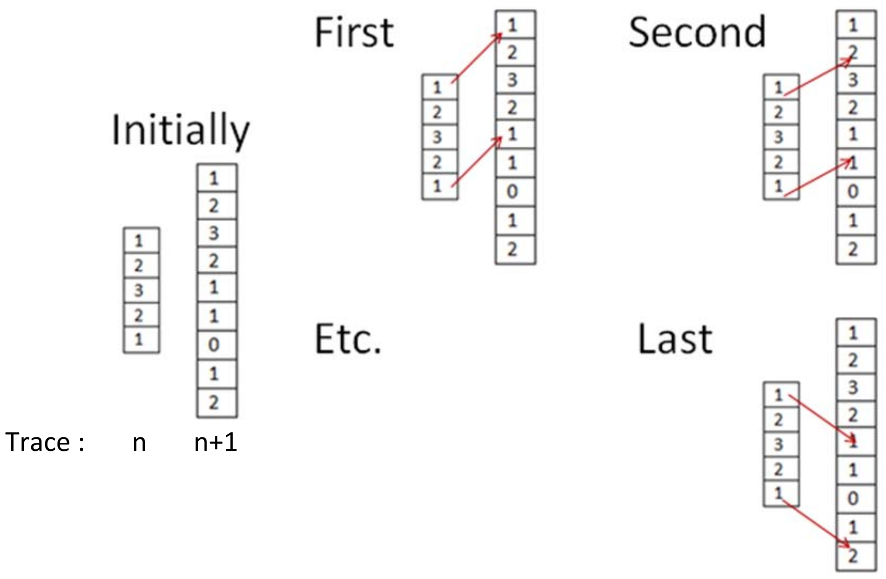


Figure 17 Schematic representation of how the algorithm calculates the correlation. If the maximum correlation is higher than a pre-defined threshold, the points are regrouped. In this example the correlation is maximum in the first case.

The synthetic case used for this method is presented in Figure 18: it has one single fracture and two intersecting fractures with different apertures.

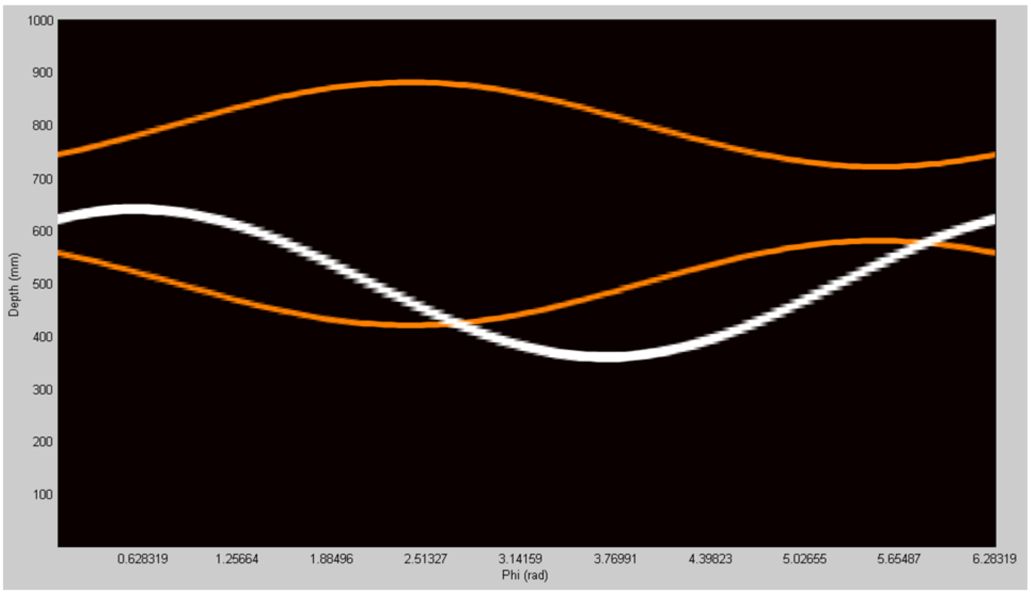


Figure 18 Synthetic case used to test the sinusoid reconstruction using correlations

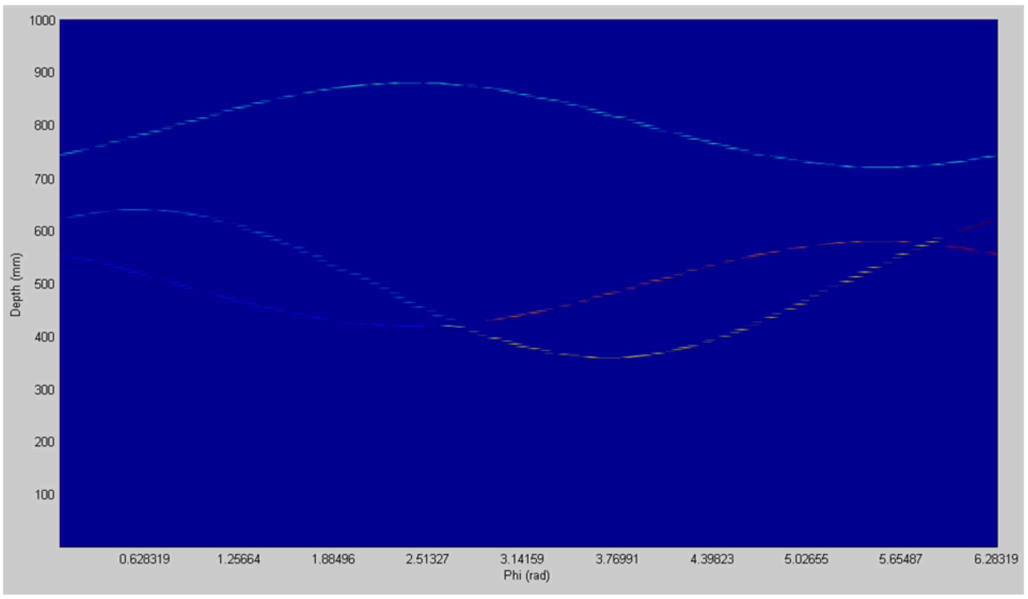


Figure 19 Image showing how the points have been regrouped using correlations. Every color corresponds to one group of points.

Figure 19 shows the results of the regrouping using correlations. When there is a single fracture, the algorithm is capable of regrouping all the points belonging to this fracture but there are some issues when two fractures are crossing: points are regrouped as long as one fracture is far enough from the other. To regroup points further one can either fit a sinusoid through every group of points and regroup the points if the arguments of the sinusoids are close enough or extrapolate every group of points using the derivative. This latter is actually leading to better results when applied on real data. Figure 20 presents how the derivative helps to regroup points together.

In the end this method proves to be quite effective on synthetic data (Table 1) and actually the final version of the interpretation program has been derived from it (see IV). The reason why it is effective is that it can find fractures with not perfect sinusoidal shapes thanks to the way the regrouping is done. Yet, there are also some drawbacks. If one applied directly this correlation method to real data one would find that the computation time is very important, there may also be some issues with the fitting and finally this method is not especially resistant to noise. The pre-processing steps are actually critical for this method because one is looking for consistent continuous events in the image.

Precision Amplitude	Precision Phase	Precision Depth	Computing time
<1%	<1%	<1%	8.87 s

Table 1 Performances of the correlation method on a perfect synthetic case

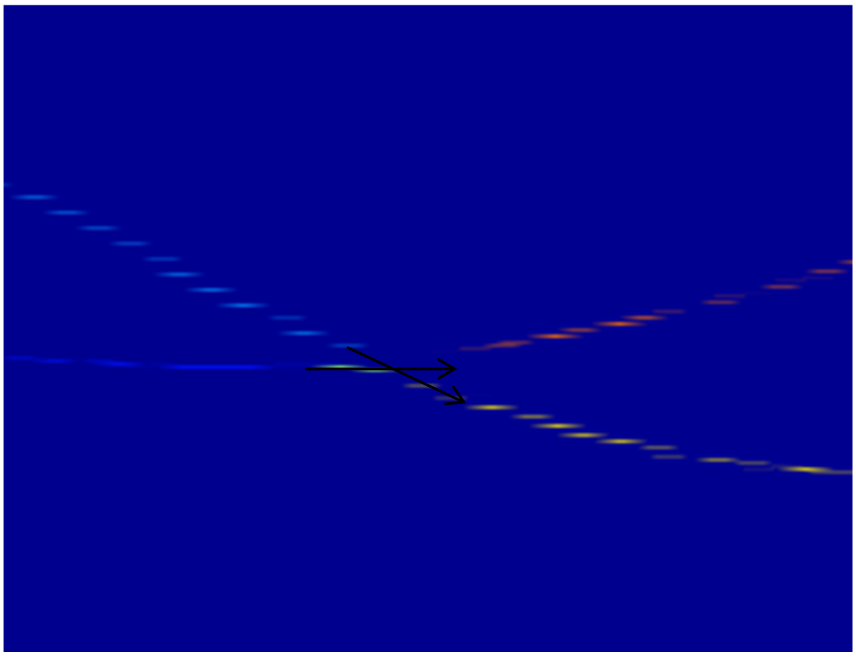


Figure 20 Zoom on the intersection of two fractures showing how a regrouping can be made by extrapolating along the tangent at the end of groups

2) Hough Transform

The Hough transform is a feature extraction technique that is very effective in noisy environments. The concept behind it is to create a parameter space in which a voting procedure is applied.

In the case of fracture interpretation, sinusoids have to be extracted from the edge maps. A general equation of a sinusoid is:

$$y_0 = A * sin(x_0 - Phi) + D \quad (5)$$

Where A is the amplitude, Phi is the phase, D is the depth, (x_0, y_0) are the coordinates of a point in the edge map belonging to the sinusoid.

Rewriting equation 5 leads to:

$$A = \frac{y_0 - D}{sin(x_0 - Phi)} \tag{6}$$

The depth can actually be determined by applying a Hough transform on successive sliding windows as it will be shown later. Getting rid of the depth D leads to:

$$A = \frac{y_0}{sin(x_0 - Phi)} \tag{7}$$

Equation 7 is actually the basis of the Hough transform because one can plot in the parameter space (Phi, A) all the possible 2π periodical sinusoids going through the edge points of interest (x_0, y_0). At the point where all the curves cross in the parameter space one can read the amplitude and phase of the sinusoid of interest.

Let’s assume that a perfect sinusoid (with or without gaps) is in the middle of an edge map. Then if one applies equation 7 to every point in the edge map, i.e. if one considers all the 2π periodic sinusoids going through all these points, one would get Figure 21. From the intersection point we read the amplitude and the phase of the sinusoid.

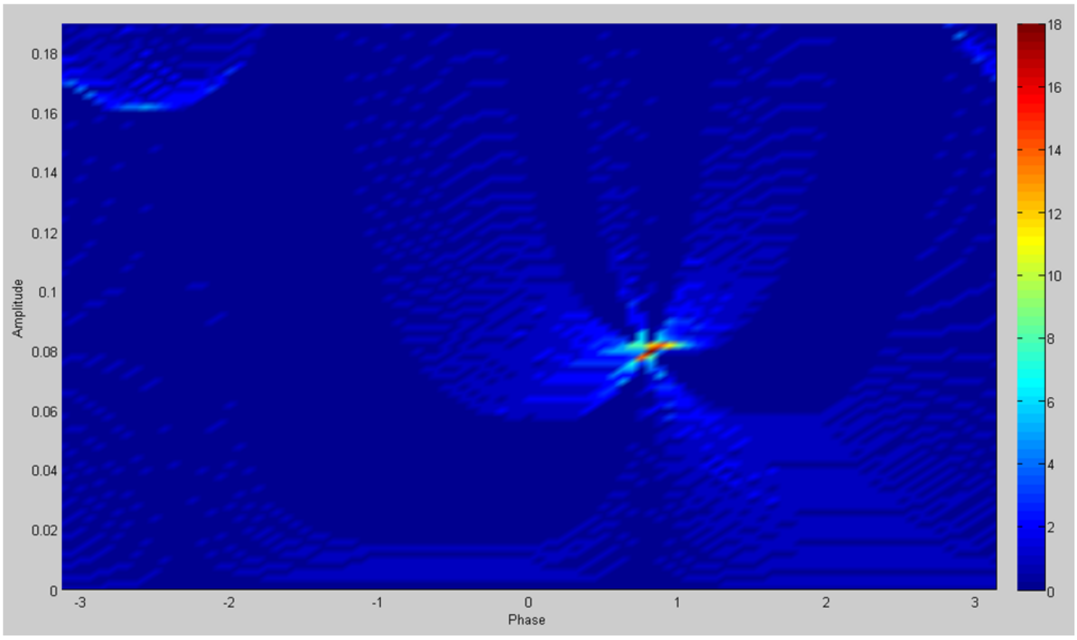


Figure 21 The parameter space when a single sinusoid is centered in an edge map without any other points.

To determine at what depth the sinusoid is located, we use the fact that if the sinusoid is not centered in the image the curves will not always cross at the same point (Figure 22) and even if they do the accumulation will be less than when the sinusoid is centered.

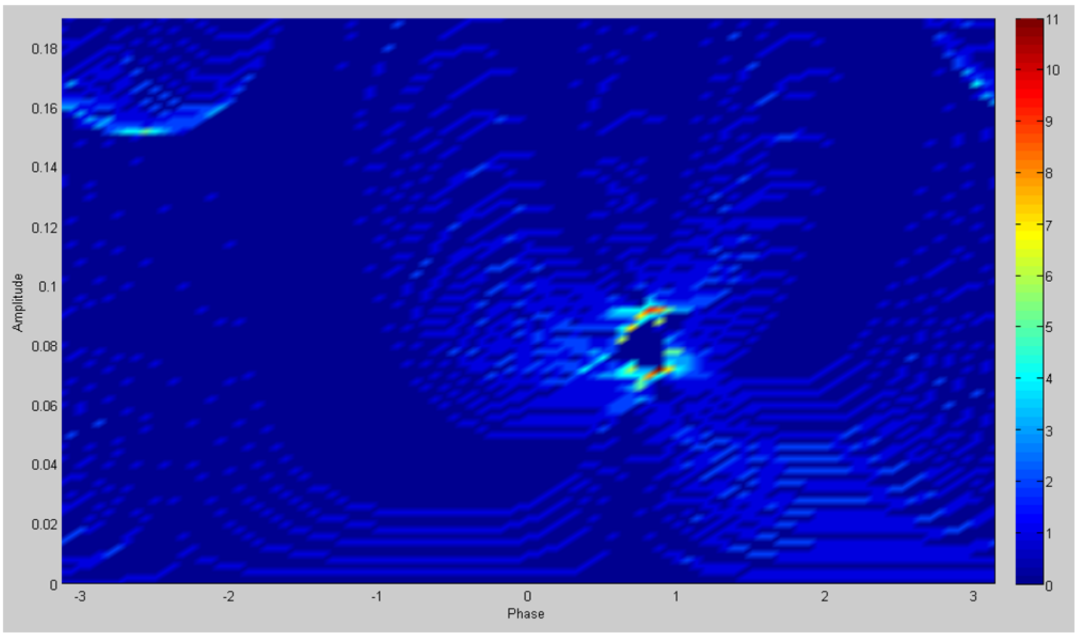


Figure 22 The parameter space when a single sinusoid is not centered in an image. Multiple local maxima appear with lower values than the one when the sinusoid is centered in the image.

In the end, to apply the Hough transform on the edge maps of image logs one should:

1. Determine the vertical size and sampling interval of the sliding window used to cover the whole image.
2. Apply the Hough transform on every window
3. Look for local maxima in the 3D parameter space
4. From the position of these local maxima one gets directly the amplitude, phase and depth of the fractures

The synthetic case used for this method is presented in Figure 23: it has one single fracture, two intersecting fractures and some random noise has also been added.

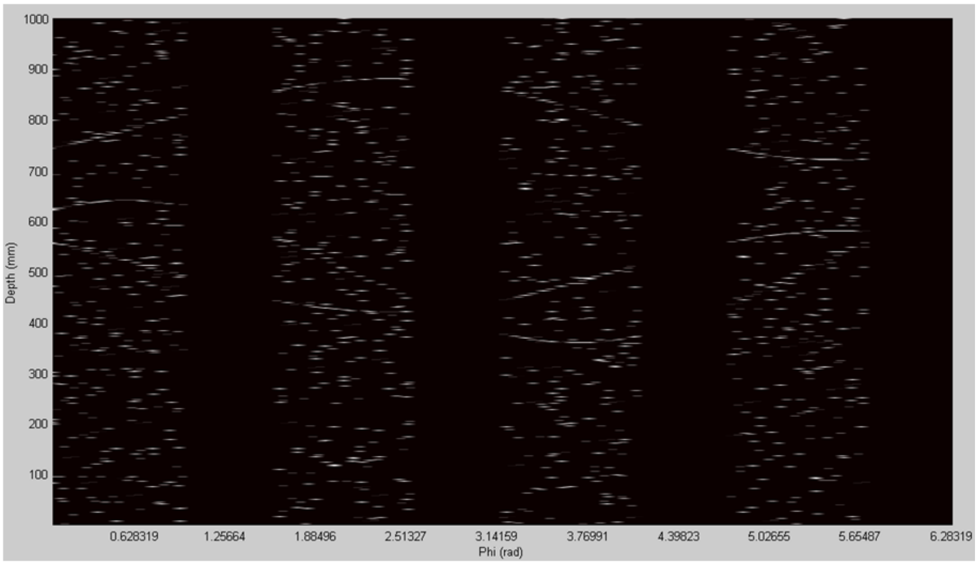


Figure 23 Synthetic case used to test the sinusoid reconstruction using the Hough transform

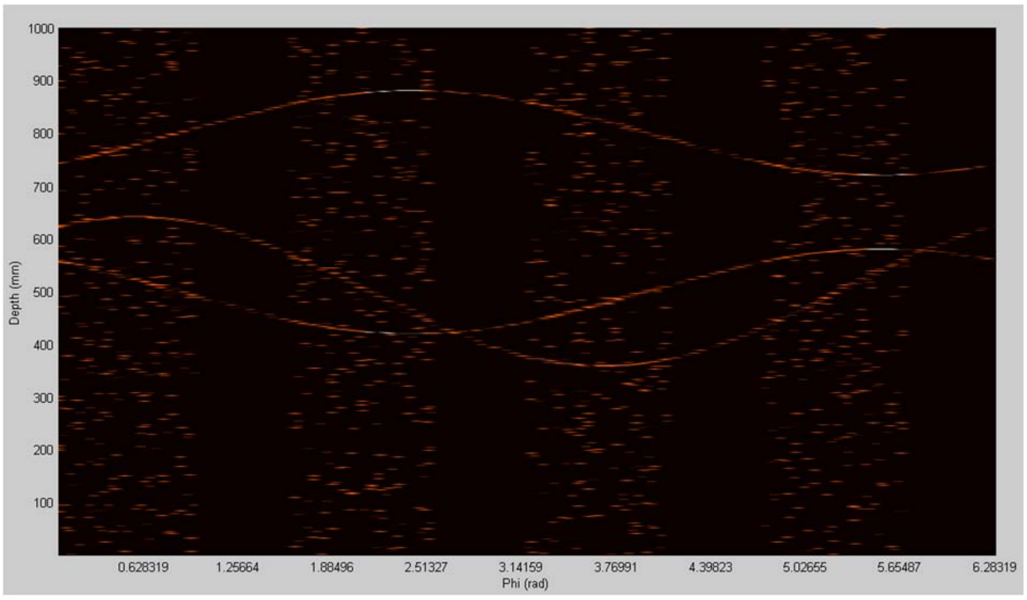


Figure 24 Results of the Hough sinusoid extraction algorithm. The method is very noise resistant.

The results of the Hough transform extraction algorithm look very good (Figure 24) because even though there is a lot of noise the algorithm still manages to extract the correct sinusoids thanks to the voting procedure. The main drawback associated with this method is the computation time when some random noise has been added (Table 2). The results obtained thanks to this method are highly dependent on the time one is ready to wait for.

	Precision Amplitude	Precision Phase	Precision Depth	Computing time
No noise	<1%	<1%	<1%	14.7 s
With noise	<1%	<1%	<1%	45.8 s

Table 2 Performances of the Hough transform method on a perfect synthetic case without and with random noise

When one tests this method on real data, one can be disappointed because the results are not so good due to fractures having not perfect sinusoidal shapes. I have tried to create a coarser parameter space to deal with this issue but even then the results are not as good as one would expect them to be. There are actually two other ways to increase the detection of fractures on real data using the Hough transform. The first one is to correct the image log for washouts and breakouts in order to be in the case where the wellbore is a perfect cylinder. The second way is to make the interpretation process semi-automatic in order to better constraint the solutions.

3) Radon transform

The idea behind the radon transform is to investigate several directions around an edge point in order to detect in what direction a feature is extending (Figure 25).

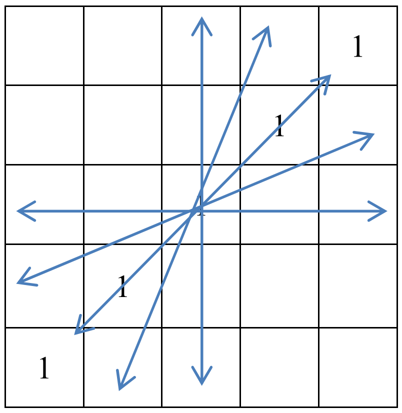


Figure 25 Schematic representation of the Radon transform when applied on an edge map. One calculates the sum along different direction to find the direction of the feature.

In theory this also works for intersecting features as shown in Figure 26 where both events at 0° and 45° are detected if the sampling angle is small enough.

				1
			1	
1	1	1	1	1
	1			
1				

Figure 26 Zoom of an edge map on the position where two fractures cross

In the end the Radon transform of an edge map leads to a 3D map where for every edge point the angles of detected lines are plotted. There are several methods to find back the arguments of the sinusoids:

1. Regroup points by following the derivative calculated with Radon
2. Identify the points with a horizontal derivative and try to regroup them together
3. Identify the points with the maximum Radon value and try to regroup them together

The first method has already been mentioned before and was actually used in one of the previous version of the interpretation program. The advantage of this method is that it can regroup points even if the fracture is not perfectly sinusoidal. On the other hand the method can be very time consuming depending on the number of directions investigated to do the Radon transform and depending on the number of fits it has to perform to calculate the arguments of the regrouped points.

The second method is simplistic because it looks for points where the derivative is zero i.e. it looks for maximums and minimums of sinusoids. After this, one maximum has to be linked to a minimum in order to get the arguments of the sinusoid. This is normally done by calculating the horizontal distance between the maximum and the minimum. If it is equal to half a period of the well and if the vertical distance is not too big then there are good chances that the two points belong to the same sinusoids. As “good chances” is difficult to translate in Matlab, what the code does is that it first links the points which go together with 100% confidence and then it takes care of the others recursively. The results of this method are actually highly dependent on the size of the window used to perform the Radon transform and they are very disappointing on real data. In real data there are often gaps in the images due to break outs and wash outs that condemn this method to work only on synthetic data.

This second method was actually tested on synthetic data. The synthetic image is presented in Figure 27, the Radon transform leads to keep only the points where the tangent is horizontal. The sinusoidal reconstruction is trivial as soon as maximums and minimums have been linked but unfortunately the results are not very convincing (Figure 28 and Table 3).

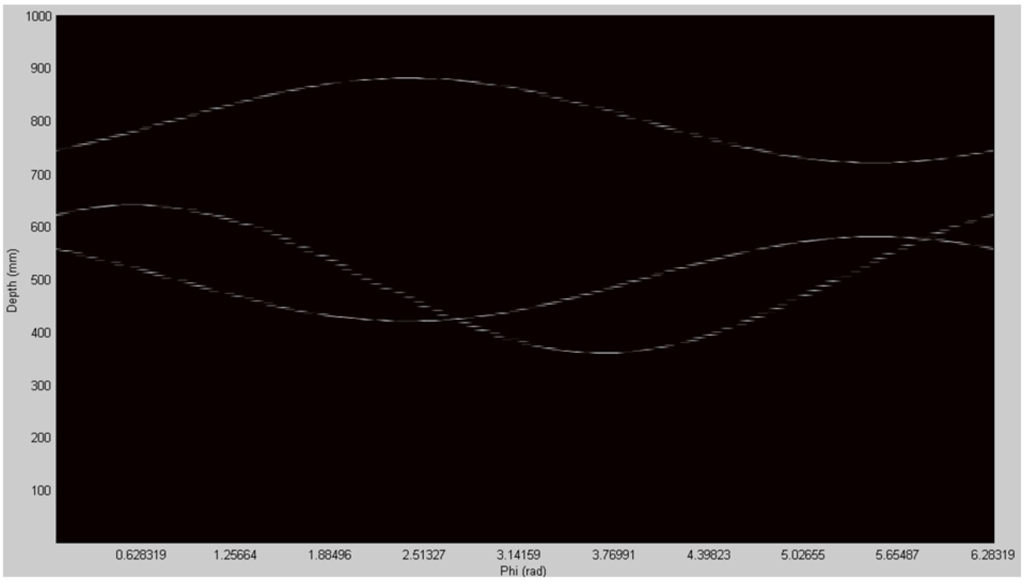


Figure 27 Synthetic case used to test the sinusoid reconstruction using the Radon transform

Precision Amplitude	Precision Phase	Precision Depth	Computing time
1.5%	18%	<1%	4.38 s

Table 3 Performances of the Radon transform on a perfect synthetic case when using the simplistic approach

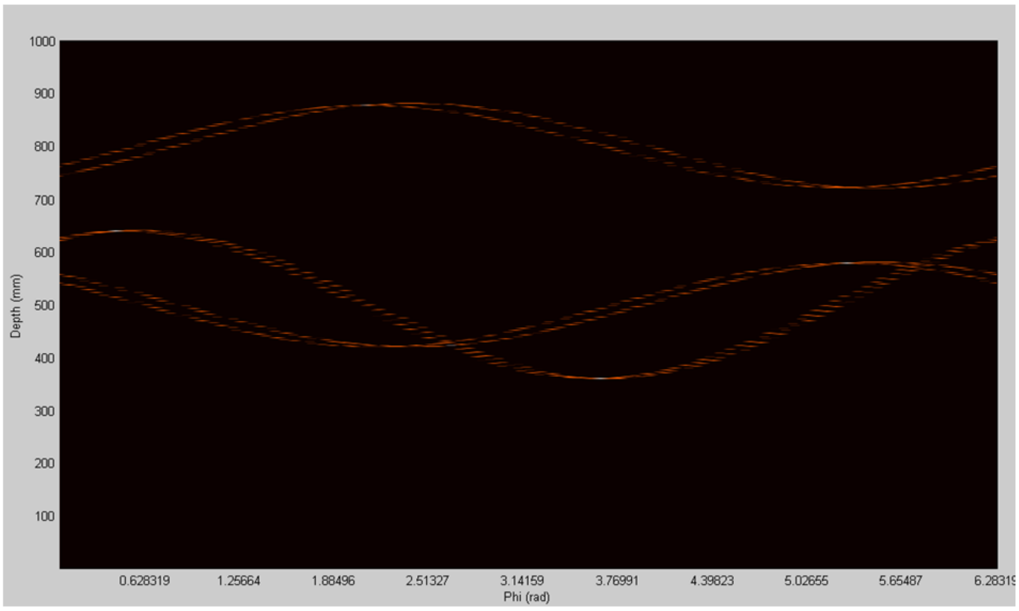


Figure 28 Results of the sinusoids extraction after applying a Radon transform and using the sipmlistic reconstruction method

Once the radon transform has been performed, the third method is using the points where the Radon transform is maximum to reconstruct the sinusoids. As before one has to find a way to regroup the points two by two. In the present case we know the position of two points and the derivatives in these points. Thanks to this it is actually possible to calculate the period of the sinusoid that would go through them:

$$T = \frac{2\pi(\varphi_1 - \varphi_2)}{\cos^{-1}\frac{y'_1}{y'_2}} \tag{8}$$

Where T is the period, φ_1 and φ_2 are the angles defining the position of the two points and y'_1 and y'_2 are the derivatives in those points.

Once the points have been regrouped two by two, the arguments of the sinusoids can be found from the position of the points and from their derivatives. Unfortunately there is a bigger issue that has to be accounted for before linking points two by two: one first needs to find those points.

The points used for the sinusoid reconstruction are usually chosen to be local maxima from the Radon transform. To make the Radon transform one calculates the sum along different direction and the sum is maximum in the places where the sinusoid is the most linear (Figure 29). The problem comes when one wants to find points of high amplitude sinusoids because as seen from the Figure 29 their Radon transform is much lower than the one of low amplitude sinusoids. Hence, it is not sure that the local maxima search will lead to pick the points of interest.

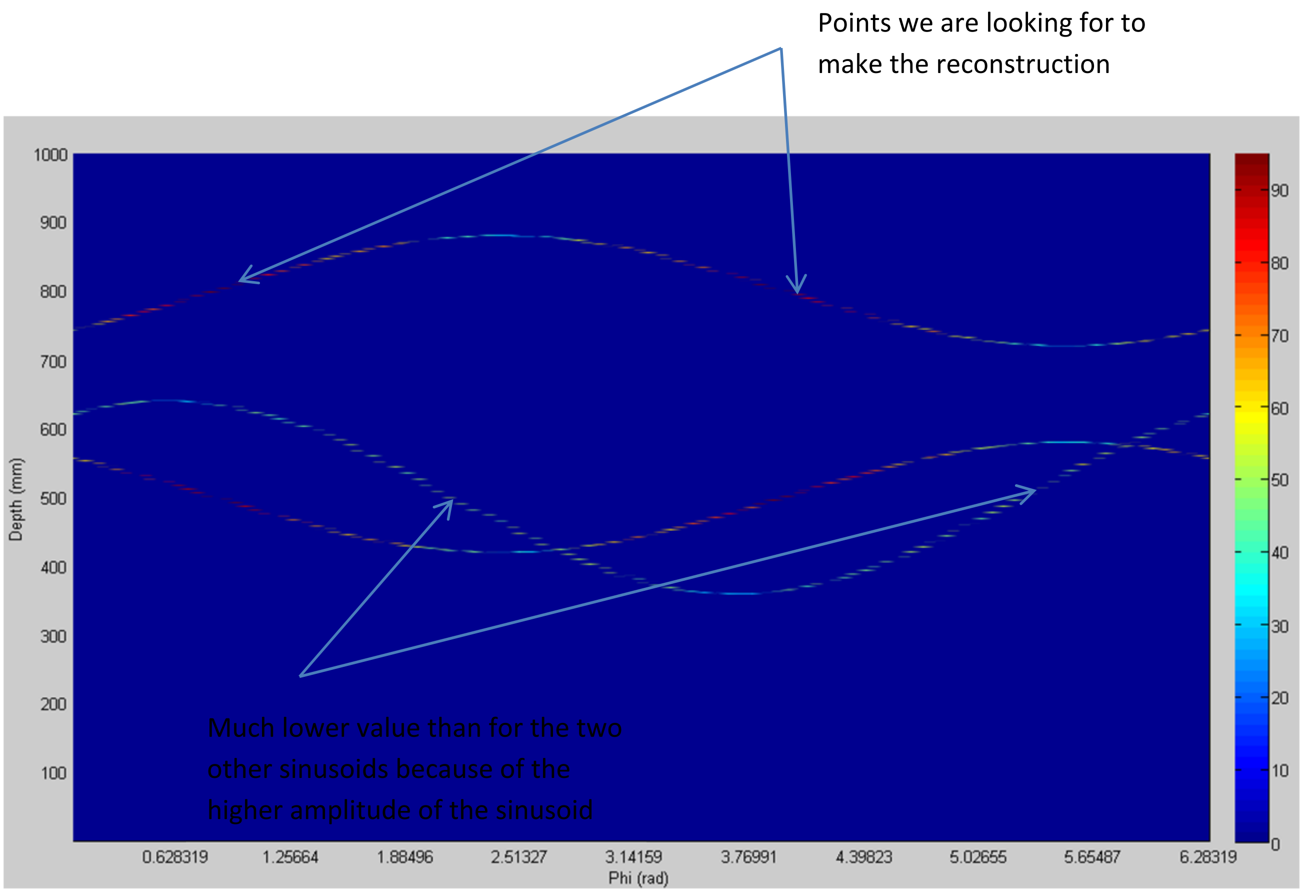


Figure 29 Results of the Radon transform on the synthetic case. The value plotted in each point corresponds to the maximum sum found when summing along different directions.

IV) Final interpretation program

After understanding and testing different methods to automatically make the interpretation of fractures, a program was made to answer the problem. It was designed with the purpose of being fast and able to interpret sinusoid like fractures. The program has been inspired from the correlation method with the purpose of being resistant to the fracture shapes; it is part of the edge-following extraction algorithms [5] [6].

1) Program workflow

The workflow of the program is the following:

1. Interpolation
2. Horizontal filtering
3. Band pass filter trace by trace
4. K-means algorithm
5. Regroup group of points together
6. Fit sinusoids through every group

This workflow seems very similar to the one previously presented to the exception of the K-means [8]. The use of this algorithm is what makes this fracture interpretation code original.

The K-means algorithm is replacing the last step of the original edge mapping process where a threshold had to be chosen. Thanks to this algorithm the values of the image are partitioned into groups without needing a threshold. The idea is to make belong every value in the image to the partition with the closest mean. In the following, the values have been partitioned in three. If one keeps only the upper values (the lithology effect has been removed so fractures should have dominant values) one gets Figure 30. One can clearly see how the partitioning has been done on this figure.

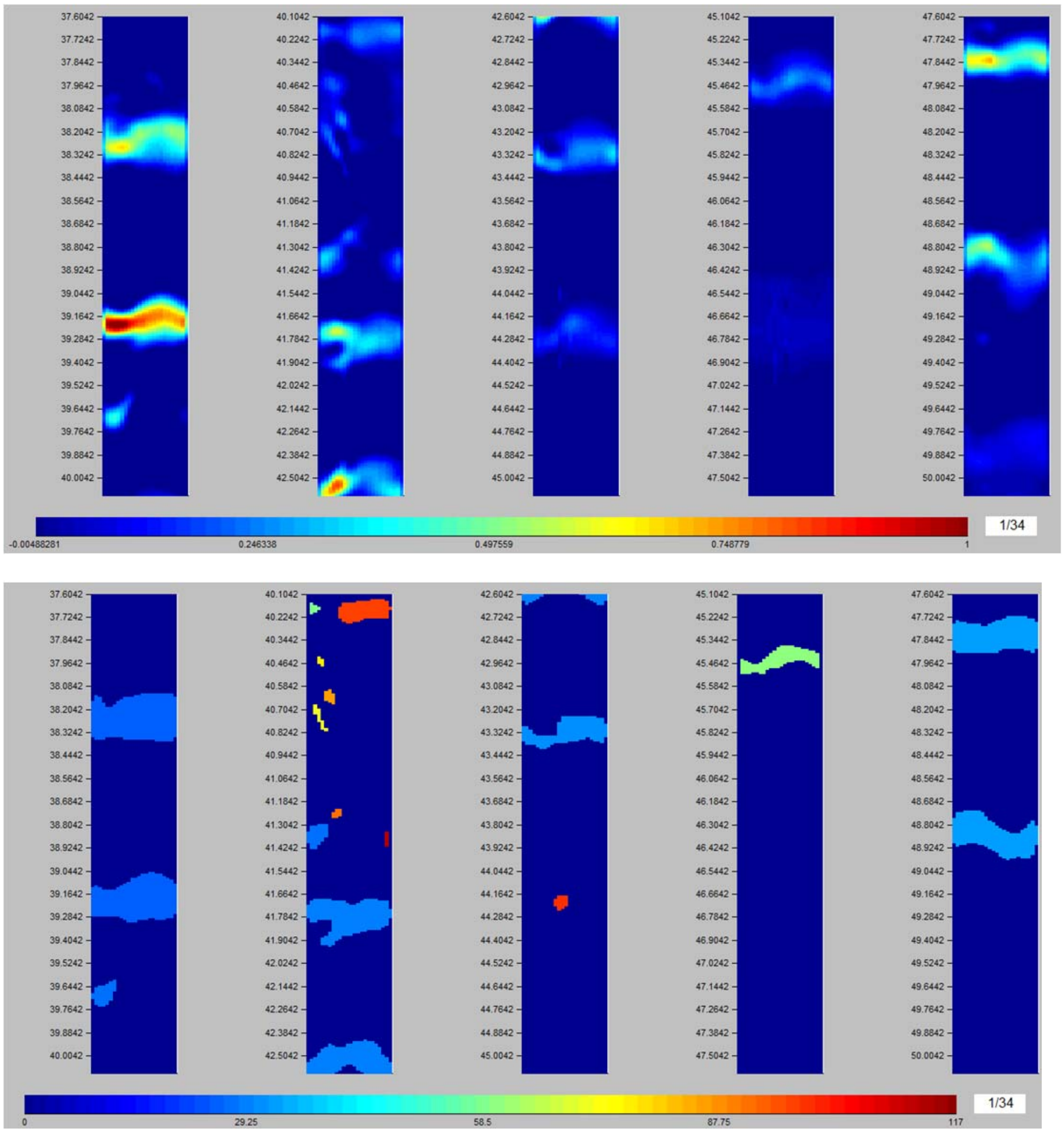


Figure 30 Top : Resistivity image after having been band passed filtered. Bottom: The same image after having made the partitioning using the K-means. The image is divided in zones.

This method identifies many groups of points in the image. One could decide to proceed like before and create directly an edge map but this would lead to the loss of a lot of information. As seen from Figure 30 many points have already been regrouped together so the program actually finds the edge points in every zone and keeps them regrouped by zone. After that, the program tries to regroup furthermore by using the derivative as shown in Figure 20. Finally it performs a fit on every group of points.

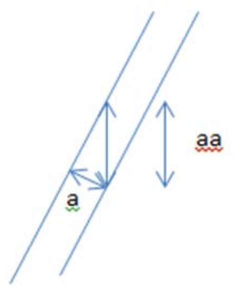
There is of course an issue when some fractures are intersecting. In that case the zone found with the K-means looks like a star with a varying amount of branches. The program is again using the derivatives to determine what branch of the star goes with which one. Afterwards, the processing is similar as before.

2) Performances

This interpretation program has been first tested on resistivity images and then on UBI images.

Figure 32 and Figure 33 show examples of the program's results when applied on the resistivity data from Geostress. On Figure 32 the interpretation has been done quite well since only two fractures have been missed, one at 39.644m and the other one at 44.284m.

Figure 33 shows some of the issues remaining and that should be improved. The first one deals with large aperture fracture. One can see that there is one at 15.8242m and it is not properly interpreted. The reason why this kind of fractures is not detected is because of the chosen band-pass filter. When the aperture of the fracture is big it is not sure whether it is a fracture or a thin-bed and this version of the program doesn't take the risk of interpreting large aperture fractures. The second issue relative to this program is the interpretation of high dip fractures. There is one at 23.6842m that is not properly interpreted. This is caused again by the chosen band-pass filter which is not taking account thick enough events. The problem is that even if a fracture has a low aperture, if its dip is important it will look as it has a big one (Figure 31). The fore last issue of this program is that it can sometimes interpret a fracture where there is none (at 17.25m). This is due to a bad combination of events which put together look like a sinusoid. Finally, the program has troubles identifying low contrast fractures. That could actually be improved by increasing the number of partitions in the K-means and by taking more zones. For example, if one makes a K-means with 4 partitions and one keeps the 2 higher partitions, more low contrast fractures would be interpreted. This could also mean that other features are detected which are not fractures.



Fracture

Figure 31 Representation of the aperture a of a fracture and its apparent aperture aa when looking at it trace by trace

In the end, for the complete data set from Geostress that is covering around 88m: 52 fractures were automatically detected out of 68, out of these 52 detected 9 were wrong. Most of the fractures that were not detected had low contrast and most of the 9 mistakes were due to large fracture aperture or high dip. The interpretation and display were carried out in 27 seconds.

The program was also tested on some UBI amplitude data and an example of the results is given in Figure 34. The program is processing 20m/40s and all the sinusoidal events were properly interpreted in Figure 34. The issue of knowing whether those events are fractures are not is another problem that could be solved by comparing with other logs. This set of data also showed the limits of the program when trying to process some areas with break-outs or wash-outs. In these cases the image should first be corrected for these effects prior to being processed.

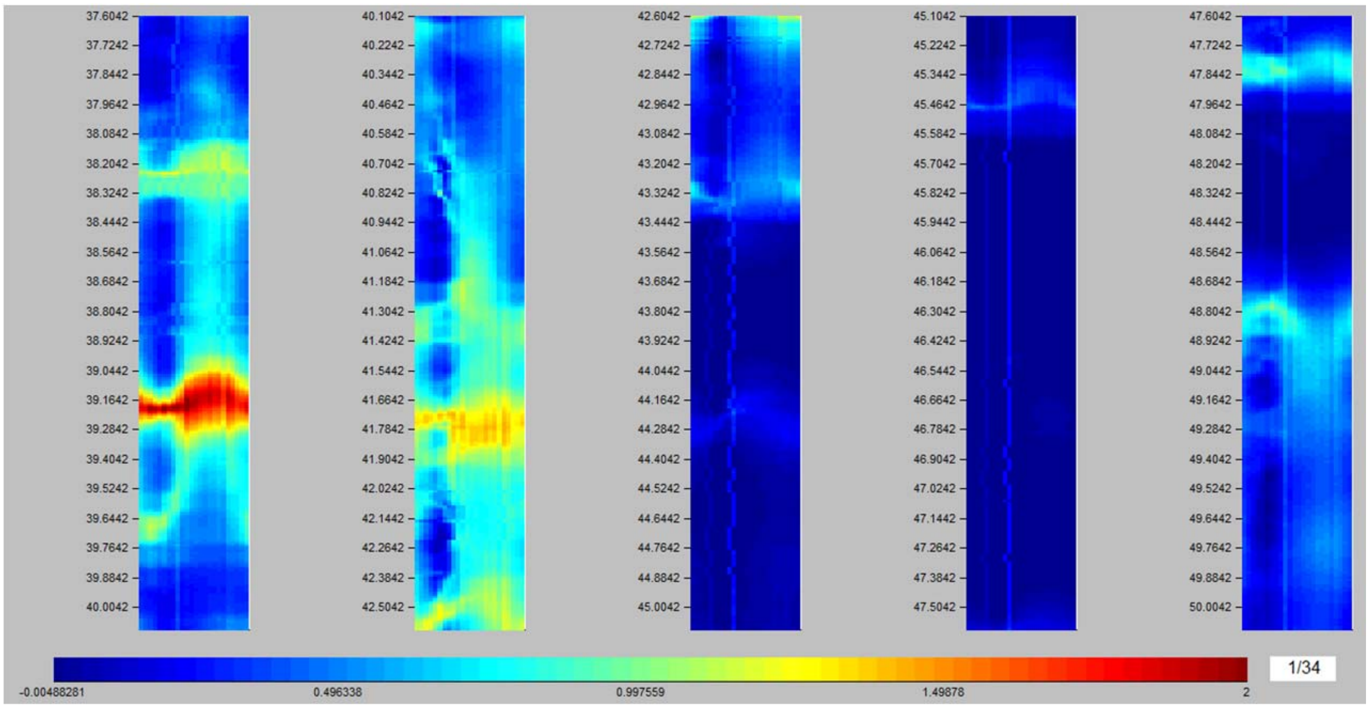
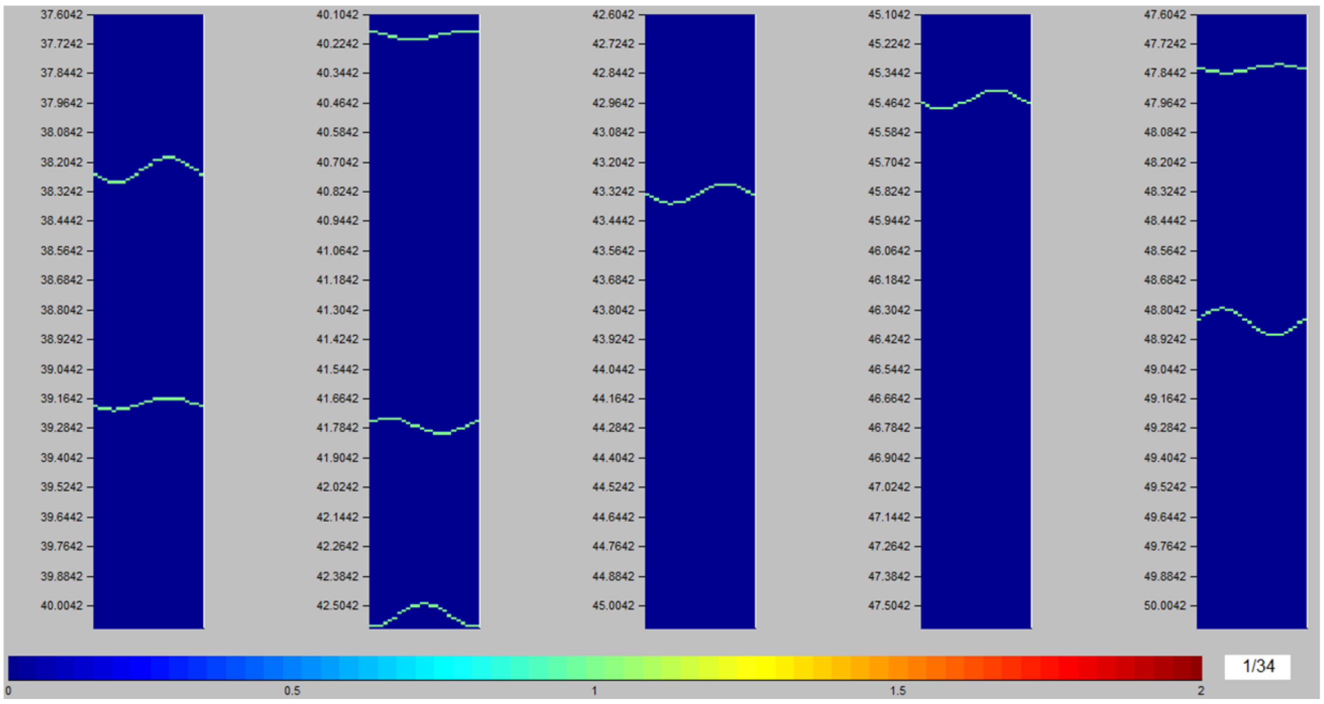
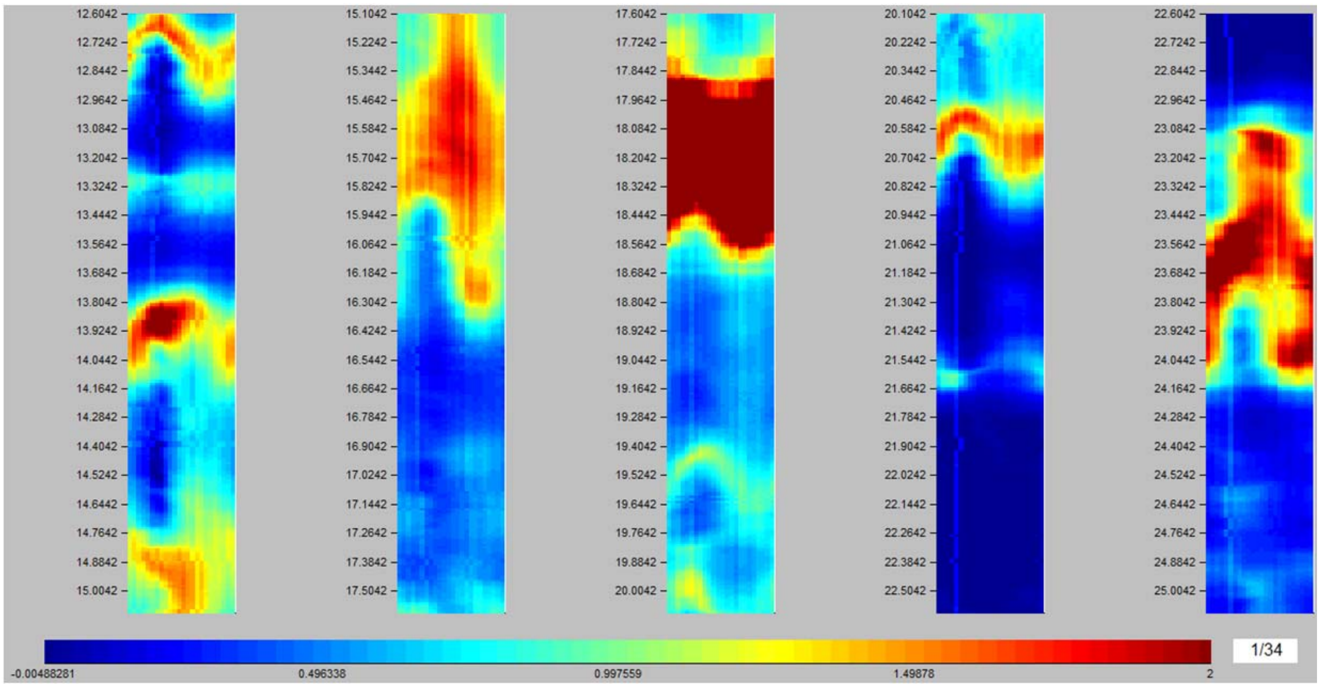



Figure 32 Top: same as Figure 8_original image. Bottom: Fractures found by the program



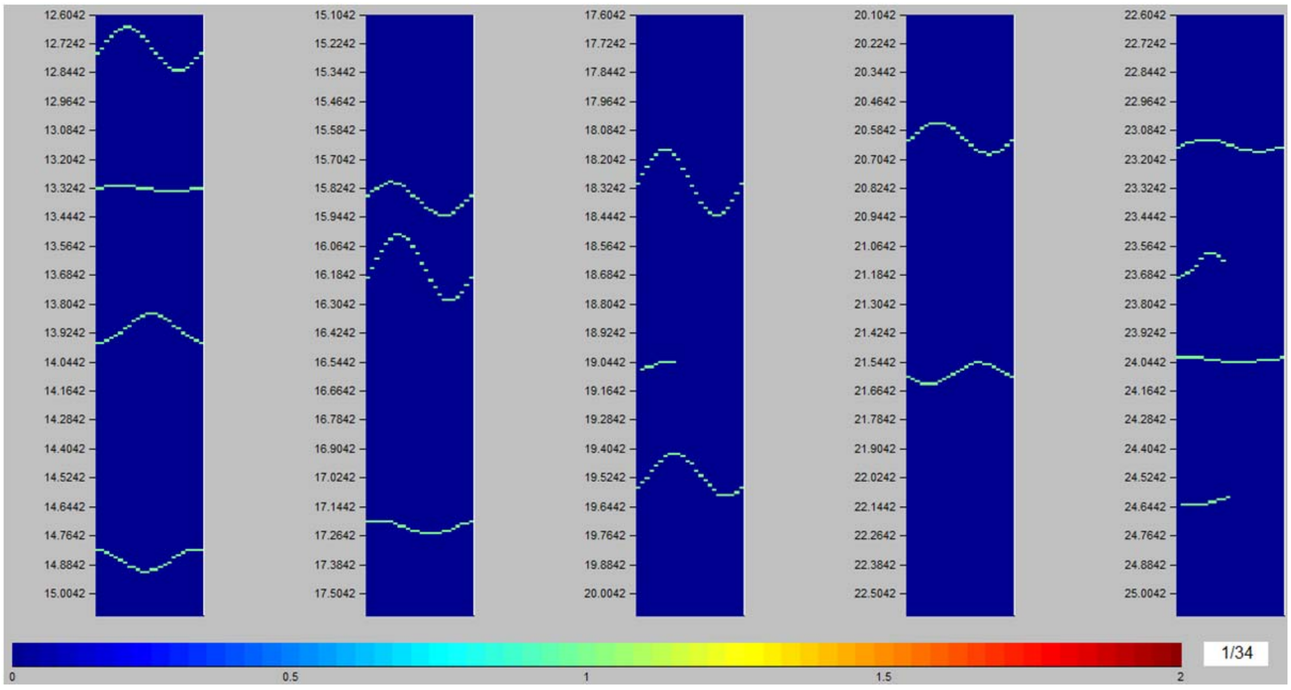


Figure 33 Top : resistivity image from Geostress. Bottom: Program's results. Some misinterpretation can be seen for fractures with large apertures (15.82m) and for fractures with high dips (23.68m)

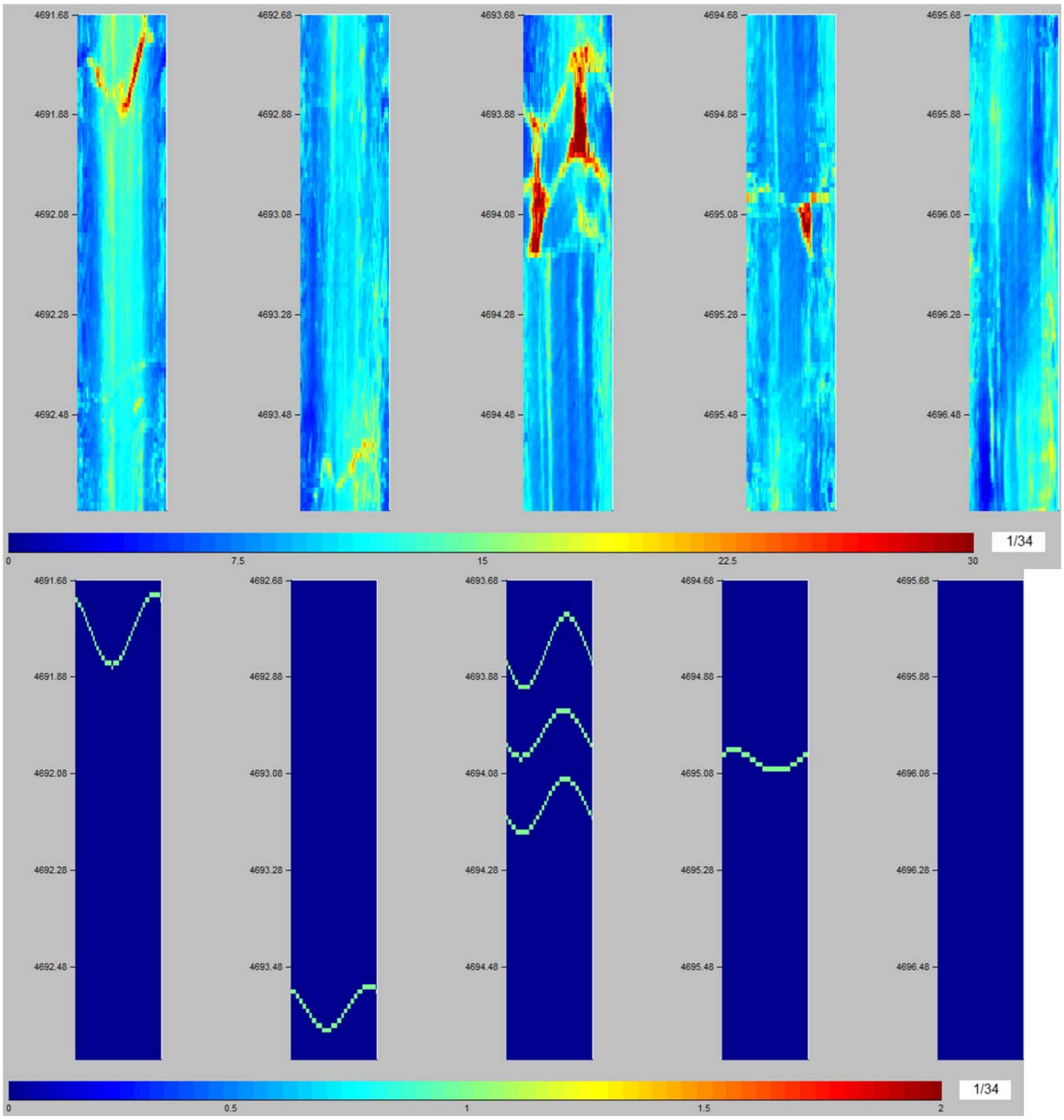


Figure 34 Top : UBI image from Statoil. Bottom: Program's results. All the sinusoidal events are well detected

Conclusion

The goal of this project was to create a tool using Matlab that could automatically extract the fracture information from an image log. An image processing usually has two steps: edge mapping and feature extraction; and the tool presented in this work has been built following the same architecture. First, different edge mapping methods have been tested on real data and a method involving a band pass filter has finally been implemented. Secondly, three different ways of extracting sinusoids from images have been tested on synthetic data: a method using correlations, another using the Hough transform and a last one using the Radon transform. These methods have been further tested on real data. Thanks to the results of these tests a new method was implemented that takes advantage of a partitioning algorithm to make the sinusoid extraction more effective. The advantages of the implemented method are that fractures with imperfect sinusoidal shape as well as crossing fractures can be interpreted and that it is quite fast. In the end, the designed program has been successfully tested on resistivity and sonic images with satisfactory results (two of three fractures were properly detected in the resistivity image).

Thanks to the tests on real data, several points that could be improved have been highlighted like the computing time and the detection of high dip or low contrast fractures. Unfortunately those improvements have not been implemented before the end of the thesis but they should however be taken care of in future versions of the program.

References

[1] van Ginkel M., Kraaijveld Robust M.A., van Vliet L.J., Reding E.P., *Curve Detection using a Radon Transform in Orientation Space Applied to Fracture Detection in Borehole Images*, R.L. Lagendijk et al. (eds), ASCI'01, Proc. 7th Annual Conference of the Advanced School for Computing and Imaging (Heijen, The Netherlands, May 30 - June 1), ASCI, Delft, 2000, 299-306

[2] Cornet F. H., Doan M.L., Fontbonne F., *Electrical imaging and hydraulic testing for a complete stress determination*, International Journal of Rock Mechanics & Mining Sciences 40 (2003) 1225–1241, 2003

[3] Torres D., Strickland R. and Gianzero M., *A New Approach to Determining Dip and Strike Using Borehole Images* , SPWLA 31st Armual Logging Symposium, June 24-27.1990

[4] Vincent Ph., Gartner J.-E., Attali G., *An Approach to Detailed Dip Determination Using Correlation by Pattern Recognition,* Journal of petroleum technology February 1979, 232-240

[5] Antoine J.N., Delhomme J.P., *A Method To Derive Dips From Bedding Boundaries in Borehole Images,* SPE Fonnation Evaluation, June 1993, 96-102

[6] Ye S.J., Rabiller P., *Automated Fracture Detection on High Resolution Resistivity Borehole Imagery,* SPE Annual Technical Conference and Exhibition held in New Orleans, Louisiana. 27-30 September 1998. 777-784

[7] Delhomme J.P. , *A quantitative characterization of formation heterogeneities based on borehole image analysis,* SPWLA 33rd Annual Logging Symposium, June 14-17, 1992

[8] Yin X.C., Liu Q., Hao H.W., Wang Z.B., Huang K. , *A Rock Structure Recognition System Using FMI Images,* C.S. Leung, M. Lee, and J.H. Chan (Eds.): ICONIP 2009, Part I, LNCS 5863, pp. 838–845, 2009

[9] Langeland H., *Imaging in boreholes*, NTNU 2012

(1) http://www.slb.com/~/media/Files/evaluation/brochures/wireline_open_hole/geology/ubi_br.pdf
(2) http://www.mathworks.se/help/

Appendix

In the following, the Matlab code of the three synthetic cases on which were tried the extraction methods are shown.

Synthetic cases

If copy/pasted directly in Matlab the synthetic cases are ready to work.

Correlation Method

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
clear all;
close all;
tic

%parameters of the code
ncol=100;
nline=1000;
nmod=100;
x=linspace(0,2*pi,ncol);
Amax=2;

%paramters of the synthetic data
% A_synth=[1.8];
A_synth=[0.8 0.8 1.4];
% phi_synth=[pi/6];
phi_synth=[3*pi/4 -pi/4 pi/6];
% thick_synth=[5];
thick_synth=[5 5 9];
% depth_synth=[0];
depth_synth=[3 0 0];

%displays the values defining the synthetic data
n_synth=length(depth_synth);
str=[ 'A0 =' , num2str(A_synth(1,:)) ];
disp(str);
str=[ 'phi0 =' , num2str(phi_synth(1,:)) ];
disp(str);
str=[ 'depth0 =' , num2str(depth_synth(1,:)) ];
disp(str);

%creates a matrix with the synthetic data in it
mat=zeros(nline,ncol);
for k=1:n_synth
    f=A_synth(1,k)*cos(x-phi_synth(1,k))+depth_synth(1,k);
    for i=1:ncol
        line=nline/2+1+round(nmod*f(1,i));
        mat(nline/2+1+round(nmod*f(1,i))-
thick_synth(k):nline/2+1+round(nmod*f(1,i))+thick_synth(k),i)=thick_synth(k
);
    end
end

% %designs an FMI type image with gaps
% j=1;
% mat_void=zeros(nline,ncol);
% for i=1:ncol
%     if j>0
%         mat_void(:,i)=1;
%     end
%     if j==15
%         j=-10;
%     end
%     j=j+1;
% end
% mat=mat.*mat_void;


%plots the sinusoids
figure
pcolor(mat), shading interp
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')


%% Finds events with a certain thickness in every trace
min_thick=3;
max_thick=30;
delta_res_thresh=3;
mat_edge=zeros(nline,ncol);

for j=1:ncol
    i=1;
    while i<=nline-max_thick-1
        if abs(mat(i,j)-mat(i+1,j))>=delta_res_thresh
            ii=1;
            while ii<max_thick
                if abs(mat(i+ii,j)-mat(i+ii+1,j))>=delta_res_thresh
                    break
                else
                    ii=ii+1;
                end
            end
            if ii>=min_thick && ii<max_thick
                mat_edge(i+floor(ii/2),j)=ii;
            end
            i=i+ii+1;
        else
            i=i+1;
        end
    end
end

mat_edge=round(mat_edge);
pcolor(mat_edge), shading interp;
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')


%% Correlation edges Regroups the points together in groups thanks to the
correlations
n_free=21; %odd number freedom of search
n_free_2=(n_free-1)/2;
mat_corre=zeros(nline,ncol,4);
corre_thresh=0.7;
```

```
%initialisation j=1
mat_edge(1:max_thick,:)=0;
mat_edge(nline-max_thick+1:nline,:)=0;
i_edge=find(mat_edge(:,1)>0);
n=length(i_edge);
k=0;
for i=1:n
    if mod(mat_edge(i_edge(i),1),2)==0
        vec=mat(i_edge(i)-
mat_edge(i_edge(i),1)/2:i_edge(i)+mat_edge(i_edge(i),1)/2-1,1);
        vec_mat=mat(i_edge(i)-mat_edge(i_edge(i),1)/2-
n_free_2:i_edge(i)+mat_edge(i_edge(i),1)/2-1+n_free_2,2);
    else
        vec=mat(i_edge(i)-(mat_edge(i_edge(i),1)-
1)/2:i_edge(i)+(mat_edge(i_edge(i),1)-1)/2,1);
        vec_mat=mat(i_edge(i)-(mat_edge(i_edge(i),1)-1)/2-
n_free_2:i_edge(i)+(mat_edge(i_edge(i),1)-1)/2+n_free_2,2);
    end
    [corre,i_corre]=max_corre(vec,vec_mat,n_free);
    i_corre=i_corre-n_free_2-1;

    if corre>=corre_thresh
        mat_corre(i_edge(i),1,1)=corre;
        mat_corre(i_edge(i)+i_corre,2,1)=corre;
        mat_corre(i_edge(i),1,2)=mat_edge(i_edge(i),1);
        mat_corre(i_edge(i)+i_corre,2,2)=mat_edge(i_edge(i),1);
        k=k+1;
        mat_corre(i_edge(i),1,3)=k;
        mat_corre(i_edge(i)+i_corre,2,3)=k;
        mat_corre(i_edge(i)+i_corre,2,4)=i_corre;
    end
end

i_corre_thresh=3;
%for all the other j
for j=2:ncol-1
    mat_edge(1:max_thick,:)=0;
    mat_edge(nline-max_thick+1:nline,:)=0;
    i_edge=find(mat_edge(:,j));
    n=length(i_edge);

    %Makes the regrouping if necessary
    for i=1:n
        if mod(mat_edge(i_edge(i),j),2)==0
            vec=mat_corre(i_edge(i)-
mat_edge(i_edge(i),j)/2:i_edge(i)+mat_edge(i_edge(i),j)/2-1,j,1);
        else
            vec=mat_corre(i_edge(i)-(mat_edge(i_edge(i),j)-
1)/2:i_edge(i)+(mat_edge(i_edge(i),j)-1)/2,j,1);
        end
        i_edge_find=find(vec);
        if isempty(i_edge_find)~=1
            m=length(i_edge_find);
            if m==1
                temp_i=i_edge(i);
                if mod(mat_edge(i_edge(i),j),2)==0
                    i_edge(i)=i_edge(i)-
mat_edge(i_edge(i),j)/2+i_edge_find-1;
                else
                    i_edge(i)=i_edge(i)-(mat_edge(i_edge(i),j)-
1)/2+i_edge_find-1;
```

```
                end
                mat_edge(temp_i,j)=0;
                mat_edge(i_edge(i),j)=mat_corre(i_edge(i),j,2);
            end
        end
    end

    %Goes on with the points in a column
    for i=1:n
        if mod(mat_edge(i_edge(i),j),2)==0
            vec=mat(i_edge(i)-
mat_edge(i_edge(i),j)/2:i_edge(i)+mat_edge(i_edge(i),j)/2-1,j);
            vec_mat=mat(i_edge(i)-mat_edge(i_edge(i),j)/2-
n_free_2:i_edge(i)+mat_edge(i_edge(i),j)/2-1+n_free_2,j+1);
        else
            vec=mat(i_edge(i)-(mat_edge(i_edge(i),j)-
1)/2:i_edge(i)+(mat_edge(i_edge(i),j)-1)/2,j);
            vec_mat=mat(i_edge(i)-(mat_edge(i_edge(i),j)-1)/2-
n_free_2:i_edge(i)+(mat_edge(i_edge(i),j)-1)/2+n_free_2,j+1);
        end
        [corre,i_corre]=max_corre(vec,vec_mat,n_free);
        i_corre=i_corre-n_free_2-1;

        if corre>=corre_thresh
            if abs(i_corre-mat_corre(i_edge(i),j,4))<i_corre_thresh
                if mat_corre(i_edge(i),j,3)==0
                    k=k+1;
                    mat_corre(i_edge(i),j,3)=k;
                    mat_corre(i_edge(i)+i_corre,j+1,3)=k;
                else

mat_corre(i_edge(i)+i_corre,j+1,3)=mat_corre(i_edge(i),j,3);
                end
            end
            if mat_corre(i_edge(i),j,4)==0 ||
mat_corre(i_edge(i),j,4)~=0&&abs(i_corre-
mat_corre(i_edge(i),j,4))<i_corre_thresh
                mat_corre(i_edge(i),j,1)=corre;
                mat_corre(i_edge(i)+i_corre,j+1,1)=corre;
                mat_corre(i_edge(i),j,2)=mat_edge(i_edge(i),j);
                mat_corre(i_edge(i)+i_corre,j+1,2)=mat_edge(i_edge(i),j);
                mat_corre(i_edge(i)+i_corre,j+1,4)=i_corre;
            end
        end
    end
end

%% Plots the regrouped points
figure
pcolor(mat_corre(:,:,3)), shading interp
colormap(jet)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')

%% Processing of the regrouped points
%finds the arguments of the best fitting sinusoid
n_points_sin=5;
mat_fit=zeros(ncol,6,k);
mysin=@(Amplitude, Phase, Depth ,x) Amplitude * cos( x-Phase ) + Depth;
```

```
kk=0;
for i=1:k
    [ifind,jfind] = find(mat_corre(:,:,3)==i);
    if length( ifind ) >= n_points_sin
        m=mean(ifind);
        sinfit=fit( x(jfind)' , ifind , mysin , 'StartPoint', [100,0,m]);
        kk=kk+1;
        mat_fit(1,1,kk)=length( ifind );
        mat_fit(1:length(ifind) ,2,kk)=ifind;
        mat_fit(1:length(ifind) ,3,kk)=jfind;
        mat_fit(1,4,kk)=sinfit.Amplitude;
        mat_fit(1,5,kk)=sinfit.Phase;
        mat_fit(1,6,kk)=sinfit.Depth;
    end
end
mat_fit(:,:,kk+1:end)=[];

%regroups the points belonging to the same sinusoid together and
%recalculates the arguments of the sinusoid
prec=0.1;
for i=1:kk
    j=i+1;
    while j<kk+1
        if abs((mat_fit(1,4,j)-mat_fit(1,4,i))/mat_fit(1,4,i))<prec...
                && abs((mat_fit(1,5,j)-
mat_fit(1,5,i))/mat_fit(1,5,i))<prec...
                && abs((mat_fit(1,6,j)-mat_fit(1,6,i))/mat_fit(1,6,i))<prec

ifind=vertcat(mat_fit(1:mat_fit(1,1,i),2,i),mat_fit(1:mat_fit(1,1,j),2,j));

jfind=vertcat(mat_fit(1:mat_fit(1,1,i),3,i),mat_fit(1:mat_fit(1,1,j),3,j));
            mat_fit(1,1,i)=mat_fit(1,1,i)+mat_fit(1,1,j);

            sinfit=fit( x(jfind)' , ifind , mysin , 'StartPoint',
[mat_fit(1,4,i),mat_fit(1,5,i),mat_fit(1,6,i)]);
            mat_fit(1:length(ifind) ,2,i)=ifind;
            mat_fit(1:length(ifind) ,3,i)=jfind;
            mat_fit(1,4,i)=sinfit.Amplitude;
            mat_fit(1,5,i)=sinfit.Phase;
            mat_fit(1,6,i)=sinfit.Depth;

            mat_fit(:,:,j)=[];
            j=j-1;
            kk=kk-1;
        end
        j=j+1;
    end
end

%removes all the groups containing less than n_points_sin_tot points
n_points_sin_tot=8;
i=1;
while i<kk+1
    if mat_fit(1,1,i)<n_points_sin_tot
        mat_fit(:,:,i)=[];
        i=i-1;
        kk=kk-1;
    end
    i=i+1;
```

```
end

mat_fit(1,4,:)=mat_fit(1,4,:)/100;
mat_fit(1,6,:)=(mat_fit(1,6,:)-500)/100;
mat_fit(1,:,:)

mat_final=zeros(nline,ncol);
for i=1:kk
    for j=1:mat_fit(1,1,i)
        mat_final(mat_fit(j,2,i),mat_fit(j,3,i))=i;
    end
end

figure
pcolor(mat_final), shading interp
colormap(jet)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')
toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ corre , i_corre ] = max_corre( vec , vec_mat , n_free )
%calculate the max correlation
n_vec=length(vec);
temp_corre=zeros(n_free,1);

if std(vec)~=0
    for i=1:n_free
        R=corrcoef([vec vec_mat(i:i+n_vec-1)]);
        temp_corre(i,1)=R(1,2);
    end
    [corre,i_corre]=nanmax(temp_corre);
else
    corre=0;
    i_corre=0;
end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Hough Transform

```
%main:
clear all;
close all;

tic

%parameters of the code
n_max=5;
ncol=100;
nline=1000;
nmod=1000;
Amax=0.2;
size_win=2*Amax*nmod;
x=linspace(0,2*pi,ncol);
y=linspace(-Amax,Amax,2*Amax*nmod+1);
ncos=100;
```

```matlab
ndef=100;
delta_h=0.01;
delta_step=10;%on suppose une mesure tous les mms
n_win=floor(((nline/nmod)-2*Amax)/delta_h)+1;
depth=zeros(n_win,1);
depth(1,1)=-0.3;
for i=2:n_win
    depth(i,1)=depth(i-1,1)+delta_h;
end

%paramters of the synthetic data
% A_synth=[0.8];
A_synth=[0.08 0.08 0.14];
% phi_synth=[pi/6];
phi_synth=[3*pi/4 -pi/4 pi/6];
% depth_synth=[0];
depth_synth=[0.3 0 0];

%displays the values defining the synthetic data
n_synth=length(depth_synth);
str=[ 'A0 =' , num2str(A_synth(1,:)) ];
disp(str);
str=[ 'phi0 =' , num2str(phi_synth(1,:)) ];
disp(str);
str=[ 'Depth0 =' , num2str(depth_synth(1,:)) ];
disp(str);

%creates a matrix with the synthetic data in it
mat=zeros(nline,ncol);
for k=1:n_synth
    f=A_synth(1,k)*cos(x-phi_synth(1,k))+depth_synth(1,k);
    for i=1:ncol
        mat(nline/2+1+round(nmod*f(1,i)),i)=1;
    end
end


rand_th=0.98;
rand_mat=rand(nline,ncol);
mat(rand_mat>rand_th)=1;

%designs an FMI type image with gaps
j=1;
mat_void=zeros(nline,ncol);
for i=1:ncol
    if j>0
        mat_void(:,i)=1;
    end
    if j==15
        j=-10;
    end
    j=j+1;
end
mat=mat.*mat_void;

%plots the sinusoids
figure
pcolor(mat), shading interp
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
```

```matlab
xlabel('Phi (rad)')
ylabel('Depth (mm)')
tic
%% Draw the lines in the parameter space
l=1;
mat_para=zeros(ndef+1,ndef+1,n_win);
for ii=1:n_win
    mat_win=mat(l:l+size_win-1,:);
    l=l+delta_step;
    mat_para(:,:,ii)=sin_hough(mat_win,x,y,ndef,ncos,Amax);
end
toc
%% Plots the results in the parameter space
Apas=Amax/ndef;
A=0:Apas:Amax;
phipas=2*pi/ndef;
phi=-pi:phipas:pi;

figure
pcolor(phi,A(1:end-5),mat_para(1:end-5,:,61)), shading interp
xlabel('Phase')
ylabel('Amplitude')
colorbar

%% Find local maxima corresponding to the parameters of the sinusoids
[Maxima,MaxPos,Minima,MinPos]=MinimaMaxima3D(mat_para(1:end-
5,:,:),1,1,n_max,0);

%% Reorders the results and keeps only the clearest sinusoids
m_min=10;
n_sinus=0;
for i=1:n_max
    if mat_para(MaxPos(i,1),MaxPos(i,2),MaxPos(i,3))>=m_min
        n_sinus=n_sinus+1;
    end
end
win_sinus=zeros(n_sinus,3);
j=1;
for i=1:n_max
    if mat_para(MaxPos(i,1),MaxPos(i,2),MaxPos(i,3))>=m_min
        win_sinus(j,1)=A(MaxPos(i,1));
        win_sinus(j,2)=phi(MaxPos(i,2));
        win_sinus(j,3)=depth(MaxPos(i,3));
        j=j+1;
    end
end

%% Test the sinusoids found previously
%the sinusoids have to close enough to a certain amount of points and
%the standard deviation has also to be small enough
delta_h=4;
n_tresh=30;
std_thresh=1000;
mat_std=mat;

for i=1:n_sinus
    f=win_sinus(i,1)*cos(x-win_sinus(i,2))+win_sinus(i,3);
    k=0;
    sum_k=0;
    for j=1:ncol
```

```
        l=nline/2+1+round(nmod*f(1,j));
        [temp_mat,~]=find(mat(l-delta_h:l+delta_h,j)==1);
        temp_mat=temp_mat-l;
        [mm,m]=size(temp_mat);

        if mm>0&&m>0
            k=k+m;
            for jj=1:m
                sum_k=sum_k+temp_mat(1,jj)^2;
            end
        end
    end
    if k>1
        std=sqrt(sum_k/(k-1));
    end

    if k>=n_tresh&&std<=std_thresh
        str=[ 'True A =' , num2str(win_sinus(i,1)) ];
        disp(str);
        str=[ 'True phi =' , num2str(win_sinus(i,2)) ];
        disp(str);
        str=[ 'True depth =' , num2str(win_sinus(i,3)) ];
        disp(str);


        f=win_sinus(i,1)*cos(x-win_sinus(i,2))+win_sinus(i,3);
        for ii=1:ncol

mat_std(nline/2+1+round(nmod*f(1,ii)),ii)=mat_std(nline/2+1+round(nmod*f(1,
ii)),ii)+1;
        end
    end
end


%% Display

figure
pcolor(mat_std), shading interp
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')

toc
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%




function [ temp_para ] = sin_hough( mat_win , x , y , ndef , ncos, Amax)
%Makes a sinusoidal hough transform on mat_win

phi0=-pi:2*pi/ncos:pi;
Apas=Amax/ndef;
phipas=2*pi/ndef;

[Y0,X0]=find(mat_win==1);
x0=x(X0);
```

```
y0=y(Y0);
[t,m]=size(x0);

if m~=0||t~=0
    temp_para=zeros(ndef+1,ndef+1,'uint8');
    for i=1:m
        phik=-pi;
        iphik=1;

        % j=1, find the position of the first point
        if (x0(1,i)-phi0(1,1))~=0
            A0=y0(1,i)/(cos(x0(1,i)-phi0(1,1)));
            if A0>=0
                if A0>Amax
                    A0=Amax;
                end

                while 1
                    if phik>phi0(1,1)||abs(phik-phi0(1,1))<0.000001
                        j0=iphik;
                        break
                    end
                    iphik=iphik+1;
                    phik=phik+phipas;
                end

                iAk=1;
                Ak=0;
                if A0==0
                        i0=1;
                else
                    while 1
                        if A0<Ak+Apas||abs(A0-Ak-Apas)<0.000001
                            i0=iAk;
                            break
                        end
                        iAk=iAk+1;
                        Ak=Ak+Apas;
                    end
                end
                temp_para(i0,j0)=temp_para(i0,j0)+1;
            else
                iAk=1;
                Ak=0;
            end
        else
            iAk=1;
            Ak=0;
        end

        %loop on the other values
        for j=2:ncos+1
            if (x0(1,i)-phi0(1,j))~=0

                A0=y0(1,i)/(cos(x0(1,i)-phi0(1,j)));
                if A0>=0
                    if A0>Amax
                        A0=Amax;
                    end
                    while 1
                        if phik>phi0(1,j)||abs(phik-phi0(1,j))<0.000001
                            j0=iphik;
                            break
                        end
                        iphik=iphik+1;
                        phik=phik+phipas;
                    end


                    if Ak<A0&&A0<Ak+Apas||abs(A0-Ak-Apas)<0.000001
                        i0=iAk;
                    else
                        if A0<Ak||abs(A0-Ak)<0.000001
                            while 1
                                if Ak-Apas<A0||abs(A0-Ak)<0.000001
                                    i0=iAk;
                                    break
                                end
                                iAk=iAk-1;
                                Ak=Ak-Apas;
                            end
                        else
                            iAk=iAk+1;
                            Ak=Ak+Apas;
                            while 1
                                if A0<Ak+Apas||abs(A0-Ak-Apas)<0.000001
                                    i0=iAk;
                                    break
                                end
                                iAk=iAk+1;
                                Ak=Ak+Apas;
                            end
                        end
                    end
                    temp_para(i0,j0)=temp_para(i0,j0)+1;
                end
            end
        end
    end
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function
[Maxima,MaxPos,Minima,MinPos]=MinimaMaxima3D(Input,Robust,LookInBoundaries,
numbermax,numbermin)
% V 1.0 Dec 13, 07
% Author Sam Pichardo.
% This  function finds the local minima and maxima in a 3D Cartesian data.
% It's assumed that the data is uniformly distributed.
% The minima and maxima are calculated using a multi-directional
derivation.
%
% Use:
%
%
[Maxima,MaxPos,Minima,MinPos]=MinimaMaxima3D(Input,[Robust],[LookInBoundari
es],[numbermax],[numbermin])
%
```

```
% where Input is the 3D data and Robust (optional and with a default value
% of 1) indicates if the multi-directional derivation should include the
% diagonal derivations.
%
% Input has to have a size larger or equal than [3 x 3 x 3]
%
% If Robust=1, the total number of derivations taken into account are 26: 6
% for all surrounding elements colliding each of the faces of the unit
cube;
% 10 for all the surrounding elements in diagonal.
%
% If Robust =0, then only the 6 elements of the colliding faces are
considered
%
% The function returns in Maxima and MaxPos, respectively,
% the values (numbermax) and subindexes (numbermax x 3) of local maxima
% and position in Input. Maxima (and the subindexes) are sorted in
% descending order.
% Similar situation for Minima and MinimaPos witn a numbermin elements but
% with the execption of being sorted in ascending order.
%
% IMPORTANT: if numbermin or numbermax are not specified, ALL the minima
% or maxima will be returned. This can be a useless for highly
% oscillating data
%
% LookInBoundaries (default value of 0) specifies if a search of the
minima/maxima should be
% done in the boundaries of the matrix. This situation depends on the
% the desire application. When it is not activated, the algorithm WILL NOT
% FIND ANY MINIMA/MAXIMA on the 6 layers of the boundaries.
% When it is activated, the finding minima and maxima on the boundaries is
done by
% replicating the extra layer as the layer 2 (or layer N-1, depending of
the boundary)
% By example (and using a 2D matrix for simplicity reasons):
% For the matrix
% [ 4 1 3 7
%   5 7 8 8
%   9 9 9 9
%   5 6 7 9]
%
% the calculation of the partial derivate following the -x direction will
be done by substrascting
% [ 5 7 8 8
%   4 1 3 7
%   5 7 8 8
%   9 9 9 9]
% to the input. And so on for the other dimensions.
% Like this, the value "1" at the coordinate (1,2) will be detected as a
% minima. Same situation for the value "5" at the coordinate (4,1)


if nargin <1
    test=load('temp.mat');
    pf=test.uresTot(test.EvalLims(2,1):test.EvalLims(2,2));

pf=reshape(pf,length(test.EvalCoord{2}.Ry),length(test.EvalCoord{2}.Rx),len
gth(test.EvalCoord{2}.Rz));
    Input = abs(pf)*1.5e6;
    clear test;
    clear pf;
    Robust =1;
end

Asize=size(Input);

if length(Asize)<3
    error('MinimaMaxima3D can only works with 3D matrices ');
end


if (Asize(1)<3 || Asize(2)<3 || Asize(3)<3)
    error('MinimaMaxima3D can only works with matrices with dimensions 
equal or larger to [3x3x3]');
end

if ~isreal(Input)
    warning('ATTENTION, complex values detected!!, using abs(Input)');
    Input=abs(Input);
end

if ~exist('Robust','var')
    Robust=1;
end

if ~exist('LookInBoundaries','var')
    LookInBoundaries=0;
end

if ~exist('numbermax','var')
    numbermax=0;
end

if ~exist('numbermin','var')
    numbermin=0;
end

[xx_base,yy_base,zz_base]=ndgrid(1:Asize(1),1:Asize(2),1:Asize(3));


IndBase=sub2ind(Asize,xx_base(:),yy_base(:),zz_base(:));

if Robust ~= 0
    Numbder_dd=26;
else
    Numbder_dd=6;
end

if LookInBoundaries==0
    lx=1:Asize(1);
    lx_p1=[2:Asize(1),Asize(1)];
    lx_m1=[1,1:Asize(1)-1];
    ly=1:Asize(2);
    ly_p1=[2:Asize(2),Asize(2)];
    ly_m1=[1,1:Asize(2)-1];
    lz=1:Asize(3);
    lz_p1=[2:Asize(3),Asize(3)];
    lz_m1=[1,1:Asize(3)-1];
else
    lx=1:Asize(1);
    lx_p1=[2:Asize(1),Asize(1)-1]; %We replicate the layer N-1 as the layer
N+1
    lx_m1=[2,1:Asize(1)-1]; %We replicate the layer 2 as the layer -1
    ly=1:Asize(2);
    ly_p1=[2:Asize(2),Asize(2)-1]; %We replicate the layer N-1 as the layer
N+1
    ly_m1=[2,1:Asize(2)-1]; %We replicate the layer 2 as the layer -1
    lz=1:Asize(3);
    lz_p1=[2:Asize(3),Asize(3)-1]; %We replicate the layer N-1 as the layer
N+1
    lz_m1=[2,1:Asize(3)-1];%We replicate the layer 2 as the layer -1
end

for n_dd=1:Numbder_dd
    switch n_dd
        case 1
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1)
            [xx,yy,zz]=ndgrid(lx_p1,ly,lz);

        case 2
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1)
            [xx,yy,zz]=ndgrid(lx_m1,ly,lz);

        case 3
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(y)-
elem(y+1)
            [xx,yy,zz]=ndgrid(lx,ly_p1,lz);

        case 4
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(y)-
elem(y-1)
            [xx,yy,zz]=ndgrid(lx,ly_m1,lz);

        case 5
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(z)-
elem(z+1)
            [xx,yy,zz]=ndgrid(lx,ly,lz_p1);

         case 6
            %%%%%%%%%%%%%%%%%%% %% This index is used to calculated
elem(z)-elem(z-1)
            [xx,yy,zz]=ndgrid(lx,ly,lz_m1);
        case 7
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y+1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_p1,lz);
        case 8
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y-1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_m1,lz);
        case 9
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y-1)
            [xx,yy,zz]=ndgrid(lx_m1,ly_m1,lz);
        case 10
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y+1)
            [xx,yy,zz]=ndgrid(lx_m1,ly_p1,lz);
        case 11
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,z+1)
            [xx,yy,zz]=ndgrid(lx_p1,ly,lz_p1);
        case 12
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,z-1)
            [xx,yy,zz]=ndgrid(lx_p1,ly,lz_m1);
        case 13
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,z-1)
            [xx,yy,zz]=ndgrid(lx_m1,ly,lz_m1);
        case 14
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,z+1)
            [xx,yy,zz]=ndgrid(lx_m1,ly,lz_p1);
        case 15
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(y+1,z+1)
            [xx,yy,zz]=ndgrid(lx,ly_p1,lz_p1);
        case 16
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(y+1,z-1)
            [xx,yy,zz]=ndgrid(lx,ly_p1,lz_m1);
        case 17
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(y-1,z-1)
            [xx,yy,zz]=ndgrid(lx,ly_m1,lz_m1);
        case 18
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(y-1,z+1)
            [xx,yy,zz]=ndgrid(lx,ly_m1,lz_p1);
         case 19
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y+1,z+1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_p1,lz_p1);
         case 20
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y+1,z-1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_p1,lz_m1);
         case 21
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y-1,z+1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_m1,lz_p1);
         case 22
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x+1,y-1,z-1)
            [xx,yy,zz]=ndgrid(lx_p1,ly_m1,lz_m1);
         case 23
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y+1,z+1)
            [xx,yy,zz]=ndgrid(lx_m1,ly_p1,lz_p1);
         case 24
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y+1,z-1)
            [xx,yy,zz]=ndgrid(lx_m1,ly_p1,lz_m1);
         case 25
           %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y-1,z+1)
            [xx,yy,zz]=ndgrid(lx_m1,ly_m1,lz_p1);
         case 26
```

```
            %%%%%%%%%%%%%%%%%% %% This index is used to calculated elem(x)-
elem(x-1,y-1,z-1)
             [xx,yy,zz]=ndgrid(lx_m1,ly_m1,lz_m1);

    end

    Ind_dd=sub2ind(Asize,xx(:),yy(:),zz(:));

    part_deriv = Input(IndBase)-Input(Ind_dd);

    if n_dd >1
        MatMinMax= (sign_Prev_deriv==sign(part_deriv)).*MatMinMax;
    else
        MatMinMax=sign(part_deriv);
    end

    sign_Prev_deriv=sign(part_deriv);
end

%Well , now the easy part, all values MatMinMax ==1 are local maximum and
%the values MatMinMax ==-1 are minimun

AllMaxima=find(MatMinMax==1);
AllMinima=find(MatMinMax==-1);

if numbermax ==0
    nmax=length(AllMaxima);
else
    nmax=numbermax;
end
nmax=min([nmax,length(AllMaxima)]);
smax=1:nmax;

if numbermin ==0
    nmin=length(AllMinima);
else
    nmin=numbermin;
end

nmin=min([nmin,length(AllMinima)]);

smin=1:nmin;

[Maxima,IndMax]=sort(Input(AllMaxima),'descend');
Maxima=Maxima(smax);
IndMax=AllMaxima(IndMax(smax));

MaxPos=zeros(nmax,3);
[MaxPos(:,1),MaxPos(:,2),MaxPos(:,3)]=ind2sub(Asize,IndMax);

[Minima,IndMin]=sort(Input(AllMinima));
Minima=Minima(smin);
IndMin=AllMinima(IndMin(smin));

MinPos=zeros(nmin,3);
[MinPos(:,1),MinPos(:,2),MinPos(:,3)]=ind2sub(Asize,IndMin);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Radon Transform

General radon transform with no sinusoids reconstruction but with a 3D picture of the orientation space:

```
clear all;
close all;

%parameters of the code
ncol=100;
nline=1000;
nmod=100;
x=linspace(0,2*pi,ncol);

%paramters of the synthetic data
% A_synth=[0.8];
A_synth=[0.8 0.8 1.4];
% phi_synth=[pi/6];
phi_synth=[3*pi/4 -pi/4 pi/6];
% depth_synth=[0];
depth_synth=[3 0 0];

%displays the values defining the synthetic data
n_synth=length(depth_synth);
str=[ 'phi0 =' , num2str(phi_synth(1,:)) ];
disp(str);
str=[ 'A0 =' , num2str(A_synth(1,:)) ];
disp(str);

%creates a matrix with the synthetic data in it
mat=zeros(nline,ncol);
for k=1:n_synth
    f=A_synth(1,k)*cos(x-phi_synth(1,k))+depth_synth(1,k);
    for i=1:ncol
        mat(nline/2+1+round(nmod*f(1,i)),i)=1;
    end
end


% %designs an FMI type image with gaps
% j=1;
% mat_void=zeros(nline,ncol+1);
% for i=1:ncol+1
%     if j>0
%         mat_void(:,i)=1;
%     end
%     if j==15
%         j=-10;
%     end
%     j=j+1;
% end
% mat=mat.*mat_void;


%plots the sinusoids
```

```
figure
pcolor(mat), shading interp


%% Creates the orientation space
n_max=1;
thr_rad=0;
[Y0,X0]=find(mat==1);
n_edge=length(X0);
mat_or=zeros(nline,ncol);
cst_mat_or=0;
mat_or(:,:,2)=cst_mat_or*ones(nline,ncol);
n_phi_rad=181; %181 or 361 or a multiple of 180  +1 or at least odd number
phi_min_thresh=0*pi/180;
phi_rad=linspace(phi_min_thresh,pi-phi_min_thresh,n_phi_rad);
n_phi_rad=n_phi_rad-1;
n_x_line=1000;
n_factor=5;%odd number. Used to refine the matrix.
n_factor_2=(n_factor-1)/2;
size_win=40;
n_thresh=10;



% Finds the path in the matrix along which the sum is performed
[cell_path_rad]=find_path( phi_rad , size_win , n_x_line , n_factor );
n_cell_path_rad=length(cell_path_rad);

% Creates a bigger matrix to remove the effect of edges
mat_enlarged=zeros(nline+2*size_win,ncol+2*size_win);
mat_enlarged(size_win+1:nline+size_win,1:size_win)=mat(:,ncol-
size_win+1:ncol);
mat_enlarged(size_win+1:nline+size_win,size_win+1:ncol+size_win)=mat;
mat_enlarged(size_win+1:nline+size_win,ncol+size_win+1:end)=mat(:,1:size_wi
n);

% Creates a larger matrix large_mat with a size of
n_factor*length(mat_enlarged) to
% increase the resolution
if n_factor==1
    XXL_mat=mat;
else
    XXL_mat=zeros((nline+2*size_win)*n_factor,(ncol+2*size_win)*n_factor);
    for i=1:nline+2*size_win
        for j=1:ncol+2*size_win
            if mat_enlarged(i,j)~=0
                for ii=1:n_factor
                    for jj=1:n_factor
                        XXL_mat((i-1)*n_factor+ii,(j-
1)*n_factor+jj)=mat_enlarged(i,j);
                    end
                end
            end
        end
    end
end

% Computes the sum of lines with different angles in a matrix centred on an
edge point
% Core of the radon transform
figure
```

```
for i=1:n_edge
    mat_local=XXL_mat((Y0(i)-1)*n_factor+1:(Y0(i)+2*size_win)*n_factor,...
        (X0(i)-1)*n_factor+1:(X0(i)+2*size_win)*n_factor);

    sum_cell_path_rad=zeros(n_cell_path_rad+2,1);
    for j=1:n_cell_path_rad
        sum_cell_path_rad(j+1,1)=sum(mat_local(cell_path_rad{1,j}));
    end
    sum_cell_path_rad(1,1)=sum_cell_path_rad(n_cell_path_rad+1,1);
    sum_cell_path_rad(n_cell_path_rad+2,1)=sum_cell_path_rad(2,1);

    [pks,locs] = findpeaks(sum_cell_path_rad , 'NPEAKS', n_max,
'SORTSTR','descend');
    if pks(1)>=n_thresh
        mat_or(Y0(i),X0(i),1)=pks(1);
        plot3(Y0(i),X0(i),mat_or(Y0(i),X0(i),1),'b+'), hold on
        mat_or(Y0(i),X0(i),2)=cell_path_rad{2,locs(1)}(3,1);
    end
end
hold off
mat_or(:,:,2)=abs(mat_or(:,:,2));



%%
figure
pcolor(mat_or(:,:,1)), shading interp
colormap(jet)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

In the next case a sinusoidal reconstruction has been added using the simplistic method exposed in the chapter Radon transform.

```
clear all;
close all;
tic
%parameters of the code
ncol=100;
nline=1000;
nmod=100;
x=linspace(0,2*pi,ncol);

%paramters of the synthetic data
% A_synth=[0.8];
A_synth=[0.8 0.8 1.4];
% phi_synth=[pi/6];
phi_synth=[3*pi/4 -pi/4 pi/6];
% depth_synth=[0];
depth_synth=[3 0 0];

%displays the values defining the synthetic data
n_synth=length(depth_synth);
str=[ 'phi0 =' , num2str(phi_synth(1,:)) ];
disp(str);
str=[ 'A0 =' , num2str(A_synth(1,:)) ];
disp(str);
```

```
str=[ 'Depth0 =' , num2str(depth_synth(1,:)) ];
disp(str);

%creates a matrix with the synthetic data in it
mat=zeros(nline,ncol);
for k=1:n_synth
    f=A_synth(1,k)*cos(x-phi_synth(1,k))+depth_synth(1,k);
    for i=1:ncol
        mat(nline/2+1+round(nmod*f(1,i)),i)=1;
    end
end



% %designs an FMI type image with gaps
% j=1;
% mat_void=zeros(nline,ncol+1);
% for i=1:ncol+1
%     if j>0
%         mat_void(:,i)=1;
%     end
%     if j==15
%         j=-10;
%     end
%     j=j+1;
% end
% mat=mat.*mat_void;



%plots the sinusoids
figure
pcolor(mat), shading interp
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')

%% Creates the orientation space
n_max=1;
thr_rad=0;
[Y0,X0]=find(mat==1);
n_edge=length(X0);
mat_or=zeros(nline,ncol);
cst_mat_or=5;
mat_or(:,:,2)=cst_mat_or*ones(nline,ncol);
n_x_line=1000;
n_factor=3;%odd number. Used to refine the matrix.
n_factor_2=(n_factor-1)/2;
size_win=10;
n_thresh=10;

phi_min_thresh=20*pi/180;
n_phi_rad=41; %odd number
phi_rad_1=linspace(0,phi_min_thresh,n_phi_rad);
phi_rad_2=linspace(pi-phi_min_thresh,pi,n_phi_rad);
phi_rad=horzcat(phi_rad_1,phi_rad_2);
n_phi_rad=2*n_phi_rad-1;

% Finds the path in the matrix along which the sum is performed
[cell_path_rad]=find_path( phi_rad , size_win , n_x_line , n_factor );
n_cell_path_rad=length(cell_path_rad);
```

```
% Creates a bigger matrix to remove the effect of edges
mat_enlarged=zeros(nline+2*size_win,ncol+2*size_win);
mat_enlarged(size_win+1:nline+size_win,1:size_win)=mat(:,ncol-
size_win+1:ncol);
mat_enlarged(size_win+1:nline+size_win,size_win+1:ncol+size_win)=mat;
mat_enlarged(size_win+1:nline+size_win,ncol+size_win+1:end)=mat(:,1:size_wi
n);

% Creates a larger matrix large_mat with a size of
n_factor*length(mat_enlarged) to
% increase the resolution
if n_factor==1
    XXL_mat=mat;
else
    XXL_mat=zeros((nline+2*size_win)*n_factor,(ncol+2*size_win)*n_factor);
    for i=1:nline+2*size_win
        for j=1:ncol+2*size_win
            if mat_enlarged(i,j)~=0
                for ii=1:n_factor
                    for jj=1:n_factor
                        XXL_mat((i-1)*n_factor+ii,(j-
1)*n_factor+jj)=mat_enlarged(i,j);
                    end
                end
            end
        end
    end
end

% Computes the sum of lines with different angles in a matrix centred on an
edge point
% Core of the radon transform
figure
for i=1:n_edge
    mat_local=XXL_mat((Y0(i)-1)*n_factor+1:(Y0(i)+2*size_win)*n_factor,...
        (X0(i)-1)*n_factor+1:(X0(i)+2*size_win)*n_factor);

    sum_cell_path_rad=zeros(n_cell_path_rad+2,1);
    for j=1:n_cell_path_rad
        sum_cell_path_rad(j+1,1)=sum(mat_local(cell_path_rad{1,j}));
    end
    sum_cell_path_rad(1,1)=sum_cell_path_rad(n_cell_path_rad+1,1);
    sum_cell_path_rad(n_cell_path_rad+2,1)=sum_cell_path_rad(2,1);

    [pks,locs] = findpeaks(sum_cell_path_rad , 'NPEAKS', n_max,
'SORTSTR','descend');
    if isempty(pks)~=1
        if pks(1)>=n_thresh
            mat_or(Y0(i),X0(i),1)=pks(1);
            plot3(Y0(i),X0(i),mat_or(Y0(i),X0(i),1),'b+'), hold on
            mat_or(Y0(i),X0(i),2)=cell_path_rad{2,locs(1)}(3,1);
        end
    end
end
hold off

figure
pcolor(mat_or(:,:,2)), shading interp
colormap(hot)
```

```
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')

%% Reconstruct the sinusoid thanks to the radon method
%In this method NO blank column is required
%Looks for the points where the derivative is zero and tries to regroup all
%these points two by two. From the position of these max and min of the sin
%function it is easy to get the arguments of the sin function
phi=0;
n_i_del=10;
phi_del=pi/6;
n_j_del=round(ncol*phi_del/(2*pi));
ind_points_2=zeros(1000,2);

mat_or_2=cst_mat_or*ones(nline+2*n_i_del,ncol+2*n_j_del);
mat_or_2(n_i_del+1:nline+n_i_del,n_j_del+1:ncol+n_j_del)=mat_or(:,:,2);
n_ind_thresh=5;

i=0;
while phi<=phi_min_thresh
    phi=min(min(mat_or_2));
    if phi<cst_mat_or
        [i_phi,j_phi]=find(mat_or_2==phi);
        if sum(sum((mat_or_2(i_phi(1)-n_i_del:i_phi(1)+n_i_del,j_phi(1)-
n_j_del:j_phi(1)+n_j_del)-cst_mat_or)~=0))>=n_ind_thresh
            i=i+1;
            ind_points_2(i,1:2)=[i_phi(1)-n_i_del,j_phi(1)-n_j_del];
        end
        mat_or_2(i_phi(1)-n_i_del:i_phi(1)+n_i_del,j_phi(1)-
n_j_del:j_phi(1)+n_j_del)=cst_mat_or*ones(2*n_i_del+1,2*n_j_del+1);
        if j_phi(1)<n_j_del+1
            mat_or_2(i_phi(1)-
n_i_del:i_phi(1)+n_i_del,ncol+n_j_del+(j_phi(1)-
n_j_del):ncol+n_j_del)=cst_mat_or*ones(2*n_i_del+1,abs(j_phi(1)-n_j_del-
1));
        end
        if j_phi(1)>ncol-n_j_del
            mat_or_2(i_phi(1)-
n_i_del:i_phi(1)+n_i_del,n_j_del+1:n_j_del+(n_j_del-(ncol-
j_phi(1))))=cst_mat_or*ones(2*n_i_del+1,n_j_del-(ncol-j_phi(1)));
        end
    end
end
n_ind_points_2=i;

figure
pcolor(mat_or(:,:,2)), shading interp

%identifies if the sinusoid is oritend towards the top or the
%bottom at that point

%regroups the points two by two by checking the period, the amplitude and
%by testing the solution on the edge map
mat_edge=mat;
delta_h=15;
n_tresh=40;
ind_points=zeros(n_ind_points_2,4);
temp_ind=zeros(n_ind_points_2-1,2);
n_col_T=ncol/2;
```

```
Amax=2;
prec_T=10;
n_line_A=2*Amax*nmod+1;
n_loop=2;
p=0;
for k=1:n_loop
    i=1;
    while i<n_ind_points_2+1
        l=0;
        for j=i+1:n_ind_points_2
            if abs(ind_points_2(i,1)-ind_points_2(j,1))<=n_line_A &&
abs(abs(ind_points_2(i,2)-ind_points_2(j,2))-ncol/2)<=prec_T
                [Amplitude , Phase , Depth] =
find_arg(ind_points_2(i,:),ind_points_2(j,:),ncol);
                if test_sinusoid(mat_edge , Amplitude , Phase , Depth ,
delta_h , x )>=n_tresh
                    l=l+1;
                    temp_ind(l,:)=ind_points_2(j,:);
                    temp=j;
                end
            end
        end
        if l==1
            p=p+1;
            ind_points(p,1:2)=ind_points_2(i,:);
            ind_points(p,3:4)=temp_ind(1,:);
            ind_points_2(temp,:)=[];
            n_ind_points_2=n_ind_points_2-1;
        end
        i=i+1;
    end
end
ind_points(p+1:end,:)=[];

arg_sinus=zeros(p,3);
for i=1:p

[arg_sinus(i,1),arg_sinus(i,2),arg_sinus(i,3)]=find_arg(ind_points(i,1:2),i
nd_points(i,3:4),ncol);
end
arg_sinus
%%
for i=1:p
    f=arg_sinus(i,1)*cos(x-arg_sinus(i,2))+arg_sinus(i,3);
    for ii=1:ncol
        mat(round(f(1,ii)),ii)=mat(round(f(1,ii)),ii)+1;
    end
end

figure
pcolor(mat), shading interp
colormap(hot)
set(gca,'XTickLabel',2*pi/10:2*pi/10:2*pi)
xlabel('Phi (rad)')
ylabel('Depth (mm)')
toc

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [Amplitude , Phase , Depth] = find_arg(ind_1 ,ind_2 , ncol)
%finds the arguments of the cos function going through the two points of
```

```
%coordinates ind_1 and ind_2

Amplitude=abs(ind_2(1,1)-ind_1(1,1))/2;
Depth=(ind_2(1,1)+ind_1(1,1))/2;

if ind_2(1,1)>ind_1(1,1)
    ind_1=ind_2;
end

Phase=ind_1(1,2)*2*pi/ncol;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [ n_points ] = test_sinusoid( mat_edge , Amplitude , Phase , Depth
, delta_h , x )
%Test the sinusoids found previously
%the sinusoids have to be close enough to a certain amount of points
[nline,ncol]=size(mat_edge);
mat=zeros(nline+2*delta_h,ncol);
mat(delta_h+1:delta_h+nline,:)=mat_edge;

f=round(Amplitude*cos(x-Phase)+Depth);
n_points=0;
for j=1:ncol
    n_points=n_points+sum(mat(f(1,j):f(1,j)+2*delta_h,j)==1);
end

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ cell_path_rad ] = find_path( phi_rad , size_win ,n_x_line,
n_factor )
%find a path along two points which are symmetrical compared to the center
of the matrix
%for different angles

n_phi_rad=length(phi_rad);
n_factor_2=(n_factor-1)/2;

%%find the position of the circle's boundary
win_rad=zeros((2*size_win+1)*n_factor,(2*size_win+1)*n_factor,3);
phi_min=0;
phi_max=0;
x_win_mem=size_win*n_factor+n_factor_2;
y_win_mem=0;
for i=1:n_phi_rad
    x_win_rad=(size_win*n_factor+n_factor_2+0.5)*cos(phi_rad(i));
    y_win_rad=(size_win*n_factor+n_factor_2+0.5)*sin(phi_rad(i));

    if x_win_rad>=0
        x_win=size_win*n_factor+n_factor_2;
        while 1
            if x_win_rad>x_win && x_win_rad<x_win+1 || abs(x_win_rad-
x_win)<0.000001
                break
            else
                x_win=x_win-1;
            end
        end
    else
        x_win=size_win*n_factor+n_factor_2;
        while 1
            if x_win_rad>x_win && x_win_rad<x_win+1 || abs(x_win_rad-
x_win)<0.000001
                break
            else
                x_win=x_win-1;
            end
        end
        x_win=x_win+1;
    end
    y_win=-1;
    while 1
        if y_win_rad<y_win+1 && y_win_rad>y_win ||abs(y_win_rad-y_win-
1)<0.000001
            break
        else
            y_win=y_win+1;
        end
    end
    if y_win==-1
        y_win=0;
    end
    if win_rad(size_win*n_factor+n_factor_2+1-
y_win,size_win*n_factor+n_factor_2+1+x_win,1)==1
        if phi_rad(i)<phi_min
            phi_min=phi_rad(i);
        end
        if phi_rad(i)>phi_max
            phi_max=phi_rad(i);
        end
    else
        win_rad(size_win*n_factor+n_factor_2+1-
y_win_mem,size_win*n_factor+n_factor_2+1+x_win_mem,2)=phi_min;
        win_rad(size_win*n_factor+n_factor_2+1-
y_win_mem,size_win*n_factor+n_factor_2+1+x_win_mem,3)=phi_max;
        phi_min=phi_rad(i);
        phi_max=phi_rad(i);
        x_win_mem=x_win;
        y_win_mem=y_win;
    end
    if phi_rad(i)==pi
        win_rad(size_win*n_factor+n_factor_2+1-
y_win_mem,size_win*n_factor+n_factor_2+1+x_win_mem,2)=phi_min;
        win_rad(size_win*n_factor+n_factor_2+1-
y_win_mem,size_win*n_factor+n_factor_2+1+x_win_mem,3)=phi_max;
    end
    win_rad(size_win*n_factor+n_factor_2+1-
y_win,size_win*n_factor+n_factor_2+1+x_win,1)=1;
end

temp_coor_rad=find(win_rad(:,:,1)==1);
n_coor_rad=length(temp_coor_rad);
coor_rad=zeros(n_coor_rad,2);
[coor_rad(:,2),coor_rad(:,1)]=find(win_rad(:,:,1)==1);


%%find the symmetric of the points defined in the upper part of win_rad
%and find all the matrix cell crossed by a line joining the two cells
```

```
cell_path_rad=cell(2,n_coor_rad-1);
for i=1:n_coor_rad
    x1=coor_rad(i,1)-0.5;
    y1=coor_rad(i,2)-0.5;
    x2=((2*size_win+1)*n_factor+1-coor_rad(i,1))-0.5;
    y2=((2*size_win+1)*n_factor+1-coor_rad(i,2))-0.5;

    if x1~=(2*size_win+1)*n_factor-0.5 ||
y1~=size_win*n_factor+n_factor_2+0.5
        if x1~=x2
            m_coeff=(y2-y1)/(x2-x1);
            y0=y1-m_coeff*x1;

            x_line=linspace(x1,x2,n_x_line);
            y_line=m_coeff*x_line+y0;

            ii=coor_rad(i,2)-1;
            jj=coor_rad(i,1)-1;
            n_path=1;
            temp_cell_path_rad=zeros(3*size_win*n_factor+1,2);
            temp_cell_path_rad(n_path,1)=ii;
            temp_cell_path_rad(n_path,2)=jj;
            if x2>x1
                for j=1:n_x_line
                    while 1
                        if x_line(j)<=jj+1
                            break
                        else
                            jj=jj+1;
                        end
                    end
                    while 1
                        if y_line(j)<=ii+1
                            break
                        else
                            ii=ii+1;
                        end
                    end

                    if ii~=temp_cell_path_rad(n_path,1) ||
jj~=temp_cell_path_rad(n_path,2)
                        n_path=n_path+1;
                        temp_cell_path_rad(n_path,1)=ii;
                        temp_cell_path_rad(n_path,2)=jj;
                    end
                end
            else
                for j=1:n_x_line
                    while 1
                        if x_line(j)>=jj
                            break
                        else
                            jj=jj-1;
                        end
                    end
                    while 1
                        if y_line(j)<=ii+1
                            break
                        else
                            ii=ii+1;
                        end
                    end

                    if ii~=temp_cell_path_rad(n_path,1) ||
jj~=temp_cell_path_rad(n_path,2)
                        n_path=n_path+1;
                        temp_cell_path_rad(n_path,1)=ii;
                        temp_cell_path_rad(n_path,2)=jj;
                    end
                end
            end
            temp_cell_path_rad=temp_cell_path_rad+1;
            temp_cell_path_rad(n_path+1:end,:)=[];
        else
            temp_cell_path_rad=zeros(3*size_win*n_factor+1,2);
            for j=1:(2*size_win+1)*n_factor
                temp_cell_path_rad(j,1)=j;
                temp_cell_path_rad(j,2)=coor_rad(i,1);
            end
            temp_cell_path_rad((2*size_win+1)*n_factor+1:end,:)=[];
        end

cell_path_rad{1,i}=sub2ind(size(win_rad(:,:,1)),temp_cell_path_rad(:,1),tem
p_cell_path_rad(:,2));
        cell_path_rad{2,i}(1,1)=win_rad(coor_rad(i,2),coor_rad(i,1),2);
        cell_path_rad{2,i}(2,1)=win_rad(coor_rad(i,2),coor_rad(i,1),3);

cell_path_rad{2,i}(3,1)=(cell_path_rad{2,i}(2,1)+cell_path_rad{2,i}(1,1))/2
;
        if cell_path_rad{2,i}(3,1)>pi/2
            cell_path_rad{2,i}(3,1)=cell_path_rad{2,i}(3,1)-pi;
        end
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Final program to process Statoil data

The program is available at NTNU but if one wants to test this code, one also needs the initial data on which it was tested. Since it is a Statoil property one first need their agreement before considering testing it.

To get the latest version of the code: jscornet@hotmail.fr