Institut Supérieur de l'Aéronautique et de l'Espace

Intelligent Mode Shape Recognition, application to Turbine Blades

Final Project

Internship tutors Dr. Joseph Morlier (ISAE/DMSM)

Student Mauricio Bergh (ITA)

Index

Chapter I

1. Introduction

The final objective of this study is to develop an alternative criterion for Model Shape assurance, as the most currently used one, MAC (classic Modal Assurance Criteria), alongside other calculations useful in Modal Assurance techniques, like the linear correlation coefficient, still lack a more profound analysis of mode shape similarities. The new method is meant to provide a criterion that can detect rotational and translational invariance of these mode shapes, as well as furnish a wider set of factors for comparing them. This new method will use computational Image Processing and Pattern Recognition features so as to obtain a more detailed mode shape description and modelling.

2. Theoretical Background

2.1 Modal analysis

Modal analysis is a process whereby one can describe a structure in terms of its natural characteristics which are the frequency, damping and mode shapes – its dynamic properties.

Consider a certain structure submitted to a determined force. Let us imagine the case in which this force possesses a fixed frequency of oscillation. If we change the rate of oscillation of the force while the peak force value remains the same, measuring the response on the structure (for example, using an accelerometer) we will notice that the displacement amplitude changes as we change the rate of oscillation of the input force. This response amplifies as we apply a force with a rate of oscillation that gets closer and closer to the *natural frequency* (or resonant frequency) of the system and reaches a maximum when the rate of oscillation is at the resonant frequency of the system. **[1]**

Let us consider a freely supported plate submitted to the test described before. Measuring the displacement while increasing the oscillation frequency, we obtain a response spectrum like the one below:

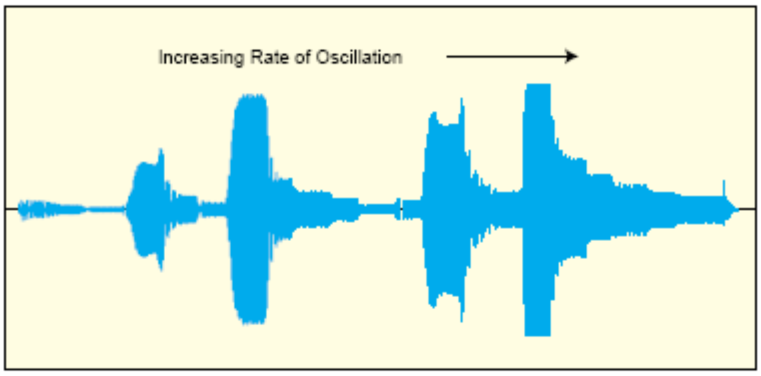


Fig.– Response spectrum

For any structure submitted to an oscillatory force, there will be a specific response spectrum. The notable peaks in the response correspond to the natural frequencies.

The rate of oscillation is increased within a time period, so this is a time based displacement response. Applying a Fourier Transform we can see a frequency based response.

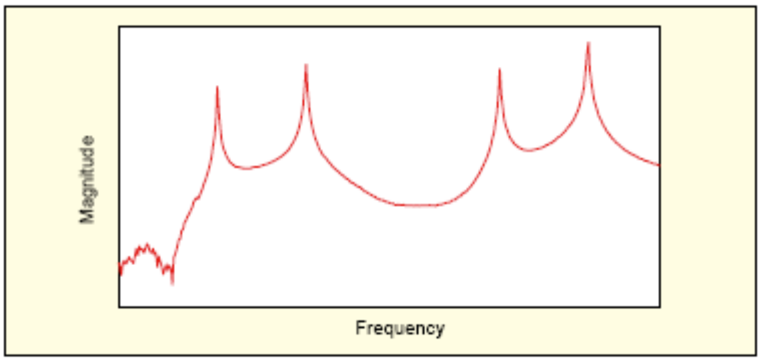


Fig.– Frequency spectrum

In the figure above the natural frequencies become very clear: they are represented by the peaks in which the response has higher magnitude. The deformation patterns at these natural frequencies also take on a variety of different shapes depending on which frequency is used for the excitation force. These deformation patterns are referred to as the *modeshapes* of the structure. (From a mathematical standpoint, there are still factors to be considered, but from a practical standpoint these deformation patterns are close enough execute modeshape analysis)

These natural frequencies and mode shapes occur in all structures that we design. Basically, there are characteristics that depend on the weight and stiffness of my structure which determine where these natural frequencies and mode shapes will exist. A design engineer needs to identify these frequencies and know how they might affect the response of the structure when a force excites it. Understanding the mode shape and how the structure will vibrate when excited helps to design better structures.

So, basically, modal analysis is the study of the natural characteristics of structures. Understanding both the natural frequency and mode shape helps to design my structural system for noise and vibration applications. We use modal analysis to help design all types of structures including automotive structures, aircraft structures, spacecraft and computers. **[1]**

Therefore, the comparison of finite element (FE) modeshape predictions with experimental measurements is an essential step in the model validation process for structural dynamics. A reliable FE model can represent the modeshape pattern before any experiment and make the design more trustworthy.

Currently the most widely used method for comparison between the FE modeling and experimental data is the modal assurance criterion (MAC), which can be interpreted as the cosine of the angle between numerical (FE model) and measured eigenvectors. However, the eigenvectors only contain the displacement of discrete coordinates, so that the MAC index carries no explicit information on shape features. A single numerical value encapsulates all the information contained in the difference between measured and predicted modes, which means that an appreciation of subtle changes in shape, either locally or globally, is clearly unobtainable from the MAC index in the case of large and complicated industrial systems. **[2]**

Alternative techniques for the MAC based on *image processing* (IP) and *pattern recognition* (PR) are already being considered nowadays. A variety of useful shape descriptors (or shape features) may be extracted from mode-shape information. For example, curvature and bending-energy descriptors represent the shapes generally. The *Fourier descriptors* (FD) and *moment descriptors* (MD) are capable of describing the global shape efficiently and accurately. Local shape information may be determined by using *wavelet descriptors* (WD). The comparison of mode shapes, one with another, is achieved by shape classification, involving the assembly of shape-descriptor terms to form the shape *feature vector* (FV). Distances between different FVs may then be used to assess the similarity of different mode shapes. In the case of statistical variability in structures, the modeshape feature vector may be considered as a multi-dimensional stochastic variable. Statistical pattern recognition techniques including Bayesian decision theory and clustering can also be adopted. Results demonstrate the good capability of shape feature vectors to recognise mode shapes both deterministically and stochastically.

2.2 Modal Assurance Criteria

Given that our model represents a realistic structure, we may proceed to the first step in our research, that is, developing an alternative "quality assurance indicator for experimental modal vectors that are estimated from measured frequency response functions" (doc. **[14]**), other than the classical Modal Assurance Criterion (MAC).

The primary method that has historically been used to validate an experimental modal model is the weighted orthogonality check comparing measured modal vectors and an appropriately sized (the size of the square weighting matrix must match the length and spatial dimension of the modal vector) analytical mass or stiffness matrix (weighting matrix). Variations of this process include using analytical modal vectors together with experimental modal vectors and the appropriately sized mass or stiffness matrix. This latter comparison is normally referred to as a pseudo-orthogonality check (POC).

However, when the orthogonality conditions are not satisfied, this result does not indicate where the problem originates. From an experimental point of view, it is important to try to develop methods that indicate confidence that the modal vector is, or is not, part of the problem.

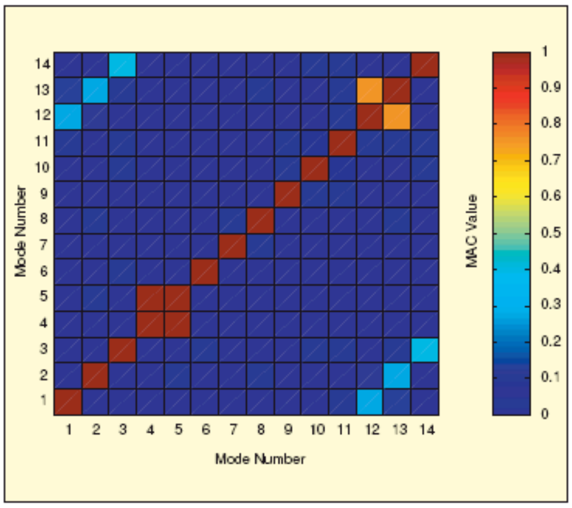
Basically this is the main principle of the ***modal assurance criterion (MAC)***. The main difference is that the MAC has concern in providing a measure of consistency (degree of linearity) between estimates of a modal vector. This provides an additional confidence factor in the evaluation of a modal vector from different excitation (reference) locations or different modal parameter estimation algorithms.

The modal assurance criterion takes on values from zero – representing no consistent correspondence, to one – representing a consistent correspondence. In this manner, if the modal vectors under consideration truly exhibit a consistent, linear relationship, the modal assurance criterion should approach unity and the value of the modal scale factor can be considered reasonable. Note that, unlike the orthogonality calculations, the modal assurance criterion is normalized by the magnitude of the vectors and, thus, is bounded between zero and one.

The modal assurance criterion can only indicate consistency, not validity or orthogonality. If the same errors, random or bias, exist in all modal vector estimates, this is not delineated by the modal assurance criterion. Invalid assumptions are normally the cause of this sort of potential error. Even though the modal assurance criterion is unity, the assumptions involving the system or the modal parameter estimation techniques are not necessarily correct. The assumptions may cause consistent errors in all modal vectors under all test conditions verified by the modal assurance criterion.

For instance, let us examine the result table for MAC for a given experiment, as presented below. It shows that the mode 1 (bottom line), for example, is consistently correspondent to the value for mode 1, obtained from the experiment. In the same way, it shows the reference value for mode 1 (left column) corresponds consistently only to mode 1. Though obvious, this correlation may eventually show startling results, if we notice in the same table, for instance, that modes 4 and 5 are also correspondent.

MAC notorious flaws reside in not showing exactly how are two or more modes correspondent, and not taking into account the validity of one isolated mode. If, let us say, the calculation of two different modes present the same aberration or malfunction, those modes will probably be shown as correspondent in MAC, although both of them contain no useful data. Or else, if two different modes calculation are both incomplete, this will probably lead to the mistaken result that they are not correspondent, when they actually can be.



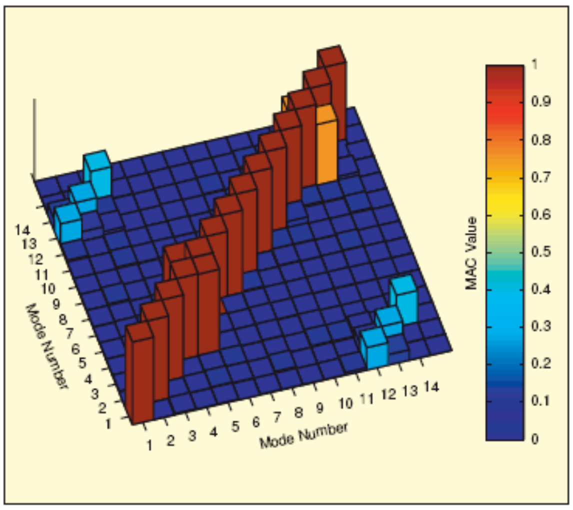


Fig.– Results for classic Model Asssurance Criteria (MAC)

Using *image processing* (IP) and *pattern recognition* (PR), well developed in other disciplines, might enable a more versatile comparison and classification of mode shapes **[15,16]**, which cannot be achieved by the conventional *Modal Assurance Criterion* (MAC) **[17]**. A set of shape features with good discriminative capability may be extracted by IP and PR methods to form a feature vector – *shape descriptor* (SD). Now the similarity/dissimilarity of the shapes can be revealed by the 'distance' of their corresponding SDs in the shape feature space according to appropriate criteria.

The Zernike moment descriptor (ZMD), Fourier descriptor (FD), and wavelet descriptor (WD) are the most popular shape descriptors, having properties that include efficiency of expression, robustness to noise, invariance to geometric transformation and rotation, separation of local and global shape features and computational efficiency. Each one has its own advantages depending on the kind of image being analysed, therefore a perfect, accurate SD would be constituted by these multiple methods sensible combination.

The comparison of mode shapes is readily achieved by assembling the shape features of each mode shape into multi-dimensional shape feature vectors (SFVs) and determining the distances separating them. A *shape feature vector* (SFV) can be formed by assembling the different SDs. The comparison of mode shapes is then transformed to the similarity measurement between the SFVs in the feature space. Dimensionless normalisation for the SFV is necessary before commencing the comparison to avoid the scaling effect.

2.3 Linear Correlation Coefficient

This is the statistical definition of the Pearson product-moment correlation coefficient between two random variables, giving a value between +1 and −1 inclusive. It is widely used in the statistical studies as a measure of the strength of linear dependence between two variables. Pearson's correlation coefficient between two variables is defined as the covariance of the two variables divided by the product of their standard deviations **[19]**. For a pair of 2-dimension matrices, the Pearson's coefficient, also commonly denoted by *r*, will be given by:

$$r = \frac{\sum_m \sum_n (A_{mn} - \bar{A})(B_{mn} - \bar{B})}{\sqrt{\sum_m \sum_n (A_{mn} - \bar{A})^2 \sum_m \sum_n (B_{mn} - \bar{B})^2}}$$

Where $\bar{A}$ and $\bar{B}$ are respectively the means of the values in A and B, and can be automatically calculated by Matlab functions mean2(A) and mean2(B).

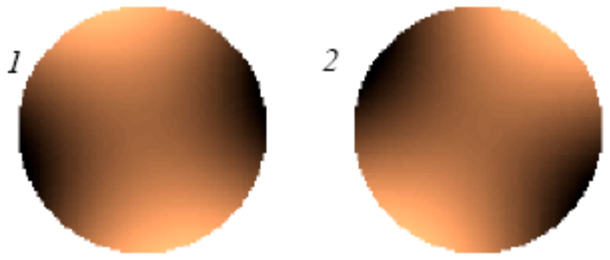
Matlab function corr2 calculates this coefficient, a 2-dimension correlation between two matrices or vectors of the same size. The use of command r = corr2(A,B) results in a scalar, which can be calculated for every mode shape pair (A,B). The correlation coefficient ranges from −1 to 1. A value of 1 means that a linear equation describes the relationship between *A* and *B* perfectly, with all data points lying on a line for which *B* increases as *A* increases; a value of −1 implies that all data points lie on a line for which *B* decreases as *A* increases; value of 0 implies that there is no linear correlation between the variables.

It is important to notice that the calculation of this coefficient itself constitutes a better comparison than the MAC, whose calculation is merely based in vector multiplication. The Pearson's correlation coefficient, or, more exactly, the method using function corr2 as a comparison standard, already takes into account, for instance, the mode shapes whose displacements are in inverse linear correlation. For values equal to -1, we can observe that the displacement are inverse, giving the idea that the mode shapes may be symmetric.

However, like MAC, it is still a purely mathematic operation that does not take into account the distribution and the quality of data given as input, nor the nature of the correlation between two mode shapes.

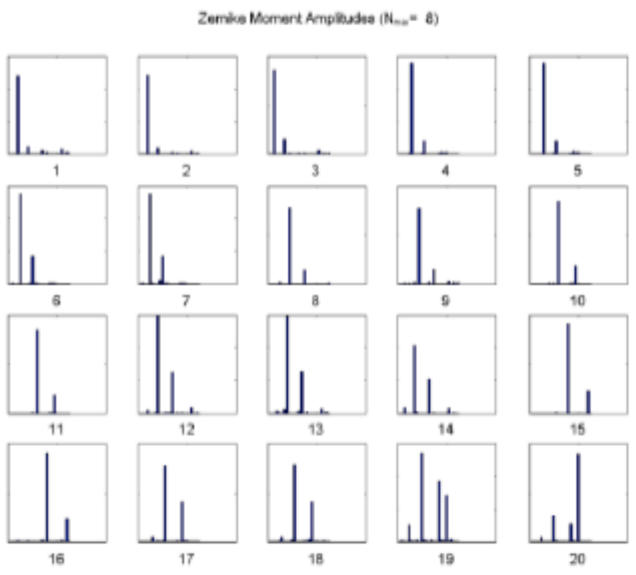
2.4 Example of a New Mode shape Assurance Criterion using ZMD of a Circular Plate

As shown in **[7]**, the free vibration of a circular plate can be modelled by finite elements. Since the circular plate is a perfectly axisymmetric structure double modes are obtained. Observing the modes 1 and 2, it is possible to visualize a correlation between them:



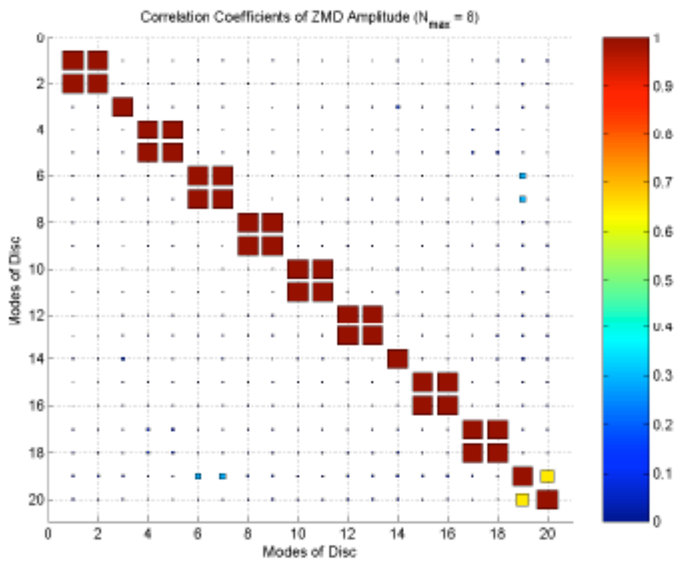
Modeshapes 1 and 2 from experiment

The conventional MAC indicates that they do not correspond totally to one another, but a simple visualization can point out they differ by simple rotation. As the Zernike moment description is invariant to rotation, those two modes have the same ZMD pattern, hence the correlation between them can be mathematically shown. In **[7]**, the Zernike descriptors for each mode and the assurance criterion using them are exemplified:



Zernike moment descriptors for 20 experimental modeshapes

The conventional mode-shape comparison method, MAC, shows nothing about the double modes. The ZMD is applied to the first 20 modes, as only a small number of the lower order ZMDs are needed to represent all the modes. Correlation of mode shapes based on the ZMD amplitudes is shown where the double modes can be clearly recognised.



Zernike-descriptor-based assurance criteria

Slight differences can be noticed between the descriptors for modes 1 and 2, but the main descriptors are practically the same, therefore justifying the correlation indicated in the new criterion. As the circular plate has a simple geometry, the necessary number of used descriptors can be low. For more complicated geometry, more descriptors might be needed and the slight differences may appear more often, as the modeshape also becomes more complex.

This is an example for the case of a singular simple structure, analysed purely by only one of the mentioned methods. It is very important to point out that the Zernike moment description is based in the Zernike moment function, which is defined within the unit circle.

Therefore, the ZMD descriptors can be sharply applied on circular geometries, or geometries that can be reshaped easily into a circle.

This is why the utilisation of multiple methods is very handy when dealing with pattern recognition. Specifically, the ZMD is powerful in discriminating circular and spherical images; the FD is more general and very effective at extracting mode-shape features by virtue of its sinusoidal kernel; the WD shows the capability of distinguishing between local and global features, to cite the most common few methods.

2.5 Image Processing

Image processing (IP) is a set of computational techniques for analyzing, enhancing and reconstructing images. Its main components are: importing, in which an image is captured through scanning or digital photography; analysis and manipulation of the image, accomplished using various specialized software applications; and output (e.g., to a printer). **[3]**

2.6 Pattern Recognition

A typical *pattern recognition* (PR) approach involves the estimation of a series of shape attributes or features with good discriminative capability. The mapping from the space of shapes to the space of shape descriptors should determine the distance between descriptors of two models as a meaningful measure of the underlying similarity of their shapes.

2.7 Fourier Descriptors

Fourier Descriptors (FDs) were originally proposed in 1960 by Cosgriff **[4]**, and thereafter became popular among the pattern recognition community through the papers of Zahn **[5]**, Persoon and Fu **[6]** and are among the most popular shape representation methods for vision and pattern recognition applications. The basic idea underlying this approach consists in representing the shape of interest in terms of a 1D, 2D or even 3D signal. The Fourier transform of this signal is determined and the FDs are calculated for this Fourier representation. As there are plenty of possible Fourier representation definitions for a single signal, FDs might be understood as a class of methods, not a single method. Some properties of the FDs directly follow from the underlying theory of the Fourier transforms and series, for instance, the invariance to geometric transformations.

The FD is based on the frequency components from *Fourier transform* (FT) of the images. According to the well-known theory of the FT, the kernel function of the SD is the complex valued sinusoid,

$$\mathfrak{D}_{\mathcal{F}}(u,v) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} e^{-i2\pi(ux+vy)} I(x,y)\,\mathrm{d}x\mathrm{d}y$$

$D_f(u,v)$ is a continuous function having the same cardinality as I(x,y), and for real applications, this needs to be reduced whilst retaining as much information as possible. Generally the low frequency and higher energy components are sufficient to describe the shape. Thus, for instance, elliptical descriptors based on the FD spectrum are feasible to indicate the distribution of the frequency energy. **[7]**

In reference **[8]** are listed some Matlab routines that can be used for simple implementation of 1D Fourier descriptors, based in the angular functions and in the elliptic function, respectively. The routine used in the development of a Fourier descriptor in Matlab was the 2D elliptic descriptor, which is stated below:

```
%Elliptic Fourier Descriptors
function EllipticDescrp(curve,n,scale)
```

```
%n=num coefficients
%if n=0 then n=m/2
%Scale amplitud output
%Function from image
X=curve(1,:);
Y=curve(2,:);
m=size(X,2);
%Graph of the curve
subplot(3,3,1);
plot(X,Y);
mx=max(max(X),max(Y))+10;
axis([0,mx,0,mx]); %Axis of the graph pf the curve
axis square; %Aspect ratio
%Graph of X
p=0:2*pi/m:2*pi-pi/m; %Parameter
subplot(3,3,2);
plot(p,X);
axis([0,2*pi,0,mx]); %Axis of the graph pf the curve
%Graph of Y
subplot(3,3,3);
plot(p,Y);
axis([0,2*pi,0,mx]); %Axis of the graph pf the curve

%Elliptic Fourier Descriptors
if(n==0) n=floor(m/2); end; %number of coefficients

%Fourier Coefficients
ax=zeros(1,n); bx=zeros(1,n);
ay=zeros(1,n); by=zeros(1,n);
t=2*pi/m;
%Graph coefficient ax
subplot(3,3,4);
bar(ax);
axis([0,n,-scale,scale]);
%Graph coefficient ay
subplot(3,3,5);
bar(ay);
axis([0,n,-scale,scale]);
%Graph coefficient bx
subplot(3,3,6);
bar(bx);
axis([0,n,-scale,scale]);
%Graph coefficient by
subplot(3,3,7);
bar(by);
axis([0,n,-scale,scale]);
%Invariant
CE=zeros(1,n);
for k=1:n
CE(k)=sqrt((ax(k)^2+ay(k)^2)/(ax(1)^2+ay(1)^2))
       +sqrt((bx(k)^2+by(k)^2)/(bx(1)^2+by(1)^2));
end
```

Image reconstruction is straight forward by applying the inverse Fourier transform. Good approximation may be obtained by retaining a sufficient number of higher energy terms.

```
%Graph of Elliptic descriptors
subplot(3,3,8);
bar(CE);
axis([0,n,0,2.2]);
for k=1:n
for i=1:m
ax(k)=ax(k)+X(i)*cos(k*t*(i-1));
bx(k)=bx(k)+X(i)*sin(k*t*(i-1));
ay(k)=ay(k)+Y(i)*cos(k*t*(i-1));
end
by(k)=by(k)+Y(i)*sin(k*t*(i-1);
ax(k)=ax(k)*(2/m);
bx(k)=bx(k)*(2/m);
ay(k)=ay(k)*(2/m);
by(k)=by(k)*(2/m);
end
%.................................................
```

In order to show the importance of the lower and higher energy terms, it is interesting that a step-by-step image reconstruction is shown, which will also be considered in this work.

2.8 Moment Descriptors

One of the simplest and most intuitive shape descriptors is the geometric moment. Given a two-dimensional continuous image I(x,y) the geometric moments $m_{p,q}$ of order (p+q) are defined as **[2]**

$$m_{p,q} = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} x^p y^q I(x,y)\, \mathrm{d}x\, \mathrm{d}y, \quad p,q = 0,1,2\ldots \tag{1}$$

For an N_x X N_y digital image the integral is replaced by summation:

$$m_{p,q} = \sum_{x=1}^{N_x} \sum_{y=1}^{N_y} x^p y^q I(x,y) \tag{2}$$

As a given image will have a unique geometric moment sequence, a sequence can be considered as a set of shape descriptors to distinguish different shapes.

However, the basis of geometric moments is not orthogonal, hence the moment sequence includes redundant information to a high degree and the high-order moments are very sensitive to noise. It also makes the original shape generally more difficult to recover from a truncated set of moment descriptors. This argument justifies the use of bases that are less intuitive than the geometric moment, but have much superior properties for discrimination of images (or shapes), for reconstruction and for comparison between one image and another. One such descriptor is the *Zernike moment descriptor* (ZMD).

2.9 Zernike Moment descriptor

The *Zernike moment* (ZM) is based on a complete set of orthogonal polynomials, rather than the traditional algebraic polynomials used in the geometric moments, defined over a circle of unit radius – known as the Zernike polynomials. The Zernike moment is one of the most important region-based shape descriptors because of its outstanding properties resulting from the orthogonality of the Zernike polynomials. These properties incluse:

- Minimum information redundancy, obtained while expressing an image as a set of mutually independent descriptors;
- Contribution of each order of moment to the image reconstruction can be separated, so that the process of regaining the original image is much easier than by geometric moment descriptors **[9]**.
- Rotational invariance **[10, 11]**, i.e., rotating an image does not change the magnitudes of its Zernike moments.
- Robustness to noise **[9]** and effectiveness, hence a small number of Zernike moments are usually sufficient for shape reconstruction.

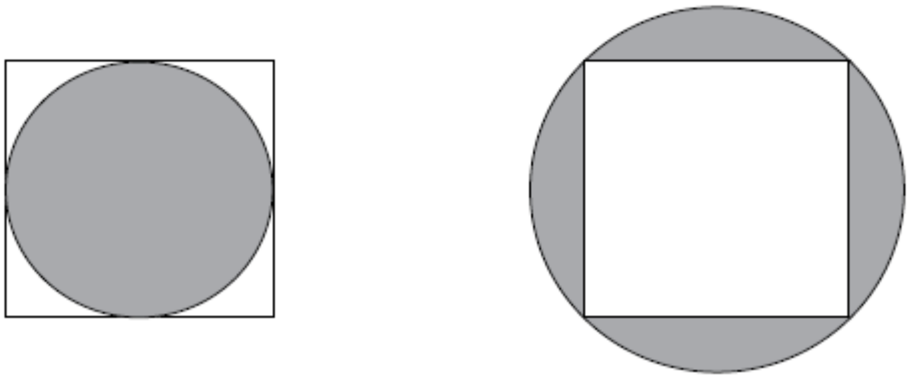
The complete set of orthogonal complex polynomials over a circle of unit radius introduced by Zernike [16] can be expressed as

$$V_{n,m}(x,y) = V_{n,m}(\rho,\vartheta) = R_{n,m}(\rho)e^{im\vartheta} \tag{3}$$

Where $i = \sqrt{-1}$,n non-negative integer, representing the order of the radial polynomial, m positive and negative integers subject to constraints n-$|m|$ even, $|m| \leq$ n, representing the repetition of the azimuthal angle, ρ length of vector from the origin to (x,y), θ the azimuthal angle between vector ρ and the x-axis in the counter clockwise direction, $R_{n,m}$ radial polynomial defined as:

$$R_{n,m} = \sum_{s=0}^{(n-|m|)/2} (-1)^s \cdot \frac{(n-s)!}{s!\left(\frac{n+|m|}{2}-s\right)!\left(\frac{n-|m|}{2}-s\right)!}\rho^{n-2s} \tag{4}$$

These polynomials are orthogonal within the unit circle, so, if necessary, the analysed shape (the area of interest) has to be remapped to be of this size before calculation of its moments. There resides the ZMD biggest flaw: it implies difficulty in mapping a unit circle to a Cartesian grid. Some approaches can be done with the purpose of reshaping the original image, and make it fit for ZMD utilisation. As illustrated below, the image can be reshaped so that circle can be within the area of interest, losing some rarely-used corner information (a), or around the area of interest, which then covers areas where there is no information, but ensures that all the information within the area of interest is included (b). **[8]**


(a) Unit circle within area of interest
(b) Area of interest within unit circle

Frequency spectrum

For simpler geometries there is also the possibility to a reshaping method that can transform a convex polygon into a unit circle, described in **[2]**.

Although very useful, the utilisation of ZMDs is restricted to the structure capability of fitting into the unit circle, after adequate mathematical manipulation. If the process of reshaping is too complex, the method effectiveness, minimal redundancy and reconstruction ease, its main advantages, can be damaged.

2.10 Wavelet Descriptors

Wavelet transformation represents an image in terms of the superposition of wavelet with different scale levels and positions. The wavelet, having better time-frequency resolution than a Fourier transform **[12]**, can be expressed as

$$\psi_{b_x,b_y,a} = \frac{1}{a}\psi\left(\frac{x-b_x}{a},\frac{y-b_y}{a}\right) \tag{5}$$

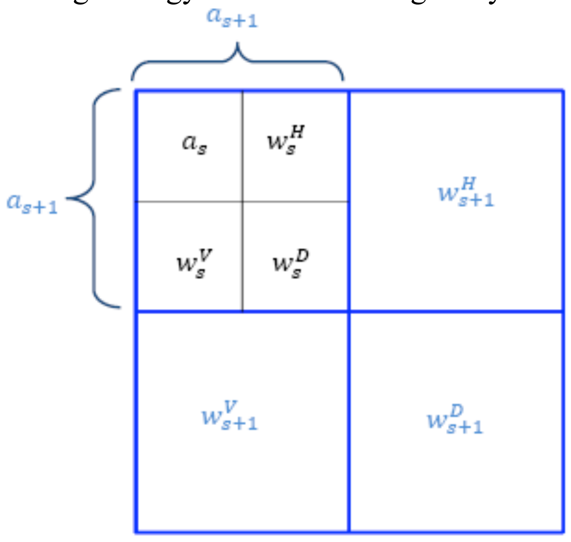
Where $a \in \Re^{+}$ is the dilating scale parameter, $(b_x,b_y) \in \Re^{2}$ are the translation parameters and $\psi_{bx,by,a}$ is the translated and dilated version of the mother wavelet ψ (x,y). The normalisation factor 1/a is included so that $\left\|\psi_{bx,by,a}\right\| = \left\|\psi\right\|$. Depending on the applications, these parameters can be chosen as either continuous or discrete values. The definition of CWT can be expressed as an inner product of the wavelet and the image, **[7]**

$$\mathfrak{D}_W\left(a, b_x, b_y\right) = \langle \psi_{b_x,b_y,a}, I(x,y) \rangle = \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \psi^{*}_{b_x,b_y,a}(x,y) I(x,y) \mathrm{d}x\mathrm{d}y \tag{6}$$

For the wavelet to be oscillatory with a null DC component, the mother wavelet must satisfy,

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \psi(x,y) \mathrm{d}x\mathrm{d}y = 0 \tag{7}$$

Considering in a simple approach the wavelet decomposition of a two-dimensional mode shape, the two-dimensional discrete wavelet transform for an image can be obtained by implementing the one-dimensional algorithm horizontally and then vertically. The outputs from each step of decomposition are the sub-images of one approximation at coarser resolution and three sub-images of detail in horizontal, vertical and diagonal directions as illustrated below**.** Thus, the comparison between images can now be carried out between the sub-images at different resolutions. In additional, these coefficients can be used as the WDs and the average energy of each sub-image may be used to form the SFV. **[7]**



The approximation sub-image a_{s+1} at scale-level $s+1$ as shown at the left can be decomposed into four sub-images at coarser resolution with scale-level s as:

Approximation:	a_s
Horizontal details:	w_s^H
Vertical details:	w_s^V
Diagonal details:	w_s^D

Frequency spectrum

3. Development of Fourier reconstruction Matlab routine

At a first moment, the main efforts were concentrated in building a Matlab routine for image processing and pattern recognition using the Fourier Descriptor method. If possible, try to develop a routine to use two or more methods alongside, or reunite two different routines.

After completing this first task, the main concern was implementing the developed routine using simply supported plate data as input. Subsequently the routine may be implemented in a damaged plate data and in the finite element turbine blade model.

It was possible to obtain 15 mode shape images from the plate data set. The results obtained and the Matlab functions development stages are presented as follows.

3.1 Testing of Fourier Descriptor with Single Plate Data

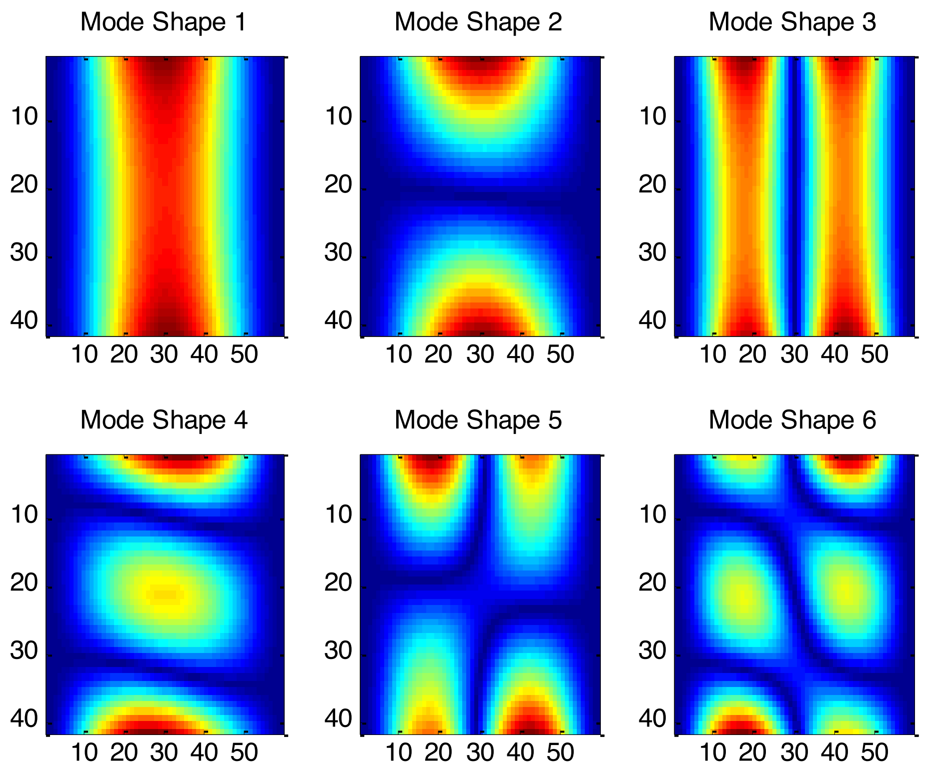
3.1.1 Nodal line detection

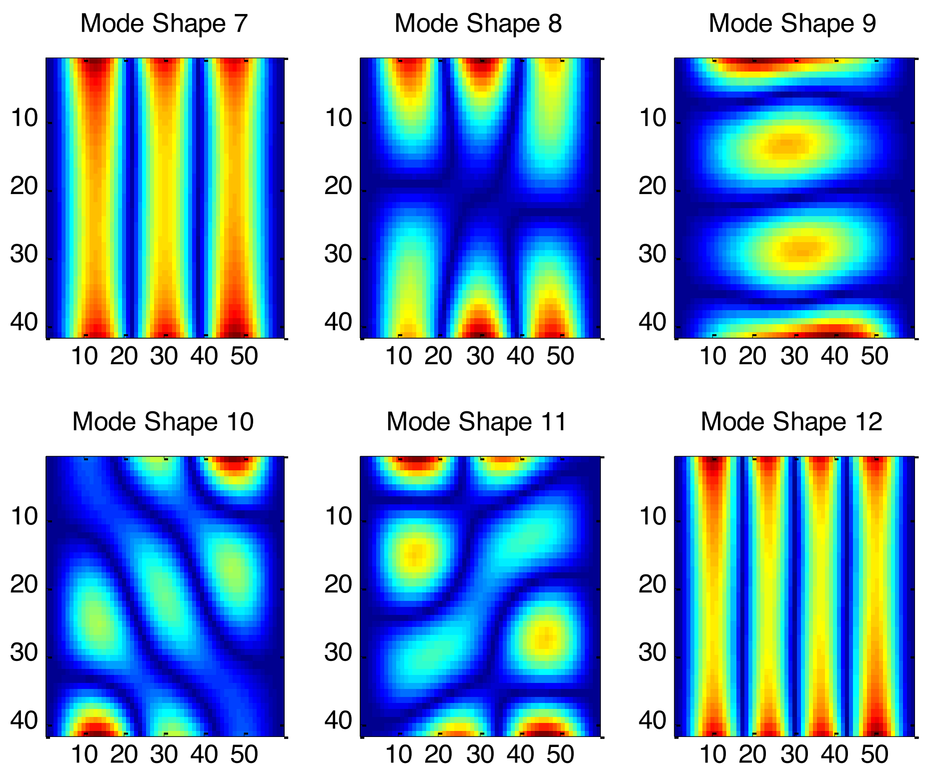
- How to use function `nodal_line_identification.m` :

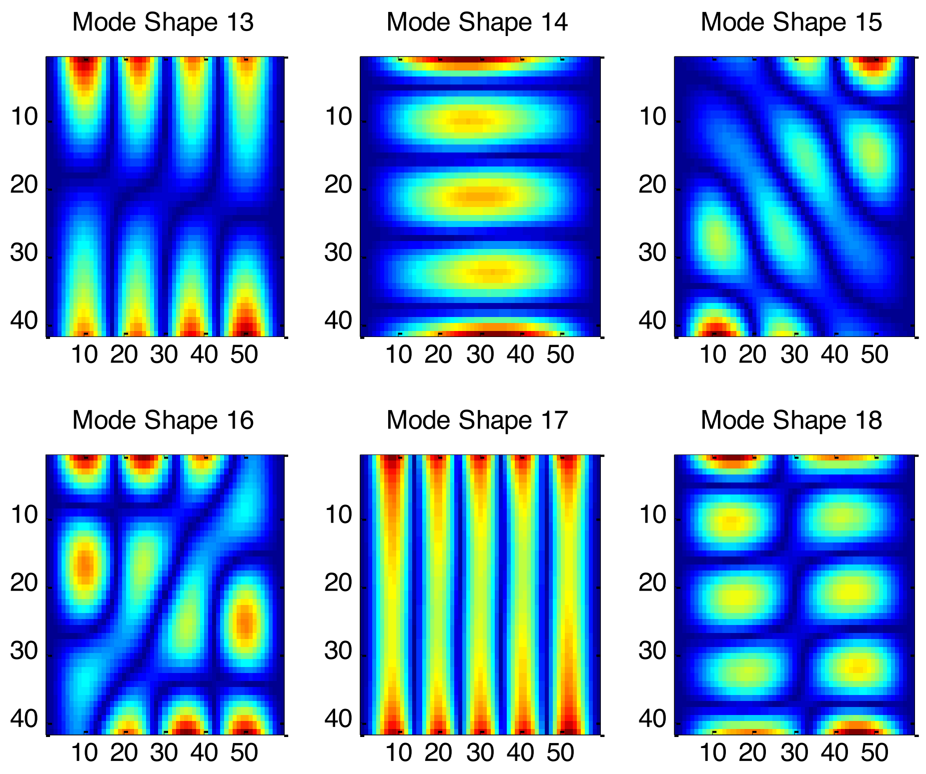
The function `nodal_line_identification.m` was developed with the intention to display the 18 plate mode shapes available, showing, at first, the 18 mode shapes altogether, and then each one with its own nodal region presented. The data were extracted from .rpt files, each file named mode#.rpt (where # represents the respective mode shape number) and containing the data due to one only mode shape. The organisation of each file is explained as comments in the function commands. The function code is as stated as follows:

```
%function for nodal line identification
clear all; close all;

%in each file mode.rpt there are 3 columns :
% column 1 : n°nodes
% column 2 : U magnitude
% column 3 : U3 (dep vertical)
% plat dimensions : 236 * 291 mm
% material : density =1550kg/m3
% E1=110.3GPa, E2=E3=7.69GPa, G12=G13=4.75GPa, G23=2.746GPa

for ii=1:18
RPT=sprintf('mode%d.rpt',ii) ;
mode=dlmread(RPT);
if (ii==1),data(:,ii)=mode(:,1); end
data(:,2*ii:2*ii+1)=mode(:,2:3);
if mod(ii,6)==0, tab=6; else tab=mod(ii,6);end
if (mod(ii,6)==1), figure;end

subplot(2,3,tab)

imagesc((reshape(data(:,2*ii),59,[]))')
title(sprintf('Mode Shape %d',ii))

if (tab==6), pause;end

end

%frequencies for each mode
%mode1: f=201.34 Hz %mode2: f=268.93 Hz
%mode3: f=502.92 Hz %mode4: f=554.49 Hz
%mode5: f=652.11 Hz %mode6: f=932.23 Hz
%mode7: f=975.54 Hz %mode8: f=1085.4 Hz
%mode9: f=1192 Hz %mode10: f=1419.4 Hz
% mode 11 : f=1544.7 Hz; %mode 12 : f=1662.7 Hz
% mode 13 : f=1787.5 Hz;  %mode 14 : f=1895.2 Hz % mode 15 : f=2030.5 Hz


% dispose all values on a «data » table, simpler to manipulate later
%data(:,1)=mode1(:,1);
%data(:,2:3)=mode1(:,2:3);  ... data(:,30:31)=mode15(:,2:3);

%mode1, Umagnitude
%Umag_mode1=(reshape(data(:,2),59,[]))';
%mode2, Umagnitude
%Umag_mode2=(reshape(data(:,4),59,[]))';
%...
%mode 15, Umagnitude
%Umag_mode15=(reshape(data(:,30),59,[]))';

for jj=1:18

      n_line((reshape(data(:,2*jj),59,[]))',jj);
end
```

Other comments regarding to the frequencies obtained from each mode shape and way each mode shape had to be reshaped to fit the image format were also left in the function. The first command sequence reads the data available in the files and displays the image extracted from each mode shape, displaying all 18 mode shapes together.
After you execute, the program will automatically generate a picture representation from the 18 data sets available in the directory (the eighteen .rpt files that have to be seen in the current directory). Three windows are going to be created, each one containing five modes and being shown in the screen by pressing any key in the keyboard. They will be presented as follows:





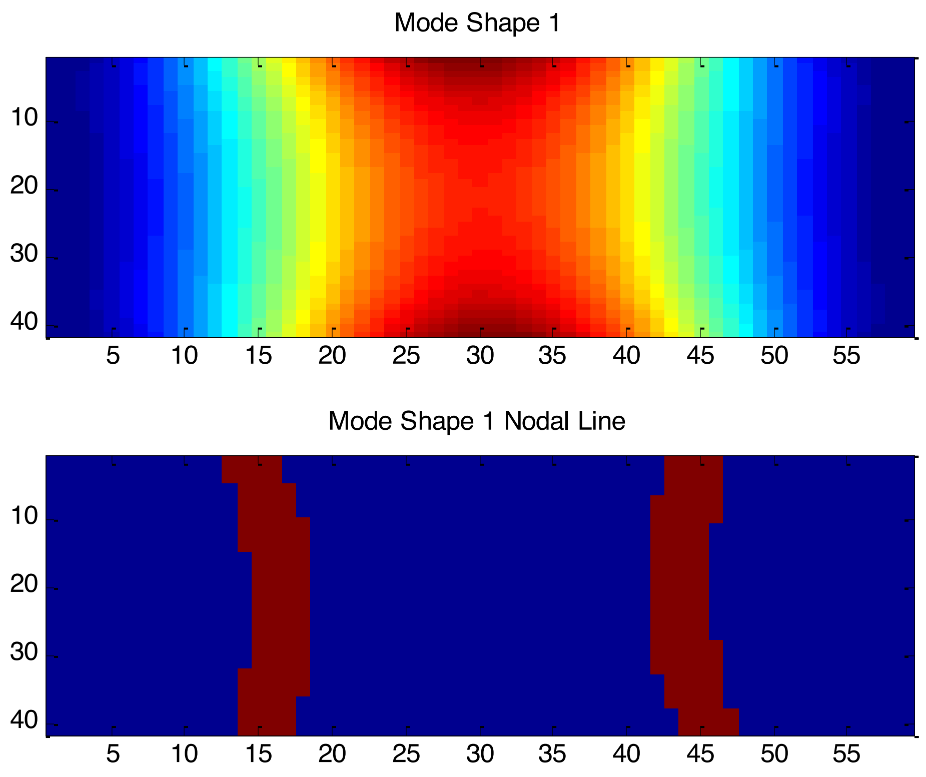


The final command lines will set the execution of function `n_line`. This function detects the region in the image closest to the nodal line. It is possible to set a threshold to define this nodal line width, which is made by setting the values of mean value "average" and interval range "epsilon" (here set to 0.5 and 0.1, respectively) to values of interest.

```
function n_line(imagemode,mode)

nomsave=sprintf('mode%d.bmp',mode) ;

        gimg=imagemode;

        %create nodal line by thresholding epsilon
        epsilon=0.1;average=0.5;
        nodalline=(gimg<epsilon+average)&(gimg>-epsilon+average);

%add complement
imwrite(imcomplement(nodalline),nomsave);
% level = graythresh(abs(gimg));
% nodalline = im2bw(abs(gimg),level);

figure;
subplot(211);imagesc(gimg); title(sprintf('Mode Shape %d',mode));
subplot(212);imagesc(nodalline);title(sprintf('Mode Shape %d Nodal Line',mode));
pause;
```

Define nodal line range

As the program represents all fifteen mode shapes and its nodal lines, it will lead to fifteen more graphics. As before, each graphic is shown after pressing any key to make the program exit the *pause* command. A representation of the plotting for mode 1 is shown below, as an example for the kind of result generated for each mode:



Function also saves each nodal line image (shown in red) as a bitmap file, there can be used later for image description. They are all named `'mode%d.bmp'` (where `%d` represents the respective mode shape number).

3.1.2 Image description

The nodal lines extracted from the mode shapes using the previous Matlab functions will be submitted to the Fourier description method. The image description of the nodal line, extracted from mode shape 1 data, is done by function demoFDplate.m. The function code is as stated below:

```
clear all;close all;
%inputimage=im2bw(imread('test4.bmp'));
inputimage=imread('mode1.bmp'); imagesc(inputimage);
s=size(inputimage);
%add 2 lines and columns to avoid boundary effects
imageC=ones(s(1)+4,s(2)+4);
 imageC(3:end-2,3:end-2)=inputimage;
%imageC= IIR(imageC,2);
%enhance image resolution coef res
res=5;
imageC = im2bw(interp2(imageC,res));
imagesc(imageC); title('Original image');

curve=CompleteFD(imageC);
EllipticDescrp(curve,50);
```

1 Change here the mode you want to analyse

2 Choose the resolution you want in the shown

3 Choose how many descriptors you want to use

Executing this code will provide a Fourier description from the mode shape given by the bitmap file name given in position 1. The number in position 2 allow you alter the resolution of the given image, if necessary (a resolution of 5 is already greatly acceptable), also knowing that a bigger resolution will improve the obtained image quality, but will take more computation time. Finally, you can choose the quantity of descriptors to be used in position 3. A number of descriptors equal to 50 leads to a very good reconstruction, but also takes computation time to be developed. The usually adopted number is 25, for reasonable results.

Notice that the function EllipticDescrp.m is used. This function is based in the function given by ref. **[8]**. The results obtained from this function and CompleteFD.m are distinct and interdependent.

The function CompleteFD.m commands were divided for better explanation as follows:

```
%Gradual elliptic Fourier Descriptor+Contour
function curve=CompleteFD(Input)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function from image

global inputimage
inputimage=Input;
curve=[];
```

```
while(checkimg(inputimage))
[extractcontour,extractimage]=Contour2(inputimage);
curve=[curve extractcontour];

  X=curve(1,:);
  Y=curve(2,:);
  if (size(X,2)~=0)
      for index=1:size(X,2)
          inputimage(Y(index),X(index))=1;
      end
  end
inputimage=extractimage;

end

% Graph of the curve
  X=curve(1,:);
  Y=curve(2,:);

    figure;
    plot(X,Y);
    mx=max(max(X),max(Y))+10;
    axis([0,mx,0,mx]); % Axis of the graph pf the curve
    axis square;                  % Aspect ratio
    title('Obtained Image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This first data set is destined to ensure that all curves in the image will be analyzed. While the subfunction `checkimg` indicates that there are still curves in the original image whose Fourier transforms were not taken into account in the obtained analyzed image, the function continues to search for curve contours in the input data set, which is erased after the contour total extraction.

The function Contour2 extracts the contour from each curve present in the input data set at a time. For each curve, the obtained contour is stored in `extractcontour` and the remaining image (the input image without the already extracted contours) is stored in `extractimage`. The entirety of obtained contours is stored in `curve`, and the condition for exiting the loop is simple: when there is no non-analyzed pixel remaining in the original image `inputimage`, and all contours of interest are in already stored in `curve`, then the `while` -loop ends. The final commands are simply to display the original image and the obtained contours.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 function f=checkimg(inputimage)

[rows,columns]=size(inputimage);
f=0;
for x=2:columns-1
   for y=2:rows-1
       if (inputimage(y,x)==0),f=1;end
   end
end

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

The subfunction `checkimg` merely checks if there is any non-analyzed pixel in the image to be analyzed. If this is true, i.e., there is any non-analyzed pixel, the subfunction returns the value 1, if not, it returns the value 0.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Contour extraction form a binary image

function [outputcontour,outputimage] = Contour2(inputimage)

global border
    %Image size
    [rows,columns]=size(inputimage);
    outputimage=inputimage;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Image border identification
%the counter b indicates the number of neighbor pixels that also belong
%to the image.

    % num neighbours #Black~=8
    border=zeros(rows,columns);

    for x=2:columns-1
      for y=2:rows-1
        if inputimage(y,x)==0
           b=0;
           for Nx=x-1:x+1
             for Ny=y-1:y+1
                 if(x~=Nx || y~=Ny)
                   if inputimage(Ny,Nx)==0
                      b=b+1;
                   end
                 end
             end
           end
           if(b~=8),border(y,x)=1;end
        end
      end
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Erase pixels that do not belong to border.
for x=2:columns-1
      for y=2:rows-1
          if (border(y,x)~=1)
              outputimage(y,x)=1;
          end
      end
end

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% follow
outputcontour=[];
```

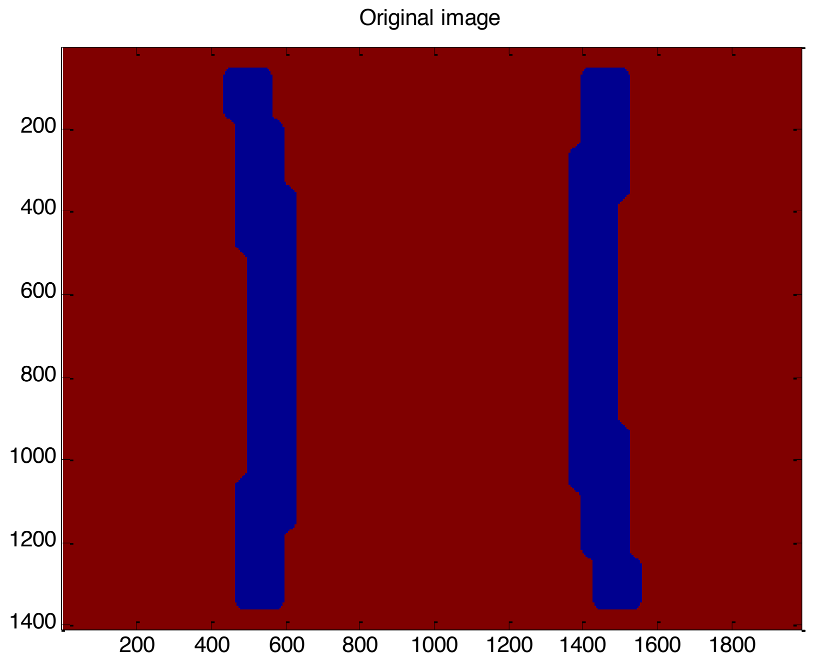
```
   % Condition: Do the following calculation only for closed curves

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %The following lines are developed to make the image borders
    %thinner (thickness=1 pixel).
    %Hence, it is not necessary for open curves, i.e., curves
    %that are already 1-pixel-thick.
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Thin borders
%
%       delete pixel if does not break a chain
%               N8: neighbours !=0
%
%       (x,y):
%                N=Card(N8(x,y))
%                                B=Sum(Card(N8(P))   for P in N8(x,y)
%
%               if((n-1)*2==B) not break a chain
%

    for x=2:columns-1
      for y=2:rows-1
          N=0; B=0;   % num neighbours
         if border(y,x)>0
             for Nx=x-1:x+1   % 8 Neigbour
                for Ny=y-1:y+1
                    if  ((Nx~=x  || Ny~=y)  && border(Ny,Nx)>0)
                      N=N+1;
                        for NNx=x-1:x+1   % 8 Neigbour
                            for NNy=y-1:y+1
                                if  ((NNx~=x  || NNy~=y)&&
border(NNy,NNx)>0)
                                if  ((NNx~=Nx  || NNy~=Ny))
                                  if( abs(NNx-Nx)<2  && abs(NNy-Ny)<2)
                                    B=B+1;
                                  end
                                end
                              end
                          end
                        end

                    end
                end
             end

          if((N-1)*2==B)
                border(y,x)=0;
            end
         end
      end
  end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
for x=2:columns-1
      for y=2:rows-1
          if border(y,x)~=1,outputimage(y,x)=1;end
      end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This section is used for the image contour detection. Once one curve is detected, every pixel inside it is erased. Then, the curves contours are extracted. A final verification is made to ensure that the contours are one-pixel-thick.

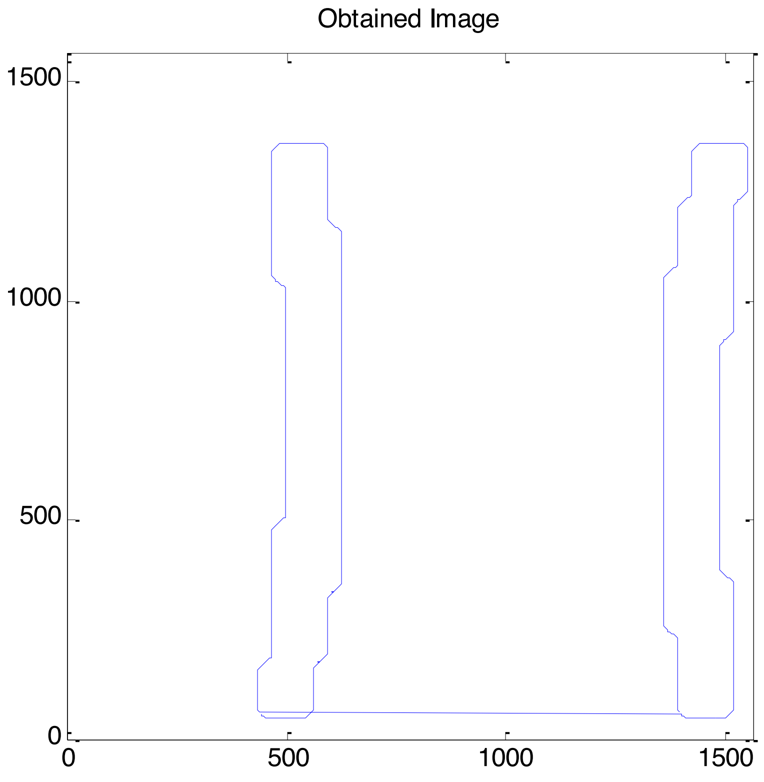
```
%Search starting point
    dmin=rows+columns;d=0;
    strtx=1; strty=1;
    for x=2:columns-1
      for y=2:rows-1

            if (outputimage(y,x)==0 && border(y,x)==1)


                d=y+x;


                if(d<dmin)


                dmin=d;


                strtx=x; strty=y;
                %ContX=x;ContY=y;

                end


            end
      end
    end

if d~=0
     %insert initial point
      sx=strtx; sy=strty;
      outputcontour=[outputcontour [sx;sy]];
      border(sy,sx)=0; %point in the output contour

     % next point
     cx=0;cy=0;
     for x=sx-1:sx+1;
        for y=sy-1:sy+1
            if(border(y,x)~=0)
                cx=x; cy=y;
            end
        end
     end

    % border following
    while( (cx~=strtx || cy~=strty) )
         if (cx~=0 && cy~=0),outputcontour=[outputcontour [cx;cy]];
%store current point
         sx=cx; sy=cy;
         border(sy,sx)=0; %point in the output contour
         else
         border(sy,sx)=0;
    end

         % next point
         n=size(outputcontour,2); % num pts
         stp=0;
         for x=sx-1:sx+1
             for y=sy-1:sy+1
                 if( (x~=sx || y~=sy) && ~stp)
                     if(n>3 && x==strtx && y==strty) % arrive to the end
                     cx=x; cy=y;
                     stp=1;              % stop cicle
                     elseif(border(y,x)~=0)
                     cx=x; cy=y;
                     end
                 end
             end
         end

      end %while ( (cx~=strtx || cy~=strty) )

end

  X=outputcontour(1,:);
  Y=outputcontour(2,:);
  if (size(X,2)~=0)
      for index=1:size(X,2)
          outputimage(Y(index),X(index))=1;
          border(Y(index),X(index))=0;
      end
  end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End of calculation for closed curves
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This final section is destined to ordinate the pixels obtained in the extracted contour. After choosing a starting pixel (in this case, the pixel with the smallest coordinates), the function searches for a pixel next to it that belongs to the contour, proceeding to finding the pixel next to this second, and so on. When the function returns to the first pixel, the loop is ended. Therefore, the function is useful for multiple closed curved analyses.

The results obtained are shown as follows. The description of an image with multiple curves has reasonable quality, no matter how many curves it has.



A representation from the black and white picture to be analysed.



A representation from the previous picture's contour.

These are the results generated by CompleteFD.m. They will be used to generate the results of function `EllipticDescrp.m`, which calculates the Elliptic Fourier function descriptors. The commands will also be divided in 4 sets, for better explanation.

```
% Elliptic Fourier Descriptors
function  EllipticDescrp(Curve,n)  % n= num coefficients
                                        % if n=0 then n=m/2
                                        % Scale amplitud output
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function from image

curve=double(Curve);
X=curve(1,:);
Y=curve(2,:);
m=size(X,2);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graph of the curve
figure;
    subplot(3,2,1);                              % The plot
    plot(X,Y);
    mx=max(max(X),max(Y))+10;
    axis([0,mx,0,mx]); % Axis of the graph pf the curve
    axis square;                     % Aspect ratio
    title('Original Image')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graph of X
  p=0:2*pi/m:2*pi-pi/m;             % Parameter
    subplot(3,2,2);                              % The plot
    plot(p,X);
    axis([0,2*pi,0,mx]); % Axis of the graph pf the curve
    title('Fourier Transform')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Graph of Y
    subplot(3,2,3);                              % The plot
    plot(p,Y);
    axis([0,2*pi,0,mx]); % Axis of the graph pf the curve
    title('Normalized Fourier Transform')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

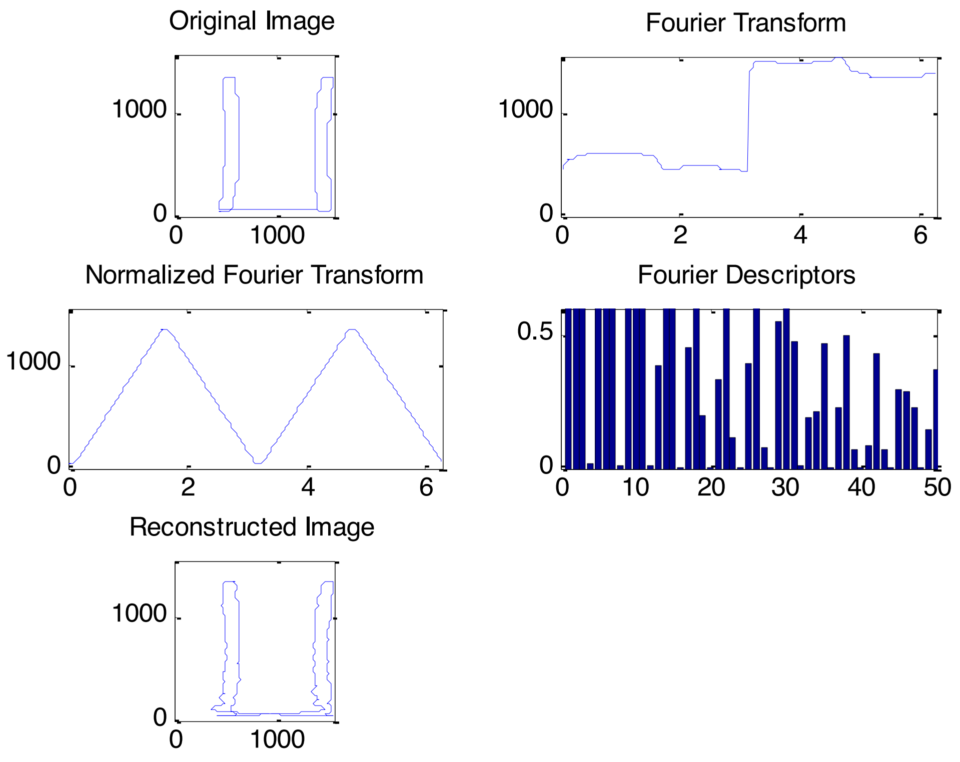
This first command set is for calculation of the Fourier transform from the obtained contour. The original contour and normalized Fourier transform are also displayed, for better visualization of the results.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Elliptic Fourier Descriptors

  if(n==0), n=floor(m/2); end; % num coefficients

    ax=zeros(1,n);                % Fourier Coefficients
    bx=zeros(1,n);
    ay=zeros(1,n);
    by=zeros(1,n);

    t=2*pi/m;

    for k=1:n
    for i=1:m
        ax(k)=ax(k)+X(i)*cos(k*t*(i-1));
        bx(k)=bx(k)+X(i)*sin(k*t*(i-1));
        ay(k)=ay(k)+Y(i)*cos(k*t*(i-1));
        by(k)=by(k)+Y(i)*sin(k*t*(i-1));
      end
    ax(k)=ax(k)*(2/m);
    bx(k)=bx(k)*(2/m);
    ay(k)=ay(k)*(2/m);
    by(k)=by(k)*(2/m);
    end

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

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Invariant (the final descriptors)
 CE=zeros(1,n);

 for k=1:n
    CE(k)=sqrt((ax(k)^2+ay(k)^2)/(ax(1)^2+ay(1)^2))+
            sqrt((bx(k)^2+by(k)^2)/(bx(1)^2+by(1)^2));
    end

subplot(3,2,4);
  bar(CE);
  axis([0,n,0,.6]);
  title('Fourier Descriptors');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

This second command set is for calculation of the descriptors. Each descriptor is calculated using the Elliptic function definition **[8]**, and the fifty calculated descriptors are displayed in absolute value (all positive).

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Reconstruction (total, for the input number of descriptors)

    ax0=0;
    ay0=0;
    for i=1:m
        ax0=ax0+X(i);
        ay0=ay0+Y(i);
  end
  ax0=double(ax0/m);
  ay0=double(ay0/m);

 RX=ones(1,m)*ax0;
 RY=ones(1,m)*ay0;

 for i=1:m
    for k=1:n
        RX(i)=RX(i)+ax(k)*cos(k*t*(i-1))+bx(k)*sin(k*t*(i-1));
        RY(i)=RY(i)+ay(k)*cos(k*t*(i-1))+by(k)*sin(k*t*(i-1));
      end
    end

 subplot(3,2,5);
 plot(RX,RY);
 mx=max(max(RX),max(RY))+10;
 axis([0,mx,0,mx]);             % Axis of the graph pf the curve
 axis square;                   % Aspect ratio
 title('Reconstructed Image');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The third command set is destined to the reconstruction using the fifty calculated descriptors. By calculating the curve values for X and Y, using as starting point only the obtained descriptors, a new curve is obtained, that is very similar to the original one. The descriptors magnitude and the reconstruction accuracy, for the use of 50 descriptors, can be observed by displaying the results calculated so far:



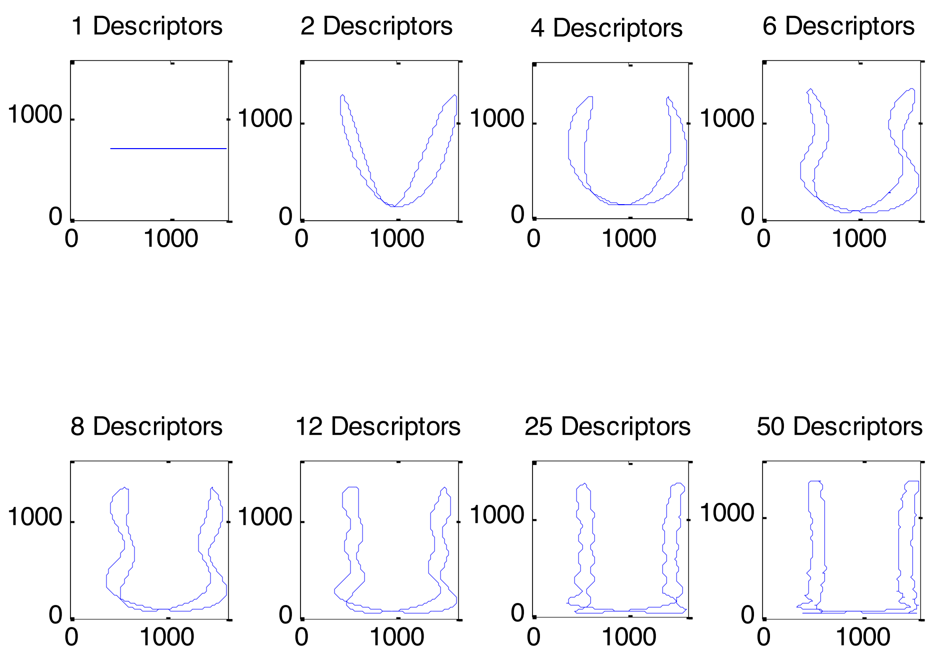
The reconstructed image already gives a good idea of the original image. For better accuracy, we can use a greater descriptor quantity, always keeping in mind that the time to calculate the reconstructed image will also be greater.

The final command set is the one that generates a visualization of the image being gradually reconstructed. It is also the one that takes the greatest computational effort, which spends more time in calculation and displaying. If more improvement in speed is wanted from the routine, this is the section to work on, while it is the one in which computational time is most critical.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Elliptic Fourier Descriptors calculation for gradual reconstruction
figure;
for R=1:8

  %if(n==0), n=floor(m/2); end; % num coefficients

     switch R
        case 1
            coeff=1;
        case 2
            coeff=2;
        case 3
            coeff=4;
        case 4
            coeff=6;
        case 5
            coeff=8;
        case 6
            coeff=12;
```

```
       case 7
            coeff=25;
        case 8
            coeff=50;

    end
    ax=zeros(1,coeff);                % Fourier Coefficients
    bx=zeros(1,coeff);
    ay=zeros(1,coeff);
    by=zeros(1,coeff);

    t=2*pi/m;

    for k=1:coeff
    for i=1:m
        ax(k)=ax(k)+X(i)*cos(k*t*(i-1));
        bx(k)=bx(k)+X(i)*sin(k*t*(i-1));
        ay(k)=ay(k)+Y(i)*cos(k*t*(i-1));
        by(k)=by(k)+Y(i)*sin(k*t*(i-1));
      end
    ax(k)=ax(k)*(2/m);
    bx(k)=bx(k)*(2/m);
    ay(k)=ay(k)*(2/m);
    by(k)=by(k)*(2/m);
    end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Gradual Reconstruction

    ax0=0;
    ay0=0;
    for i=1:m
        ax0=ax0+X(i);
        ay0=ay0+Y(i);
  end
  ax0=double(ax0/m);
  ay0=double(ay0/m);

 RX=ones(1,m)*ax0;
 RY=ones(1,m)*ay0;

 resborder=[];
 for i=1:m
    for k=1:coeff
        RX(i)=RX(i)+ax(k)*cos(k*t*(i-1))+bx(k)*sin(k*t*(i-1));
        RY(i)=RY(i)+ay(k)*cos(k*t*(i-1))+by(k)*sin(k*t*(i-1));
    end
      if (round(RY(i))>0 &&
round(RX(i))>0),resborder(round(RY(i)),round(RX(i)))=1;end
    end

 subplot(2,4,R);
 plot(RX,RY);
 mx=max(max(RX),max(RY))+10;
 axis([0,mx,0,mx]);              % Axis of the graph pf the curve
 axis square;                    % Aspect ratio
 title(sprintf('%d Descriptors',coeff));
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

The function is designed so that the reconstruction is gradually shown for 1, 2, 4, 6, 8, 12, 25 and 50 descriptors respectively. The results obtained are:



The results prove that the bigger the number of descriptors used in the reconstruction, the bigger the accuracy from the reconstruction. And, depending on the research purpose, a convenient number of descriptors may be more suitable. For our purpose, the visualization and verification of the reconstruction accuracy and detail levels, fifty descriptors is a reasonable amount.

The results obtained satisfy the research primary objective: to establish detailed and accurate description and reconstruction from a given data set.

4. Conclusion

The developed functions showed acceptable results, adding mathematical accuracy, visualization simplicity and manipulation ease to the curve description process. The main concerns regarding the functions are the restriction due to working only with closed curves. For this work purpose, the closed contours description was satisfactory, as it allowed visualization of the description process and accurate reconstruction.

Chapter II

1. Introduction

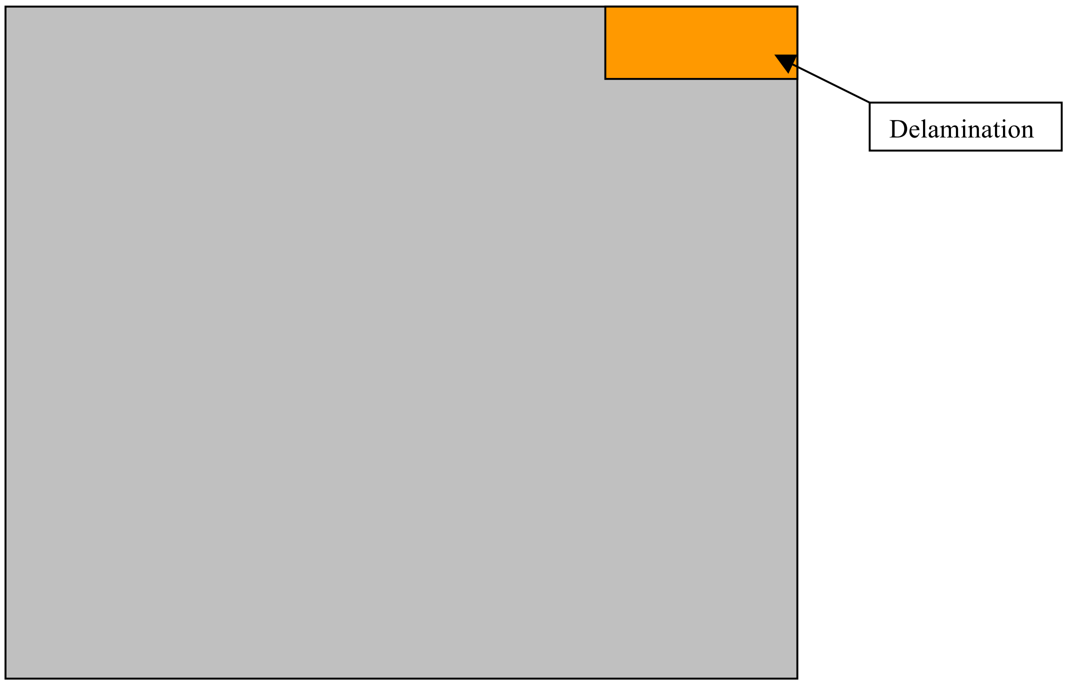
This chapter presents a simple test made with simple data, in order to give experimental support to the mathematical approach we intend to apply. It consists of a comparison between two mode shape sets from the same carbon-fibre-plate. One was provided by a modal analysis, done with several plate border constraints. The other was provided by a modal analysis obeying the same border constraints and experimental conditions, only differing from the first one by a damaged inflicted in a determined plate portion.

2. Our Approach

2.1 Example for Rectangular Plate

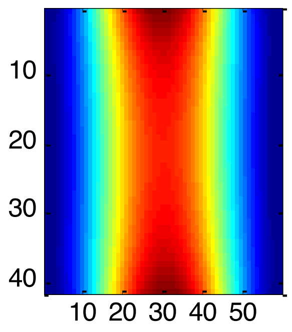
A simple example for the case of a rectangular plate with border constraints (Clamped-Simply Supported-Clamped-Simply Supported) using the Fourier descriptor was developed, in order to present the method effectiveness. Two sets of data, one for a damaged plate and another for an undamaged plate, each one containing 18 mode shape data, were submitted to a Fourier Descriptor - a Matlab command list used to calculate and display the descriptors used to characterize each mode shape – showing several mode shape differences that were taken into account for establishing a more reliable assurance criterion.

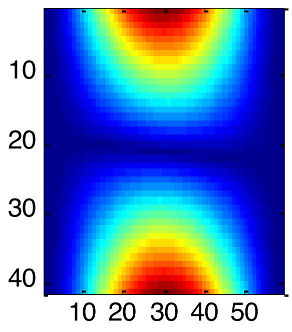
The damage inflicted to the second plate is shown in the following picture. The carbon-fibre-plate was delaminated in a specific area, i.e., the carbon fibre was irregularly posed in a particular plate area, as shown in the picture. Visualizing the mode shapes, it is possible to infer that the damage is located in a plate corner, however they do not suffice to determine the damage location exactly.

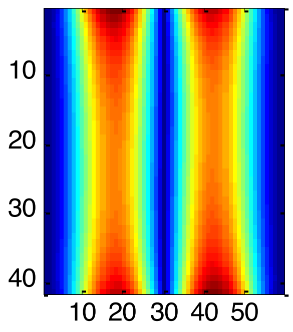


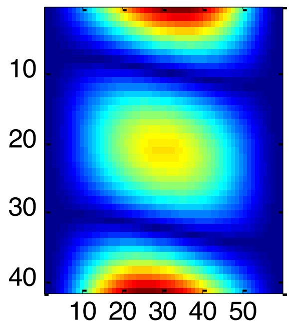
2.1.1 Undamaged Plate Data

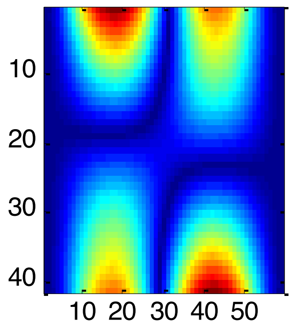
The following 18 mode shapes were extracted from the undamaged plate data.

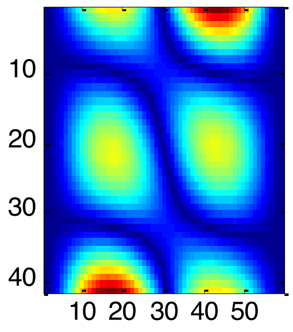

Mode 1

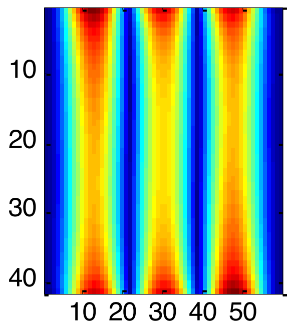

Mode 2

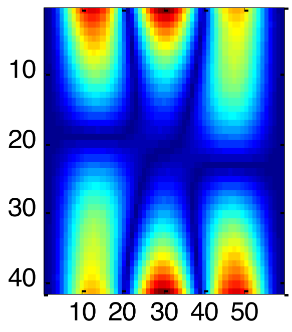

Mode 3

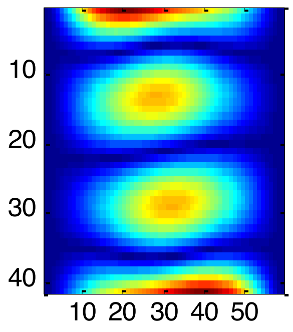

Mode 4

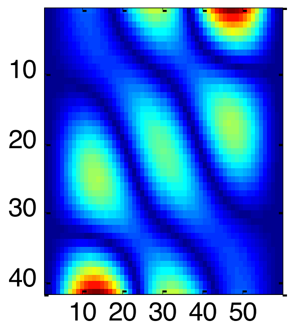

Mode 5

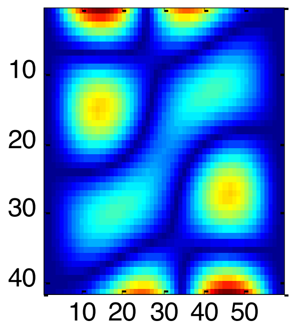

Mode 6

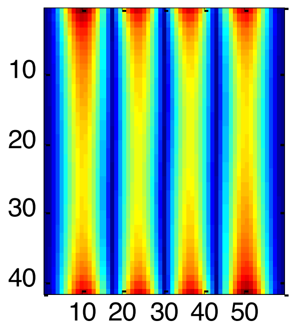

Mode 7

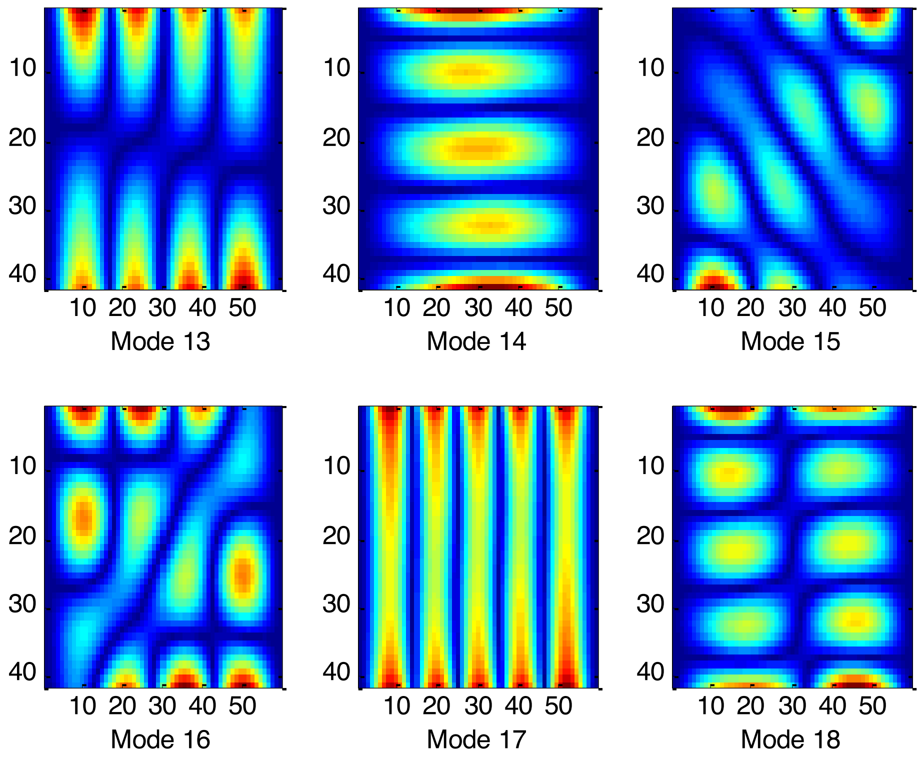

Mode 8


Mode 9


Mode 10


Mode 11


Mode 12



Using the developed Matlab function `reordonne_noeud_lineaire`, described below, 18 files in format .rpt containing the mode shape data were read and that data was transformed into the 18 mode shape images seen above. This function also calls Matlab function `n_line.m`, which displays the nodal line for each mode shape, as exemplified in sequence for mode 1. The process of development of the fuuntions used is described in chapter 5.

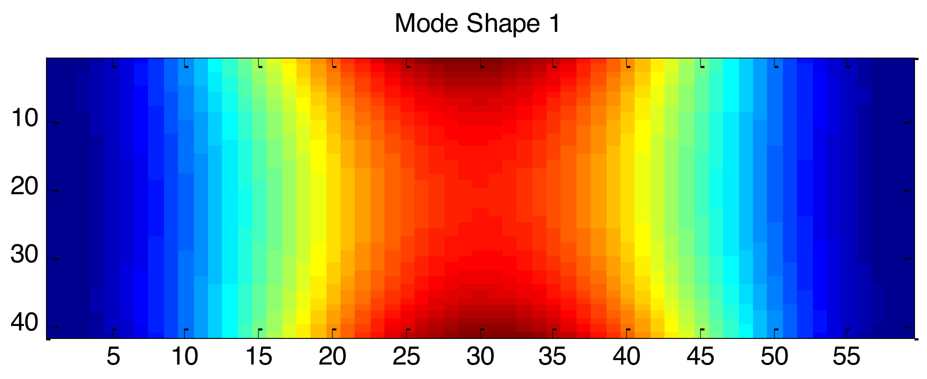
```
function reordonne_noeud_lineaire

clear all
for index=1:18
RPT=sprintf('mode%d.rpt',index) ;
mode=dlmread(RPT);
if (index==1),data(:,index)=mode(:,1); end
data(:,2*index:2*index+1)=mode(:,2:3);
if mod(index,6)==0, tab=6; else tab=mod(index,6);end
if (mod(index,6)==1), figure;end

subplot(2,3,tab)
imagesc((reshape(data(:,2*index),59,[]))')

if (tab==6), pause;end

end

clear mo* in* no*

for index=1:18
n_line((reshape(data(:,2*index),59,[]))',index);
end
```

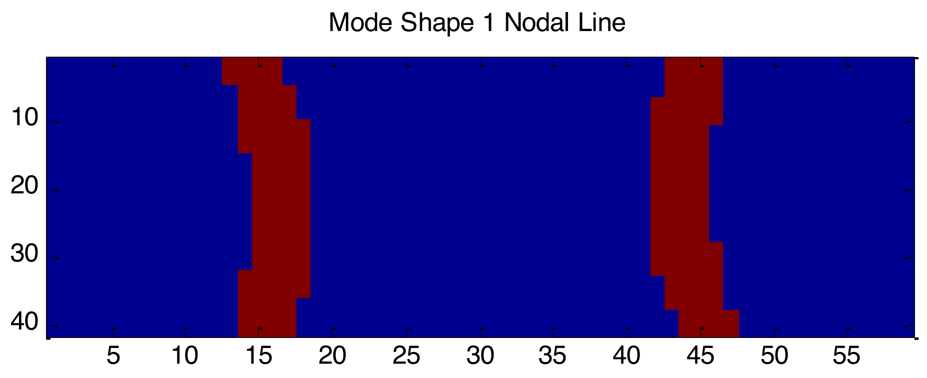
Function *reordonne_noeud_lineaire.m* reads the data stored in the .rpt files, reshaping this data into a matrix format. This format is required in order to transform each .rpt file in a image format (in our programs the bitmap format will be used).

As a combination from both of these functions, the new function *reordonne_noeud_lineaire.m* uses the subfunction *n_line*:

```
function n_line(imagemode,mode)

nomsave=sprintf('mode%d.bmp',mode) ;

        gimg=imagemode;

        %create nodal line by thresholding epsilon
        epsilon=0.1;average=0.5;
 nodalline=(gimg<epsilon+average)&(gimg>-epsilon+average);



%add complement
imwrite(imcomplement(nodalline),nomsave);
% level = graythresh(abs(gimg));
% nodalline = im2bw(abs(gimg),level);

figure;
subplot(211);imagesc(gimg);
subplot(212);imagesc(nodalline);pause;
```

The function automatically extracts the data from .rpt files from modeshapes 1 up to 15, reshaping them into the array named *data.* These reshaped data are displayed altogether. After that, the nodal line for each image format is drawn by the function *n_line*, which also saves the obtained nodal line in a bitmap file. The nodal line is determined by an interval of values detected from the matrix-shaped data. This interval is specified by the variables average and epsilon in function *n_line*. After each graphic shown, the program comes to a halt, allowing the user to visualize the plotting calmly and move on by pressing any key.

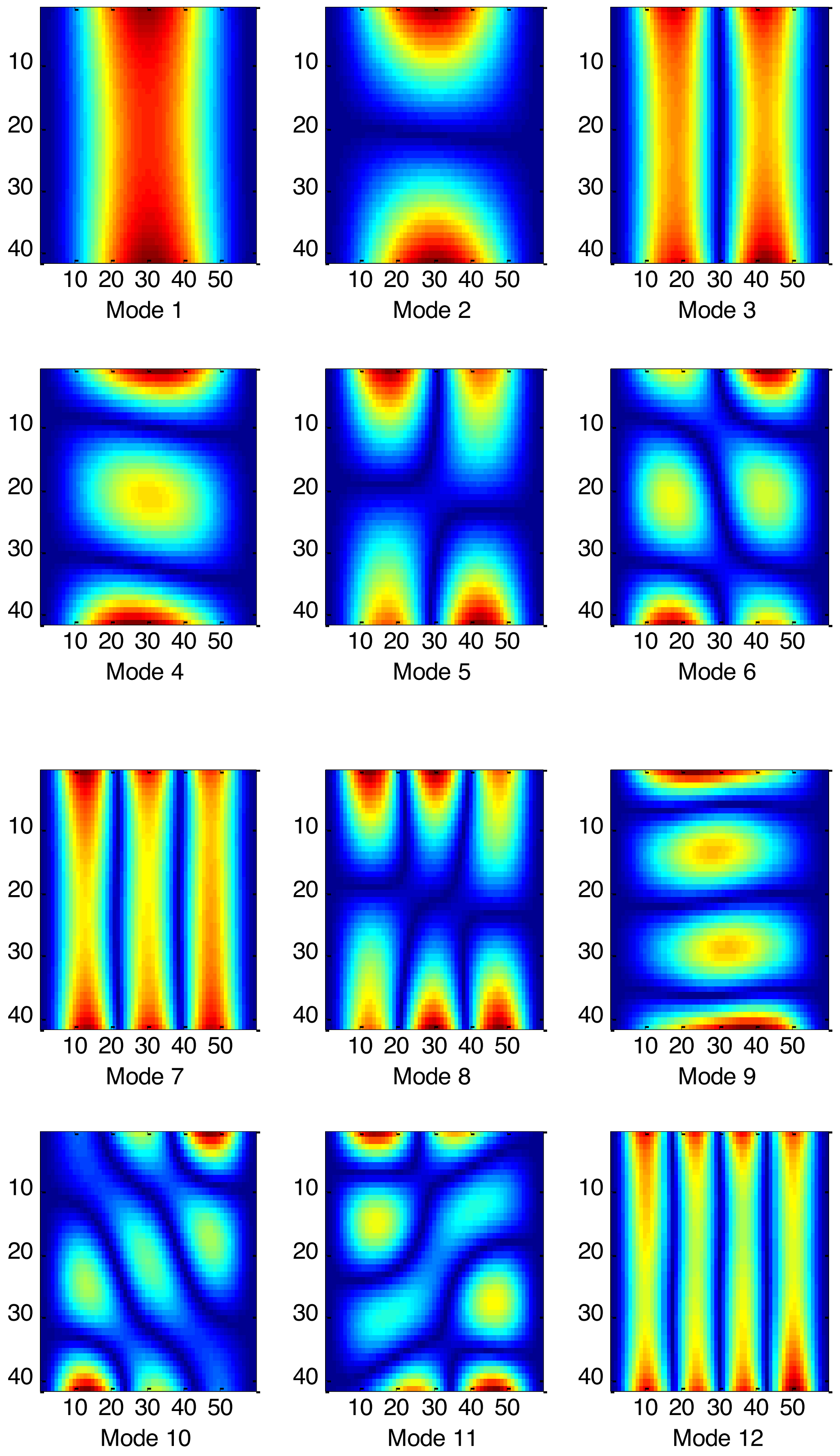


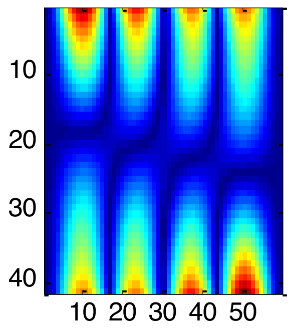


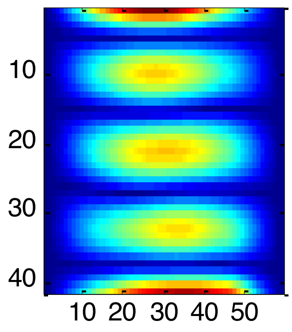
Function `n_line.m` also saves the nodal line information in a bitmap file, for the purpose of using it for further analysis.

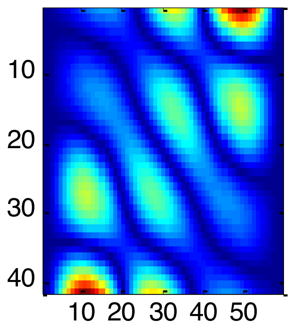
2.1.2 Damaged Plate Data

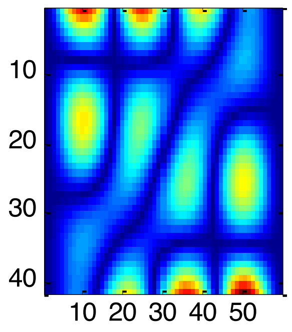
The following 18 mode shapes were extracted from the damaged plate data.

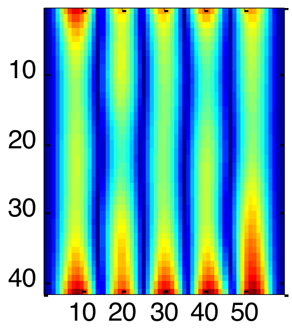

10
20
30
40
10 20 30 40 50
Mode 1
Mode 2
Mode 3
Mode 4
Mode 5
Mode 6
Mode 7
Mode 8
Mode 9
Mode 10
Mode 11
Mode 12

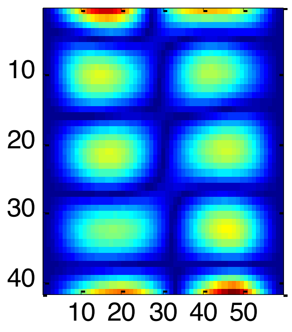

Mode 13


Mode 14

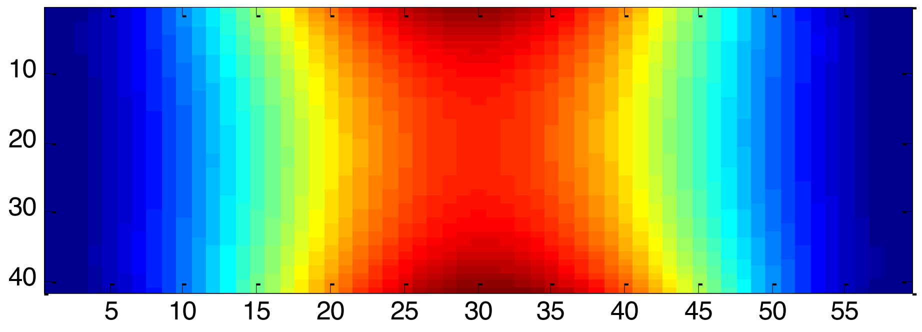

Mode 15

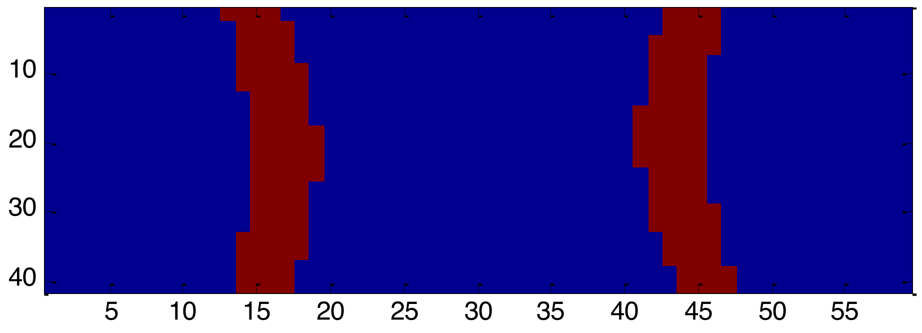

Mode 16


Mode 17


Mode 18

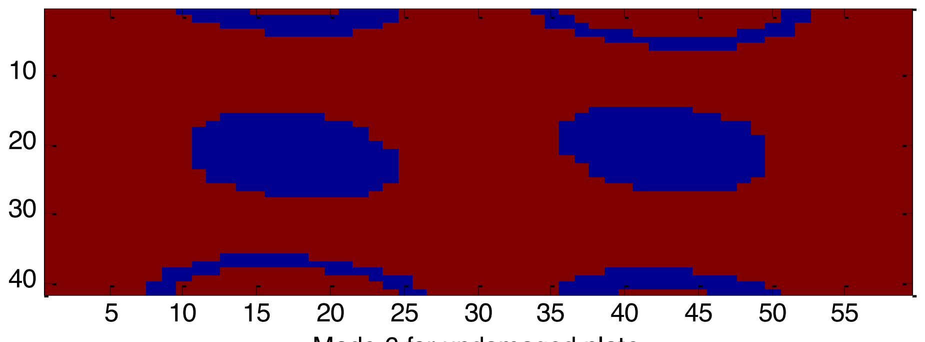
They were obtained using the same procedure as before: reading each .rpt file and producing a mode shape image. The nodal lines were also saved, but in differently named bitmap files. For instance, the nodal line for mode 1 is shown below:


Mode 1 (damaged) image

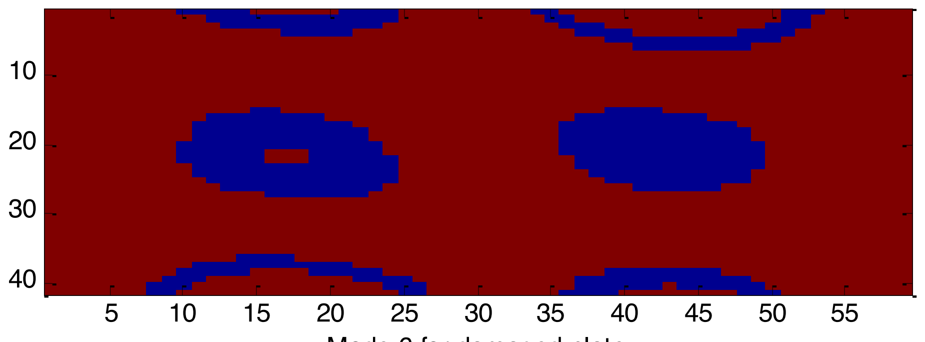

Mode 1 (damaged) nodal line

The difference between mode 1 data from the damaged and the undamaged plate is very slight. Bigger differences can be seen when for mode 5 and higher modes. The detection and quantification of these differences is the main purpose of the research, the difference between the damaged and undamaged.

For example, the differences between mode shape 6 for both plates is almost unperceivable by naked eye. One can only notice the slight change in the border of the formed shapes. However, a computer can easily quantify these differences, and that is what we will use in the development of a new assurance criteria.



Mode 6 for undamaged plate

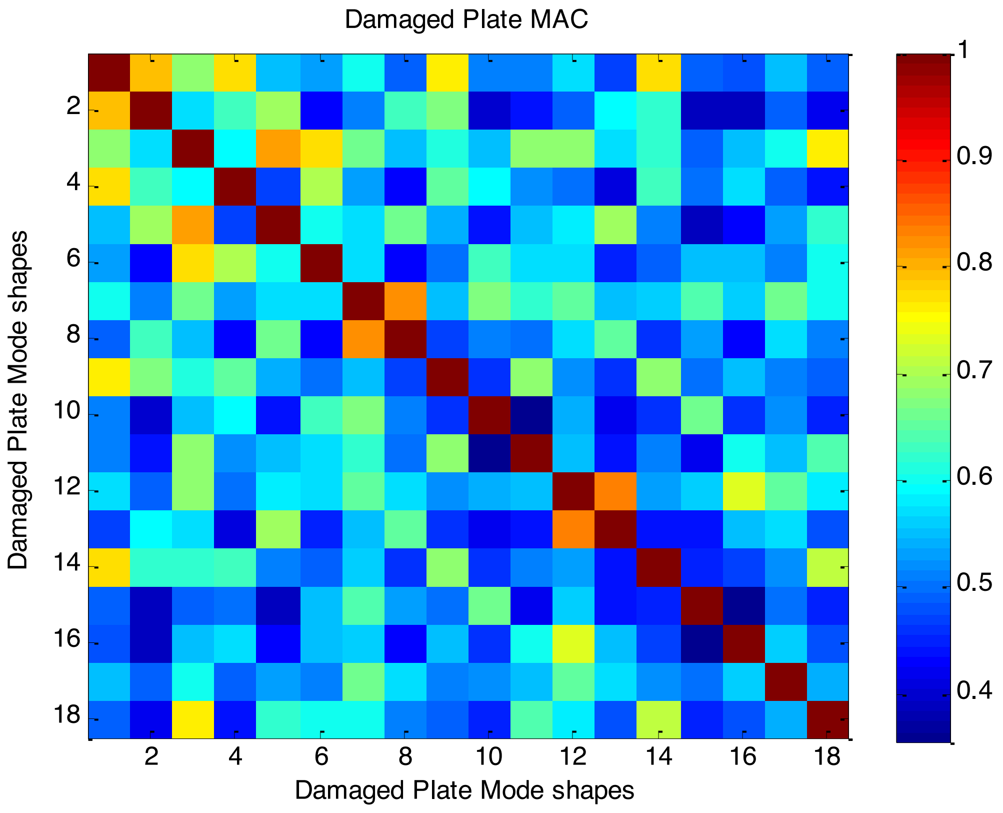


Mode 6 for damaged plate

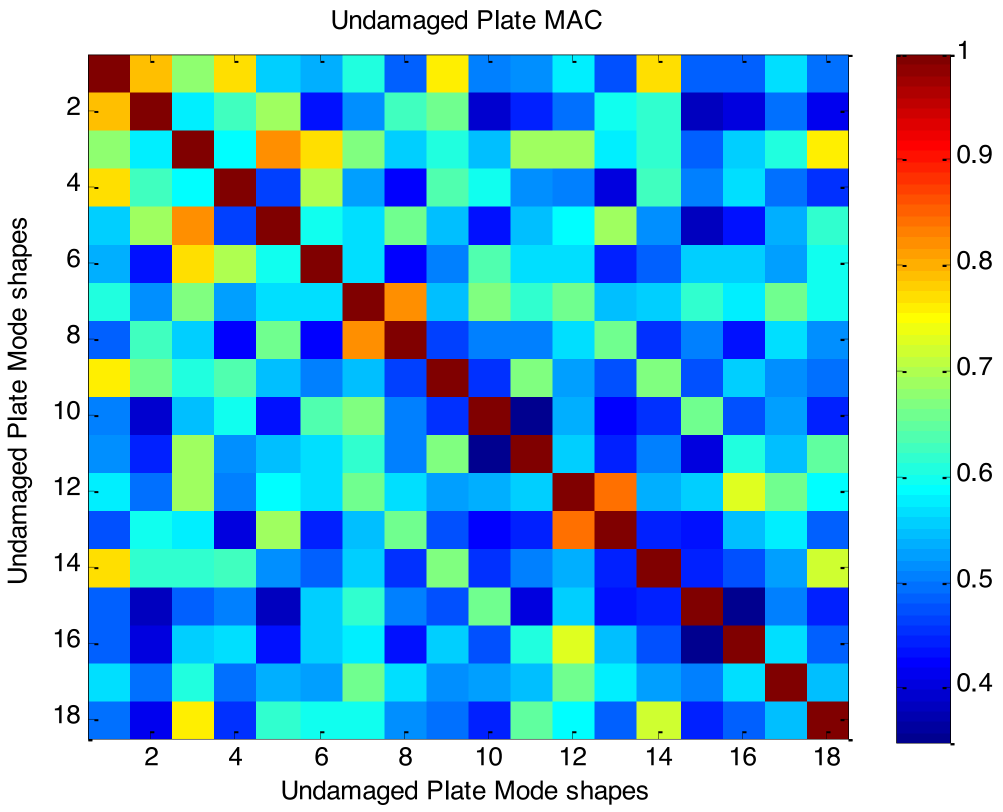
2.2 New Assurance Criteria

Using the definition of MAC, it is possible to create a new criterion, in which multiplying the image vectors from the damaged and undamaged plates will lead to results very similar to the ones given by MAC. Extracting the norm from the product matrix will lead to factors between zero and one, the number one meaning that the maximum assurance is obtained, and the zero meaning the minimal assurance is obtained.

2.2.1 Damaged Plate MAC

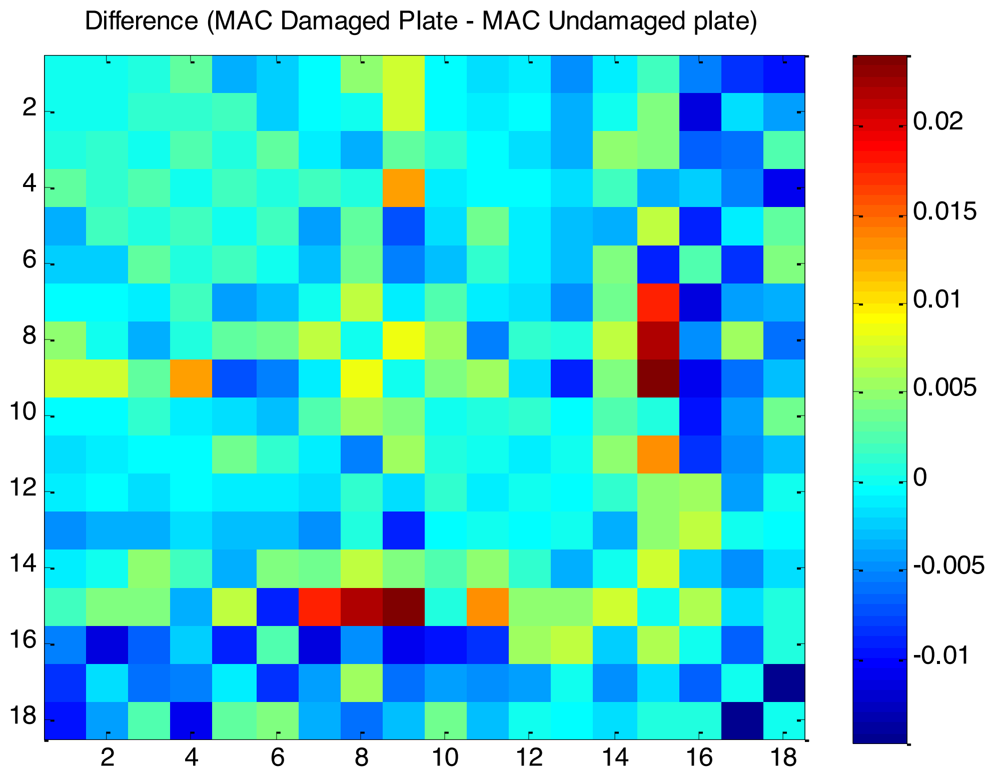


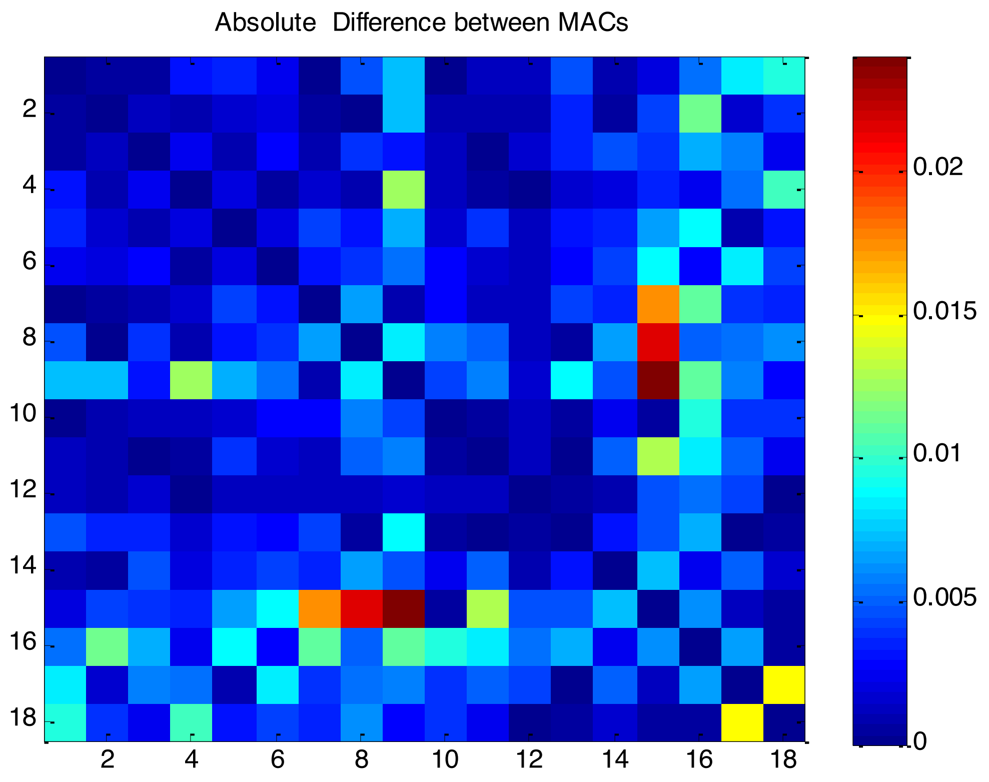
2.2.2 Undamaged Plate MAC



By simple observation it is possible to notice that the obtained criterion presents a higher accuracy, as it shows better differentiation between the various mode shapes. Although it does not show inverse proportionality as the Pearson coefficient calculation (corr2.m), it results in a much more embracing comparison between and wider interpretation of the mode shapes. As an assurance criterion, is has proposed better results than classic MAC itself.

However, in naked eye, it may seem that both results for damaged and undamaged plates are equal. In fact, paying attention being extremely strict to the details, it is possible to notice slight differences between the two obtained matrices. Calculating the difference between them, we obtain the following results:





The greatest differences are noticed when comparing the following mode shapes correlation pairs: (7, 5), (8,15), (9,15) and (17,18) . Keeping in mind that the plate delamination, i.e., the damage inflicted to the plate, it is possible to verify that the MAC result comparison is useful to determine if there is a structural injure, but its precise location is still hard to apprehend using just that (although it can be seen inferred from the mode shapes visualization). A more elaborated comparison method might be needed to fit this purpose.

3. <u>Conclusion</u>

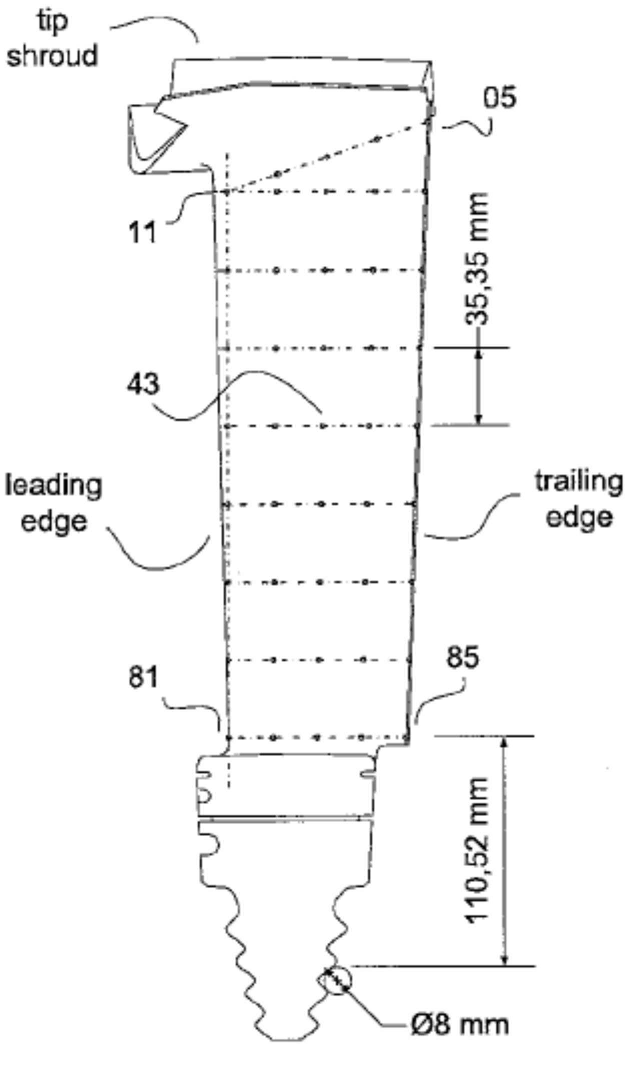
As assurance criteria, the use of mode shape image format presented wider and deeper results than the classic Modal Assurance Criteria. It was possible to detect greater range of differences between the mode shapes. And the results for damaged and undamaged bodies have shown a measurable difference, which can contribute to the identification of damaged bodies.

Nonetheless, the comparison methods used (simple difference) are not sufficient to determine with accuracy the damage location. Further efforts must then be done in the development of more sophisticated comparison methods.

Chapter III

1. CATIA Model

The modelling in Catia was guided by the test results in document [XX]. Although the model shown in [XX], presented below, gives some geometric information about the tested structure, like its length and existence of a tip shroud, it does not make clear a whole group of other interesting data, essential to build a reliable model which can lead to the same results. Despite of this, it does not mean our model is unrealistic, it just means that the data obtained from the experiment report is only to be used as advisory guide, as long as the remaining geometric features are not obtained.

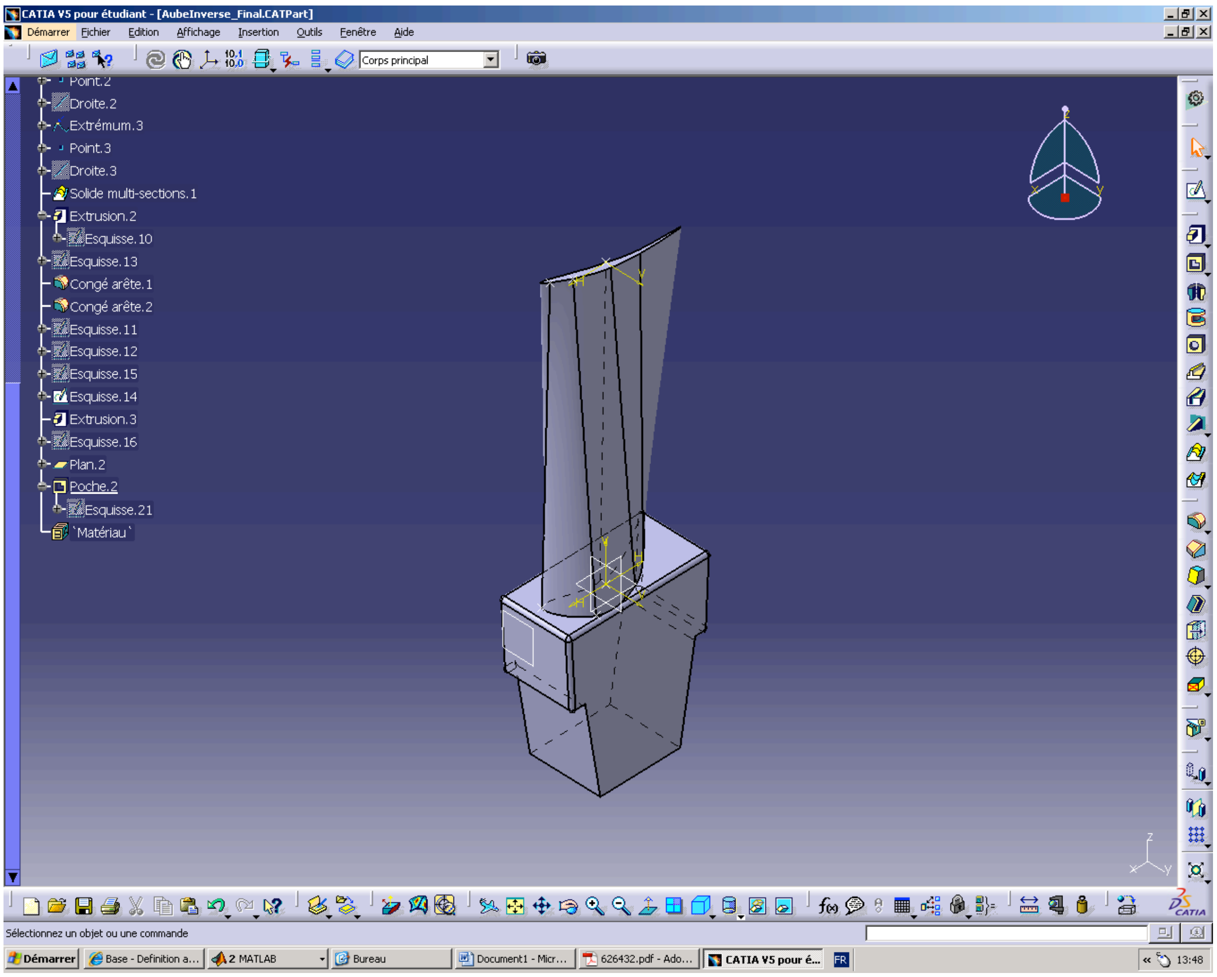


Turbine blade representation

The Catia model was initially designed in order to obtain a representation for the turbine blade studied in article **[13]**, knowing the blade geometry features used in the experiment there described were unavailable at the moment. So, based in a schematic picture shown in document **[13]**, in previous knowledge from turbine blade design and in the modal frequency values obtained in a finite-element test, the Catia model was shaped as an attempt to reach the mentioned frequencies and respect the schematics shown in the previous figure.

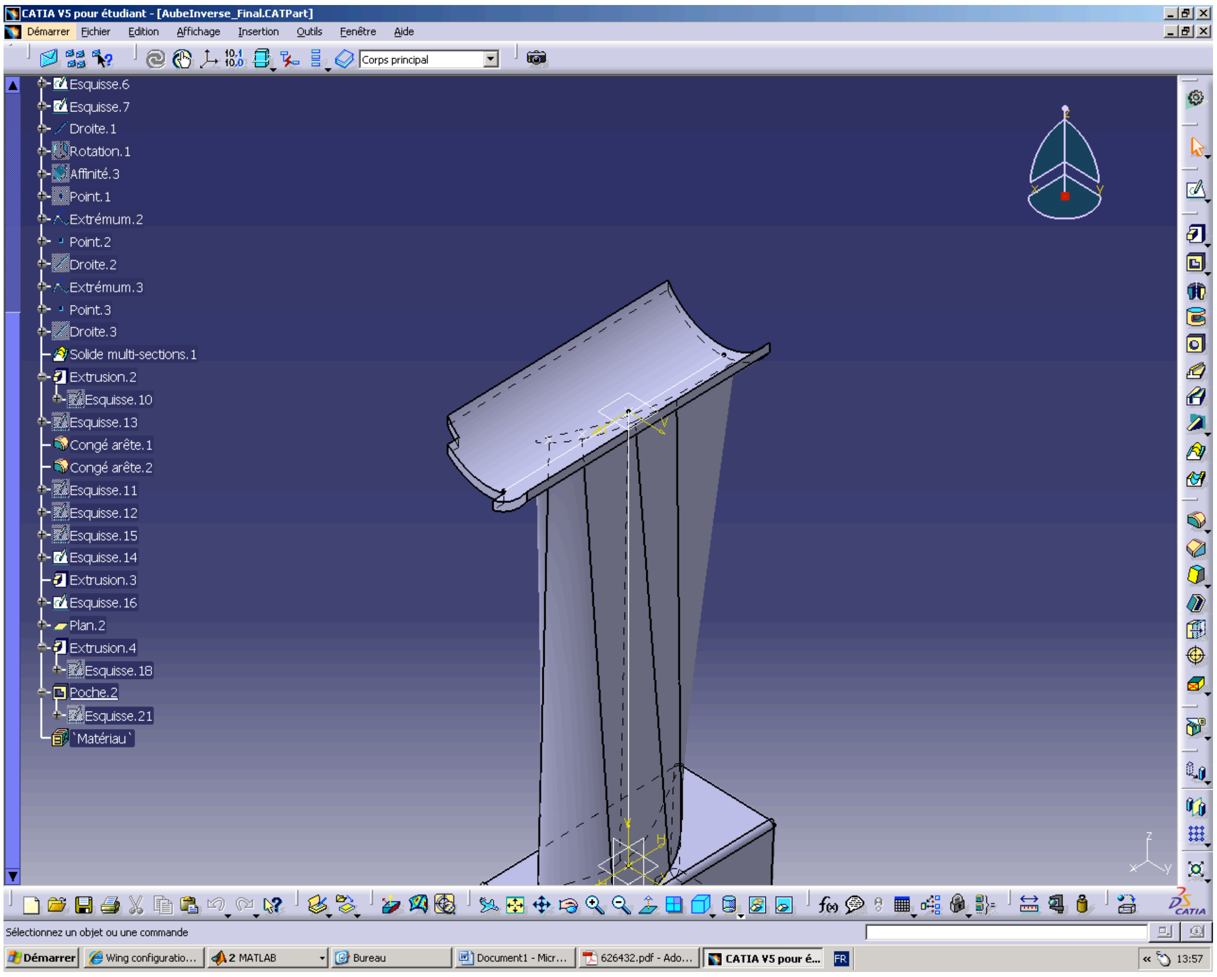
Although it is known that this approach may not result in this geometry accurate representation, the initial objectives in this research are to establish if the applied method is reasonable for a possible, realistic turbine blade model.

The model was basically constructed building a wing-like profile mounted in a static base. In our displacement test, this base is will be the clamped part and the profile will be the displacement surface. As ordinarily adopted in turbine blade construction, the profile clamped in the base (thicker) is different from the one above (narrower).



Partial model in Catia

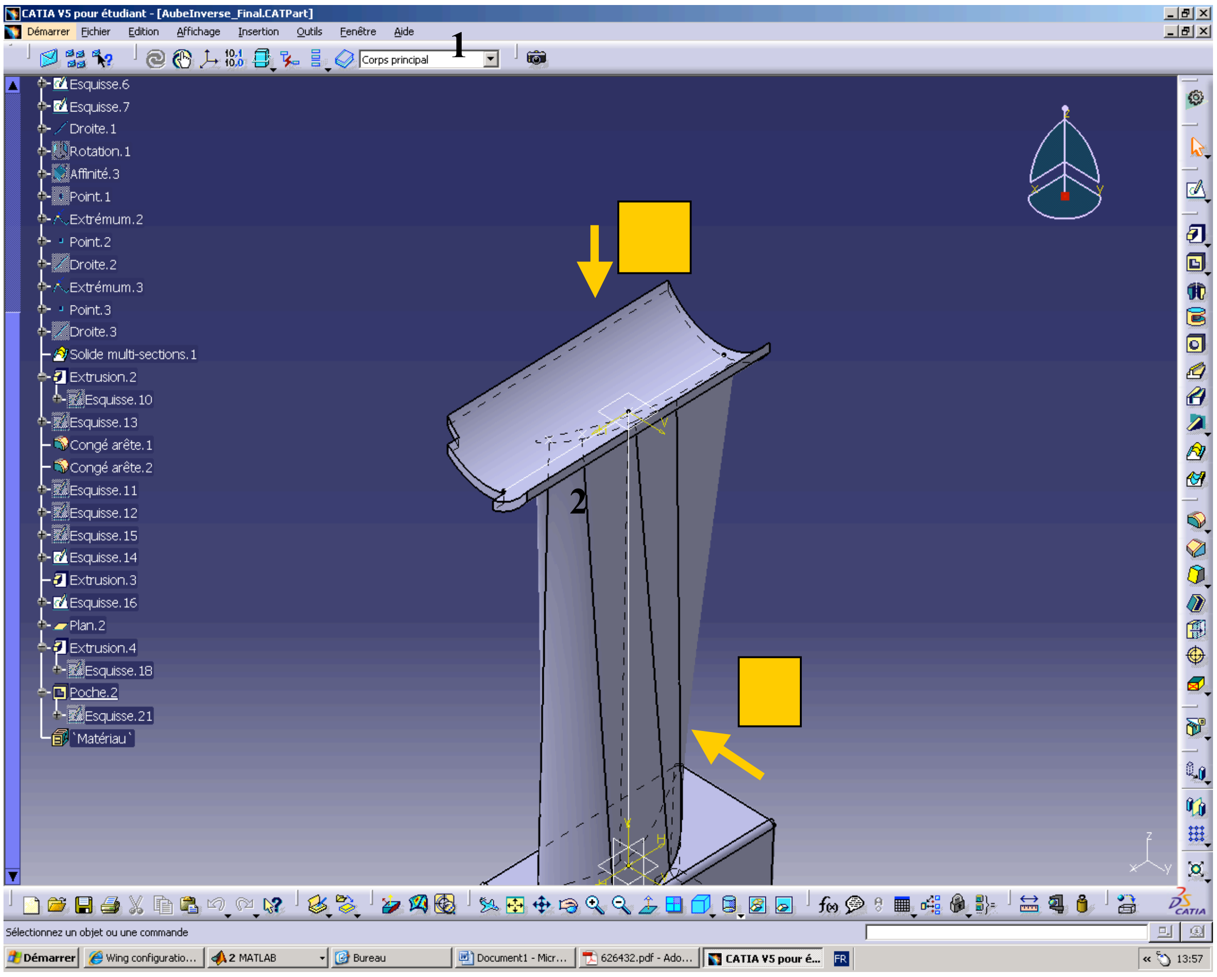
Finally a final top structure is added, for it allows assembling various blades side by side. It is expected that the blades might be assembled to form a circular turbine, but for the purpose of this modelling, the curvature from the blade structure itself (curvature in plane XZ) was not considered.



Complete model in Catia

As it was built, there are many factors that can influence deeply in the obtained modal frequency values. For example: the choice of each profile (top, base); the relative torsion between them; the profiles thickness; the top structure thickness; the top structure curvature (if necessary). Therefore, we may only adopt a qualitative approach towards the available set of data.

The turbine blade deformation data will be divided in two distinct sets. We will study the deformation obtained from the tip surface, and the deformation obtained from the lateral surface. Like represented in below, we will work with a top view from the tip (number 1) and a lateral view from the side (number 2).



CATIA data sets

2. Frequencies

The finite element test was made for several different mesh unit sizes. From a 5mm mesh unit size up to an 18mm size, the obtained results for the first six mode shapes were:

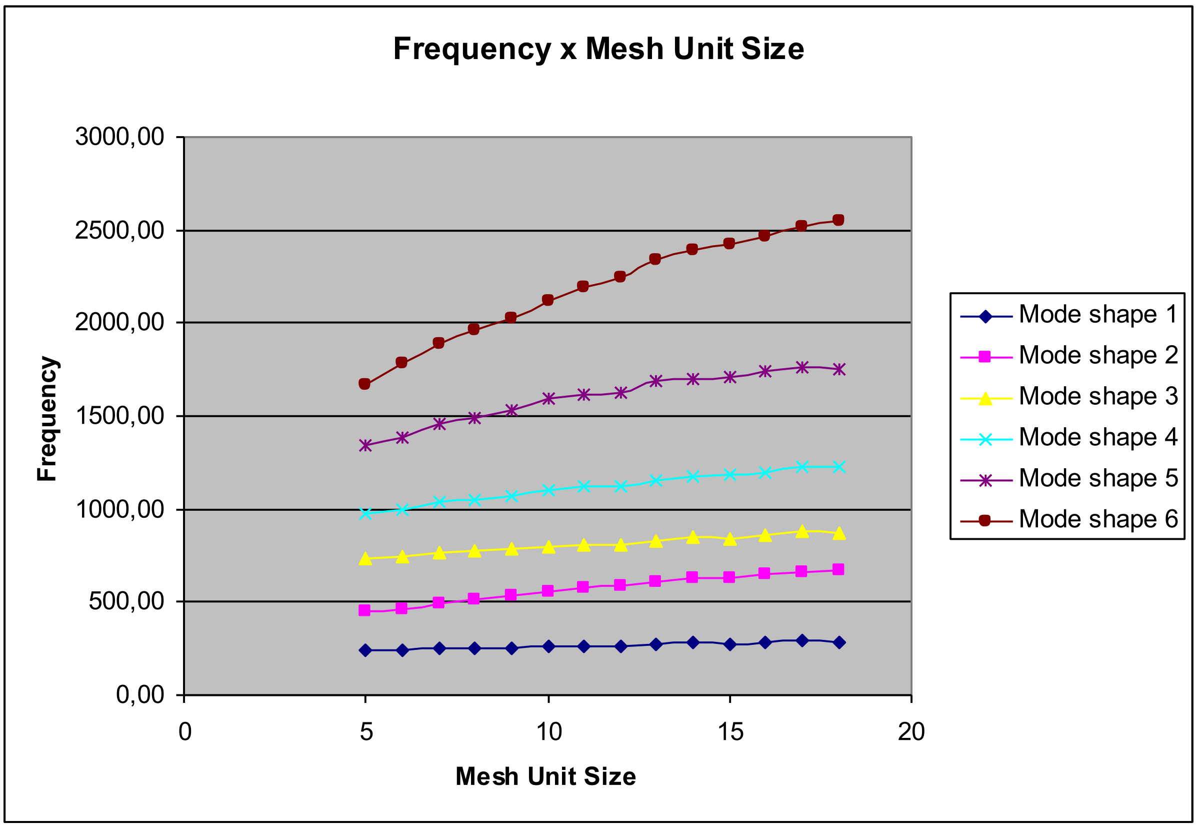
Table 1 – Catia model frequencies

	Mode Shape Frequencies in Hz							
Mode shape	5 mm mesh	6 mm mesh	7 mm mesh	8 mm mesh	9 mm mesh	10 mm mesh	11 mm mesh	12 mm mesh
1	241,43	244,61	248,28	252,15	256,01	260,38	265,64	267,23
2	447,90	465,37	490,84	509,94	531,87	556,04	575,86	585,13
3	731,36	743,43	763,87	772,56	784,91	800,11	812,87	812,14
4	980,21	1001,52	1036,00	1053,11	1074,06	1102,58	1119,70	1121,37
5	1343,82	1388,15	1457,71	1487,65	1527,87	1590,96	1616,45	1627,10
6	1671,25	1778,14	1887,37	1957,99	2028,80	2119,28	2193,61	2245,33

Table 1 (cont.) – Catia model frequencies

	Mode Shape Frequencies in Hz					
Mode shape	13 mm mesh	14 mm mesh	15 mm mesh	16 mm mesh	17 mm mesh	18 mm mesh
1	274,04	279,87	275,76	279,66	288,60	288,24
2	611,13	624,65	631,28	649,10	665,65	666,18
3	833,18	845,94	844,36	860,39	882,83	874,07
4	1157,38	1179,86	1181,21	1197,19	1222,74	1226,20
5	1690,10	1700,75	1710,59	1741,65	1766,38	1756,40
6	2341,61	2389,80	2427,84	2460,37	2513,62	2549,41

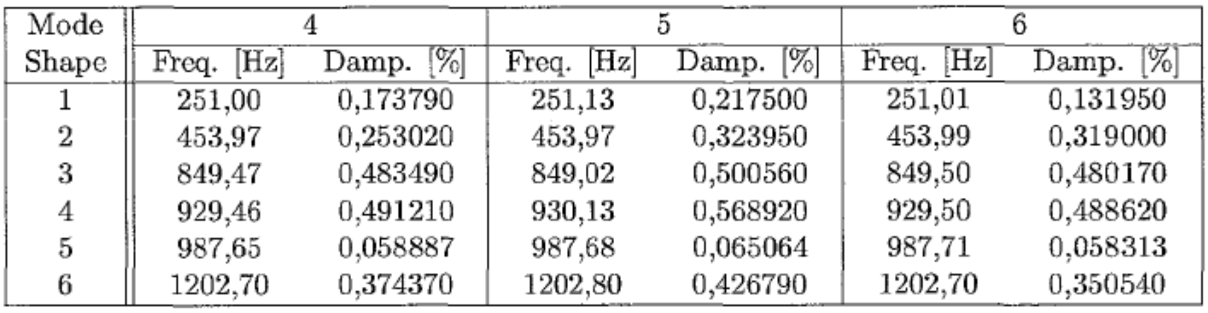
This interval was chosen given that for a mesh unit size lower than 5mm the storage computer capacity needed exceeds the available capacity of the computer used for the tests, and a size of 18mm already leads to results that greatly disagree with the experimental ones. These frequencies give an idea of how the frequencies change when we simply change the type of mesh used in finite element testing. For a better pattern observation, we can display them in a more direct graphic:



This shows that for the first modes, very little numerical discrepancy is noticed when changing the mesh unit size. But for the later modes, it is expected that this discrepancy rises as high as the mode order is. A further evaluation of the method effectiveness can be done taking into account that discrepancy.

When comparing with the results from the experience **[12]**:

Table 2 – Experimental frequencies



Mode Shape	4		5		6	
	Freq. [Hz]	Damp. [%]	Freq. [Hz]	Damp. [%]	Freq. [Hz]	Damp. [%]
1	251,00	0,173790	251,13	0,217500	251,01	0,131950
2	453,97	0,253020	453,97	0,323950	453,99	0,319000
3	849,47	0,483490	849,02	0,500560	849,50	0,480170
4	929,46	0,491210	930,13	0,568920	929,50	0,488620
5	987,65	0,058887	987,68	0,065064	987,71	0,058313
6	1202,70	0,374370	1202,80	0,426790	1202,70	0,350540

The chosen mesh for our research will be the one with 10mm unit size. For the four first modes, it presents an acceptable percentage discrepancy from the experimental data, plus it is a mesh that is most easily transformed into a square matrix image, which will reduce our efforts in reshaping the finite element data into a bitmap image representation.

Table 3 – Experiment-model percentage frequency difference

10 mm mesh	
Mode Shape	Difference (%)
1	3,74
2	22,48
3	5,81
4	18,63
5	61,03
6	76,21

It is essential to also remember the finite element test is not susceptible to experimental errors that may occur. So, considering the original data and the final results, the model can be characterized as realistic, and will fit the primary purposes.

3. Creating Image file

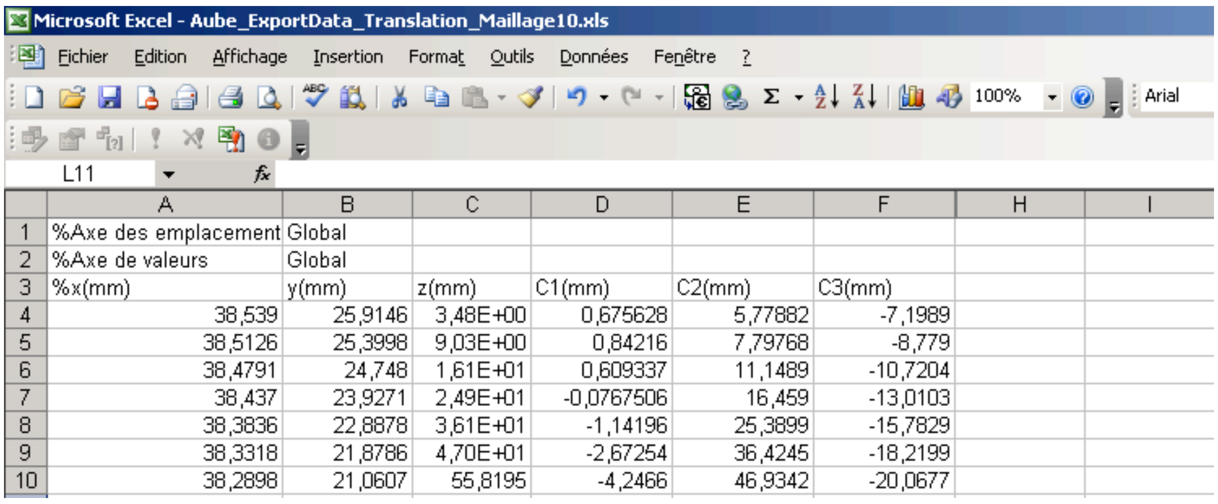
3.1 Using CATIA Export Data

The data extracted from the CATIA model are a mesh of nodes created according to the software capability of finite element calculation. The unit cell geometry for this model is defined based in the available tools for finite element calculation (in our model the unit cell is tetrahedron-shaped). This geometry follows the body geometry, which is irregular. It is important to convert the available data into a matrix, so that it can be more easily mathematically manipulated. For this purpose, the development of Matlab functions using Radial Basis Functions concepts for interpolation and reshaping of the original data was necessary.

So, basically, the reshaping process will proceed in several steps:

- First, we select from the entire model mesh, which represents the entire turbine blade geometry, the nodes that we want to analyze. This is merely a selection of which turbine blade part is going to be observed;
- Then, we create a surface that represents these selected nodes. This can be done by simply establishing an interval and creating the surface with the default Matlab command *mesh*;
- At this moment, the RBF takes place. The surface obtained from the previous step will not likely be rectangular. The RBF approximation will provide a rectangular new shape, equivalent to the previous surface.
- Finally, the rectangular resulting surface is already the desired matrix, which means it can be already stored as an image format (in our work the bitmap format will be used). This image is ready to further mathematical manipulations.

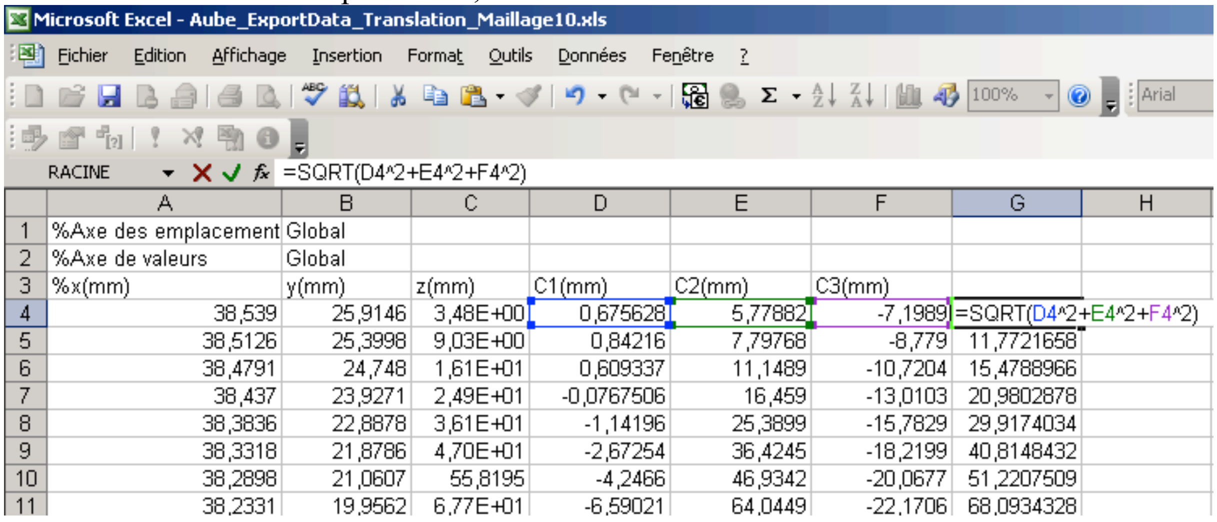
Assuming the CATIA model is ready, when we export its modal analysis data, a .xls file is generated. For each modal analysis, therefore each mode shape, the .xls file organization is:


Microsoft Excel - Aube_ExportData_Translation_Maillage10.xls
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
100%
Arial
L11
fx

	A	B	C	D	E	F	H	I
1	%Axe des emplacement	Global						
2	%Axe de valeurs	Global						
3	%x(mm)	y(mm)	z(mm)	C1(mm)	C2(mm)	C3(mm)		
4	38,539	25,9146	3,48E+00	0,675628	5,77882	-7,1989		
5	38,5126	25,3998	9,03E+00	0,84216	7,79768	-8,779		
6	38,4791	24,748	1,61E+01	0,609337	11,1489	-10,7204		
7	38,437	23,9271	2,49E+01	-0,0767506	16,459	-13,0103		
8	38,3836	22,8878	3,61E+01	-1,14196	25,3899	-15,7829		
9	38,3318	21,8786	4,70E+01	-2,67254	36,4245	-18,2199		
10	38,2898	21,0607	55,8195	-4,2466	46,9342	-20,0677		

The first column data represents the x-coordinate for one given node; the second column represents the y-coordinate; and the third column the z-coordinate. The fourth column, named C1, represents the displacement in X direction; the fifth column, named C2, represents the displacement in Y direction; the sixth column, named C3, represents the displacement in Z direction. We use these coordinate for choosing the geometric region that we want to analyze.

After choosing the area of interest, and eliminating the non-interesting area, the total displacement is calculated by the square root of the sum of the square of each displacement. In order to work with the total displacement, a seventh column must be created:


Microsoft Excel - Aube_ExportData_Translation_Maillage10.xls
Fichier Edition Affichage Insertion Format Outils Données Fenêtre ?
100%
Arial

RACINE =SQRT(D4^2+E4^2+F4^2)

	A	B	C	D	E	F	G	H
1	%Axe des emplacement	Global						
2	%Axe de valeurs	Global						
3	%x(mm)	y(mm)	z(mm)	C1(mm)	C2(mm)	C3(mm)		
4	38,539	25,9146	3,48E+00	0,675628	5,77882	-7,1989	=SQRT(D4^2+E4^2+F4^2)	
5	38,5126	25,3998	9,03E+00	0,84216	7,79768	-8,779	11,7721658	
6	38,4791	24,748	1,61E+01	0,609337	11,1489	-10,7204	15,4788966	
7	38,437	23,9271	2,49E+01	-0,0767506	16,459	-13,0103	20,9802878	
8	38,3836	22,8878	3,61E+01	-1,14196	25,3899	-15,7829	29,9174034	
9	38,3318	21,8786	4,70E+01	-2,67254	36,4245	-18,2199	40,8148432	
10	38,2898	21,0607	55,8195	-4,2466	46,9342	-20,0677	51,2207509	
11	38,2331	19,9562	6,77E+01	-6,59021	64,0449	-22,1706	68,0934328	

When developing the Matlab function that will read each set of data, we must keep in mind that the coordinates and total displacement are located in the first three columns and in the seventh one, respectively.

For more direct utilization, all 15 mode shapes where reunited in one single .xls file. The .xls file was called Aube_ExportData_Translation_Maillage10 ("Turbine blade export displacement data for a mesh unit size of 10mm"). This file contains 15 pages, each one named Mode #, where # corresponds to the mode shape order (1, 2, 3…15), and each page has the explained column organization.

3.2 Interpolation using Radial Basis Function (RBF)

A *radial basis function (RBF)* is a real-valued function whose value depends only on the distance from the origin, so that $\Phi(x) = \Phi(\|x\|)$; or alternatively on the distance from some other point ***C***, called a *center*, so that $\Phi(x - C) = \Phi(\|x - C\|)$. Any function Φ that satisfies the property $\Phi(x) = \Phi(\|x\|)$ is a radial function. The norm is usually Euclidean distance, although other distance functions are also possible. **[18]**

Radial basis functions can be used to build up function approximations of the form:

$$y(x) = \sum_{i=1}^{N} w_i \Phi(\|x - C_i\|)$$

Where the approximating function *y(x)* is represented as a sum of *N* radial basis functions, each associated with a different center C_i, and weighted by an appropriate coefficient w_i. The weights w_i can be estimated using the matrix methods of linear least squares, because the approximating function is *linear* in the weights.

The Matlab function developed for that purpose is shown below:

```
clear all; close all;

mode=1;

modename=sprintf('Mode %d',mode) ;
y=xlsread('Aube_ExportData_Translation_Maillage10',modename);
%big NaN value from mac ?
y=y(4:end,:);

y1lin=linspace(min(y(:,1)),max(y(:,1)),33);
y3lin=linspace(min(y(:,3)),max(y(:,3)),33);
figure;
[X,Z]=meshgrid(y1lin,y3lin);

y(any(isnan(y),2),:) = [];

 W=griddata(y(:,1),y(:,3),y(:,7),X,Z);

%RBF interpolation
op=rbfcreate([y(:,1)'; y(:,3)'], y(:,7)','RBFFunction', 'multiquadric',
'RBFConstant', 2);

rbfcheck(op);
%rbf=rbfcreate([y(:,1)'; y(:,3)'], y(:,7)',op);
WI = rbfinterp([X(:)'; Z(:)'], op);
WI = reshape(WI, size(X));

 mesh(X,Z,W); % interpolated

plot3(y(:,1),y(:,3),y(:,7),'.','MarkerSize',15)
view(2)
figure;
 mesh(X,Z,WI); view(2)
```

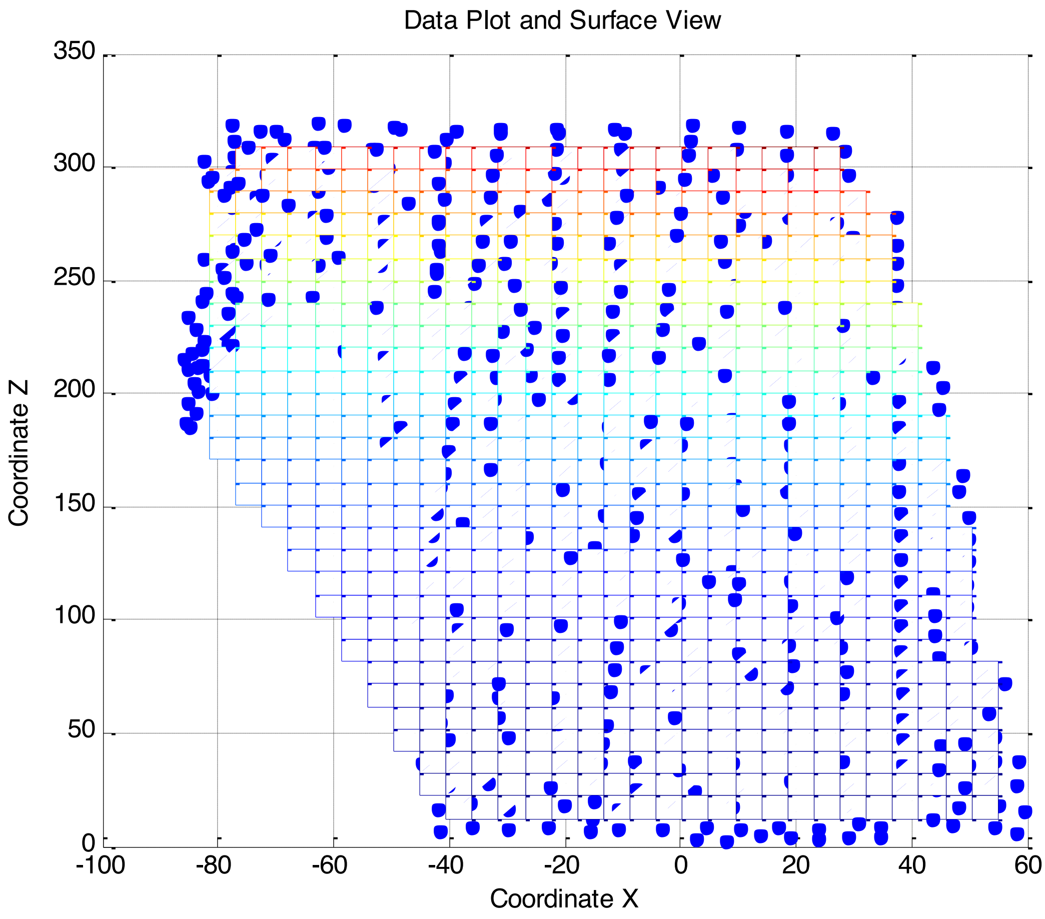
The Matlab function shown reads the .xls file called `Aube_ExportData_Translation_Maillage10` and extracts the working data from one of its pages, the one named “Mode #”, where # is the number of the mode shape defined by the variable `mode`. After creating vectors `y1lin` and `y3lin`, which store, respectively, the x and z regular interval between the minimum and maximum x and z coordinate values, the program eliminates any singularity that may occur in data using the function (isnan). Then it generates a mesh that represents the data using RBF interpolation, using the functions `rbfcreate`, `rbfcheck` and `rbfinterp`, which are shown in Appendix.

4. MAC

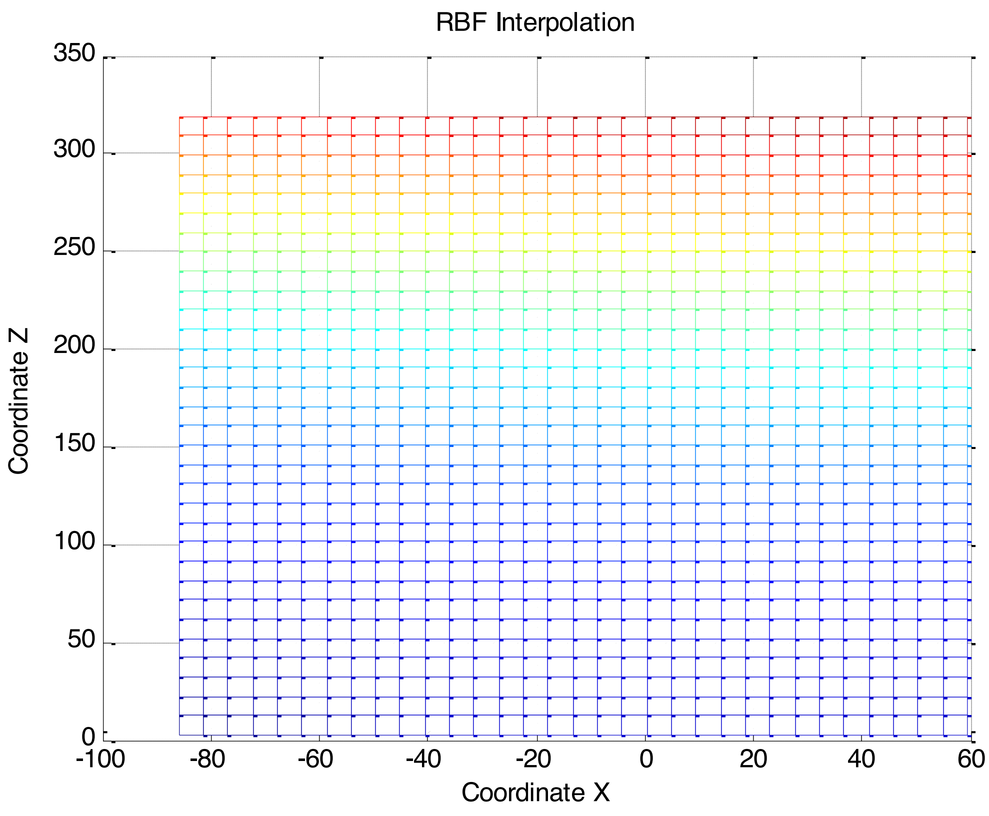
In order to execute the Radial Basis Function interpolation for every mode shape, the function above is modified. The following function is the one created for calculating and displaying all 15 mode shapes. After that, the function uses the created image format to calculate the modal assurance criteria, using function `macD.m`, as stated in Appendix. It basically reproduces the classic MAC, using the mode shape image as the input data matrix for each mode shape.

```
r mode=1:15

dename=sprintf('Mode %d',mode) ;
xlsread('Aube_ExportData_Translation_Maillage10_Top',modename);
ig NaN value from mac ?
y(4:end,:);

lin=linspace(min(y(:,1)),max(y(:,1)),33);
lin=linspace(min(y(:,2)),max(y(:,2)),33);
gure(1);subplot(3,5,mode);
,Z]=meshgrid(y1lin,y2lin);

any(isnan(y),2),:) = [];

=griddata(y(:,1),y(:,2),y(:,7),X,Z);

BF interpolation
=rbfcreate([y(:,1)'; y(:,2)'], y(:,7)','RBFFunction', 'multiquadric',
BFConstant', 2);
fcheck(op);
of=rbfcreate([y(:,1)'; y(:,3)'], y(:,7)',op);
 = rbfinterp([X(:)'; Z(:)'], op);
 = reshape(WI, size(X));

esh(X,Z,W); % interpolated
old on;

ot3(y(:,1),y(:,2),y(:,7),'.','MarkerSize',15)
ew(2)
tle(sprintf('Mode Shape %d',mode));

gure(2);subplot(3,5,mode);
esh(X,Z,WI); view(2);title(sprintf('Mode Shape %d',mode));

desave=sprintf('aubemode_top%d.bmp',mode) ;

dd complement
write(imcomplement(WI),modesave);
level = graythresh(abs(gimg));
nodalline = im2bw(abs(gimg),level);

gure(3);subplot(3,5,mode);
agesc(WI);title(sprintf('Mode Shape %d',mode));
m(:,mode)=reshape(WI,1,[]);
ld on;

d

gure;

ac=macD(LWm,LWm);
agesc(vmac);colorbar
abel('Turbine Blade Mode Shapes')
abel('Turbine Blade Mode Shapes')
```

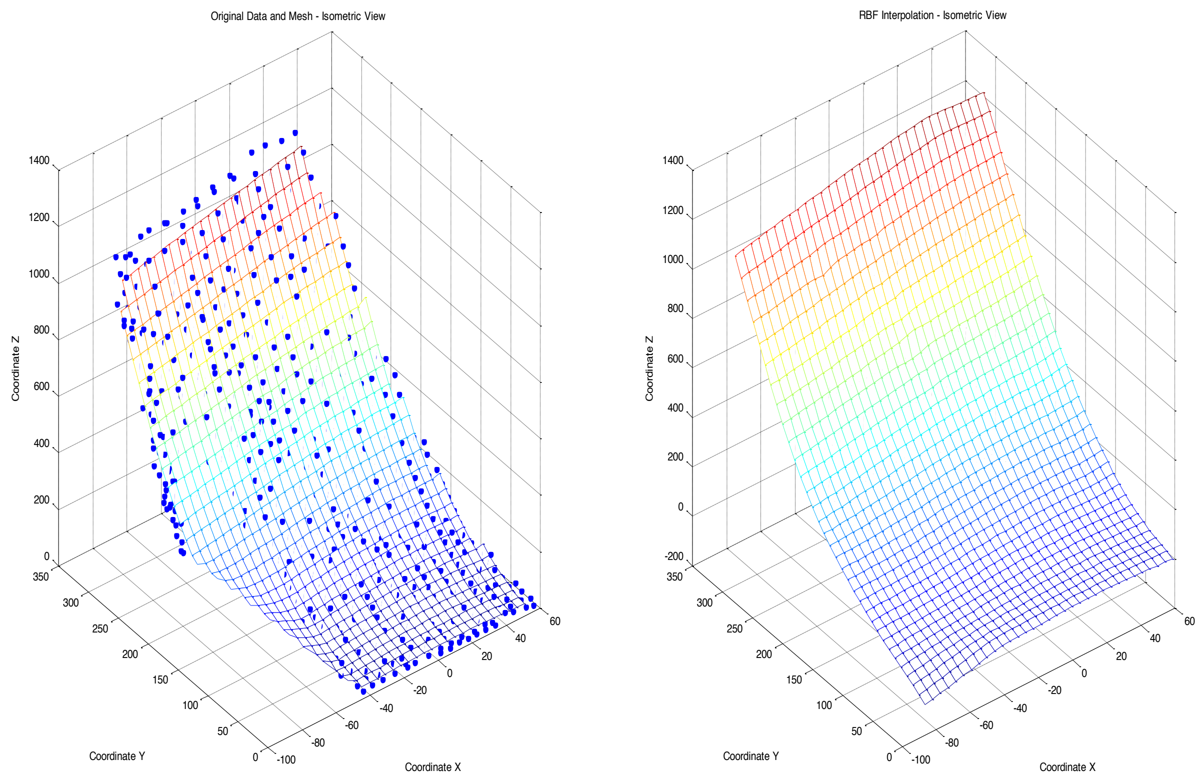
The function `test.m`, as stated above, creates mesh, RBF interpolation and image format for each mode shape, and, when this is done for every mode shape, calculates the MAC using as data input the image format of the 15 mode shapes. The obtained results are:



The first one is a superposition between the real data extracted from the .xls file, for mode shape 1, from the lateral data set (region 2) and the interpolation surface created using Matlab command *mesh*. We can notice that the surface boundaries are not rectangular shaped.

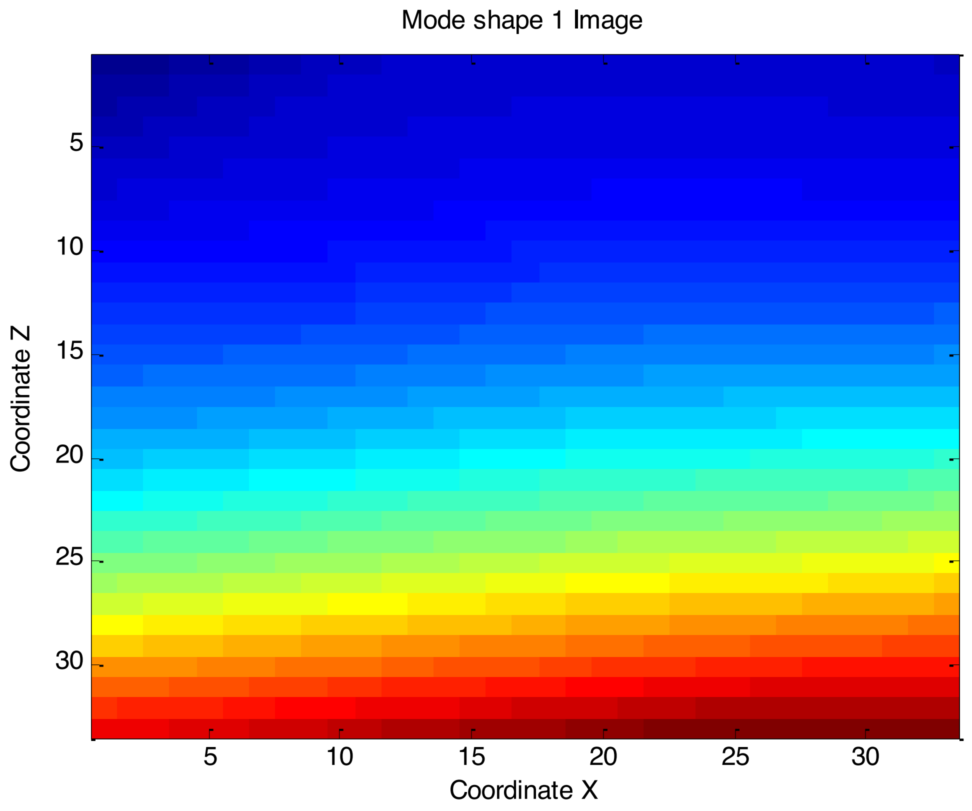


The second plot is RBF interpolation that reshapes the original surface into one with rectangular boundaries, in order to store it as a matrix afterwards. For instance, if we plotted the results side by side in isometric view, we would obtain:



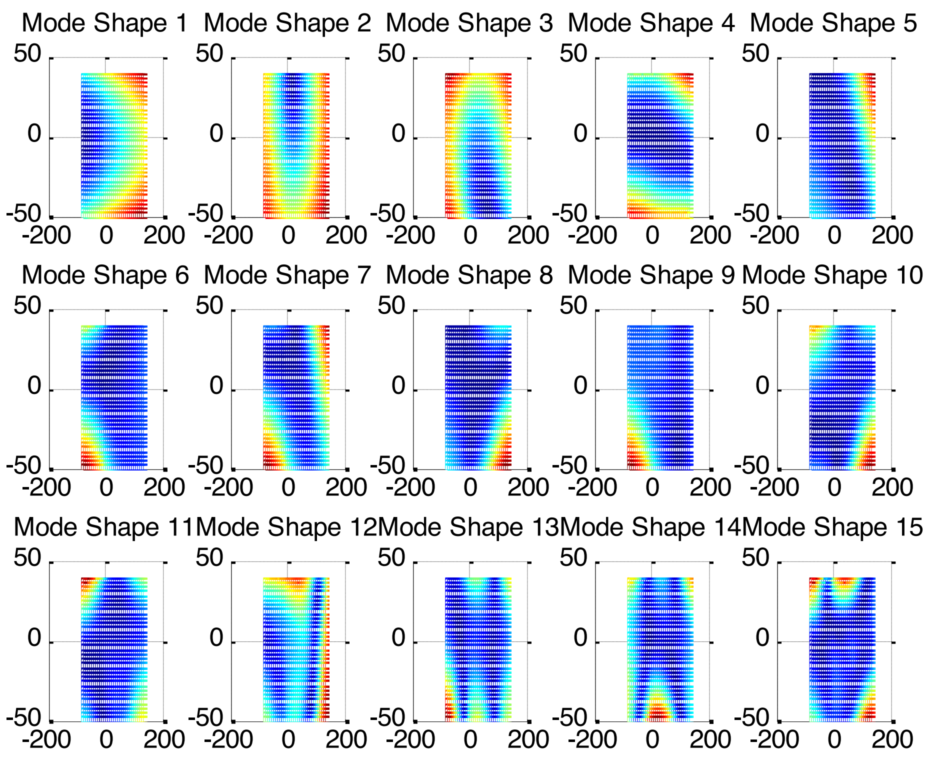
The last step is saving the obtained surface in image format, which can be done by the command sequence:

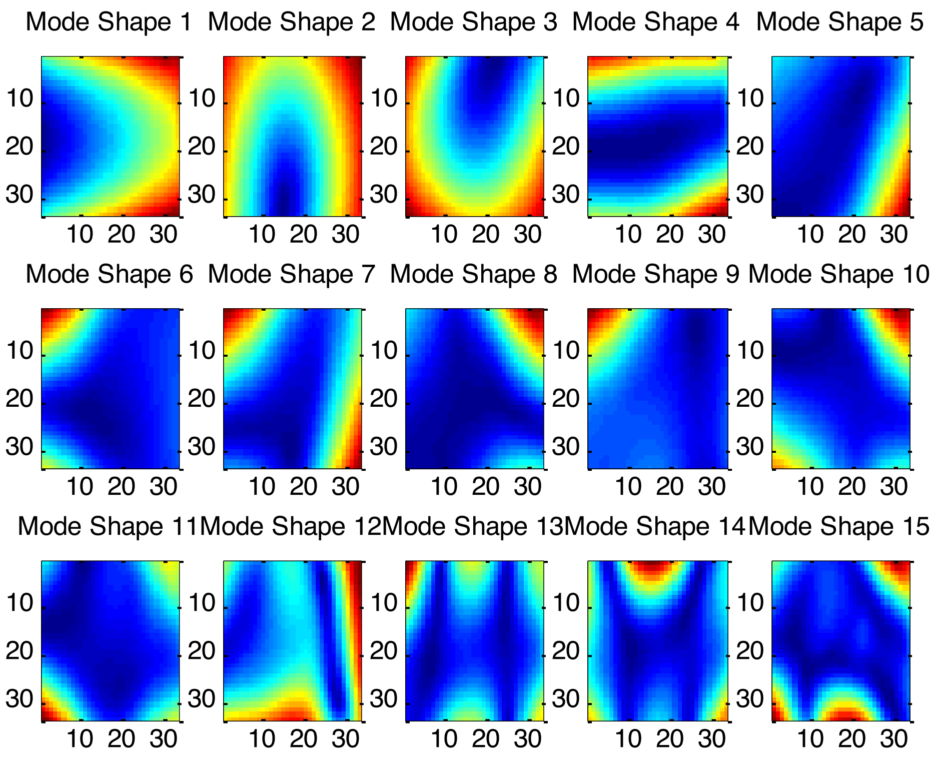
Resulting, for the Mode 1 example, in:

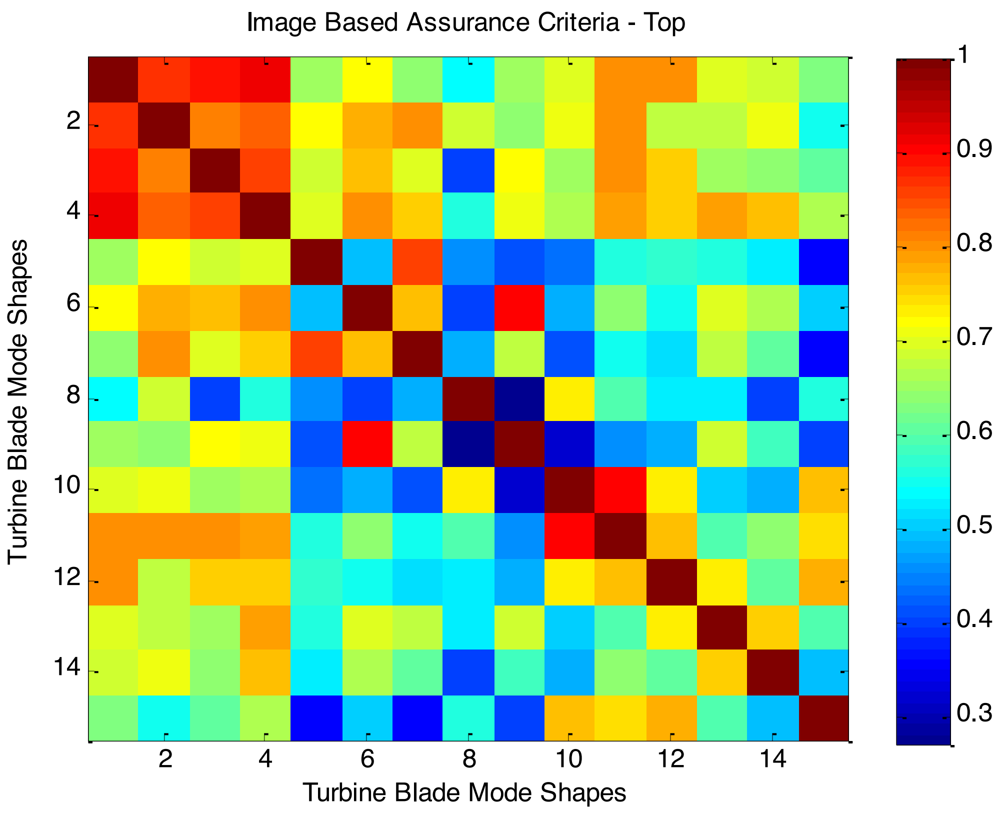


Finally, it was possible to apply the MAC in both data sets, top (number 1) and lateral (number 2), obtaining the following results:

4.1 Results for Data Set 1 (Top)

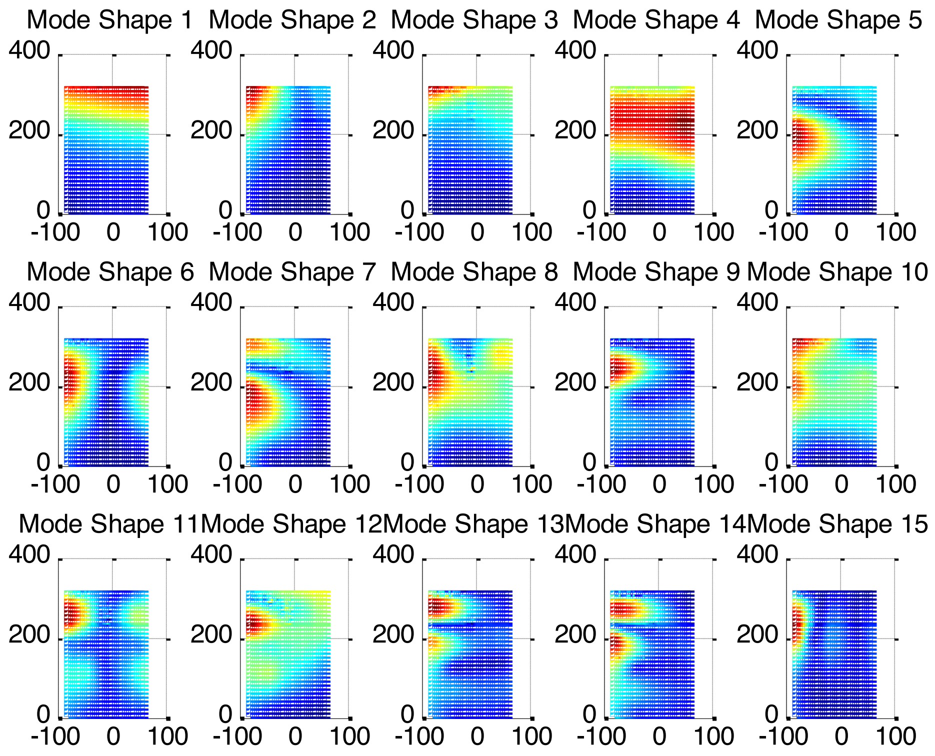




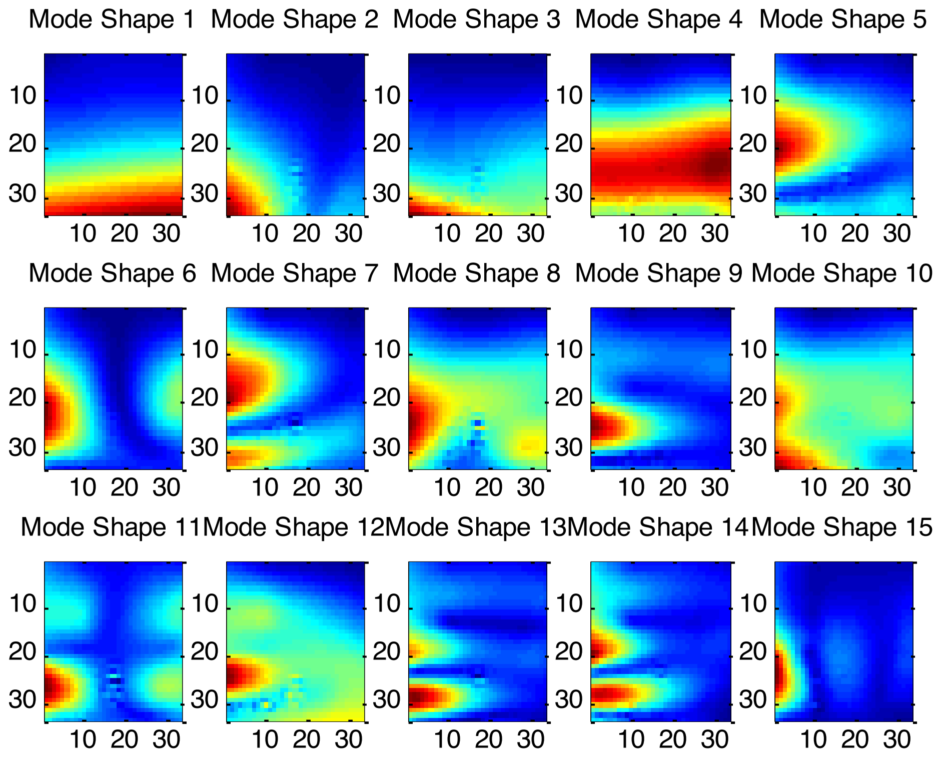


4.2 Results for Data Set 2 (Lateral)

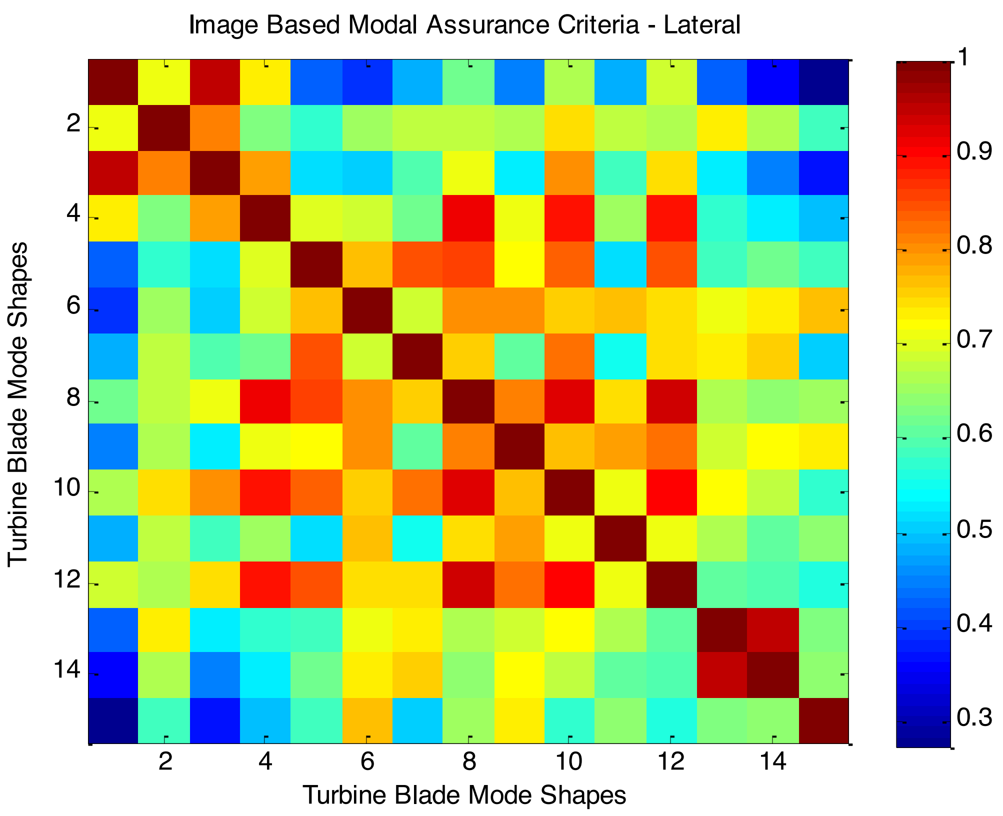
If we repeat the previous sequence for the 15 mode shapes available, we can obtain the desired 15 mode shape images. They can be presented as the resulting data mesh, for each mode shape:



Or as the image format obtained for each mode shape:



Using them, keeping in mind that each one of these image formats is a real matrix, we can use them as vectors to be fed as input for the modal assurance criteria. That will result in:



The obtained results give a general idea of the correlation level between two different mode shapes, showing, for example, the pair of modes (1, 15) as the worst correlation, which can be intuitively inferred from the mode shape images. It also shows a much deeper correlation between a greater quantity of mode pairs when compared to the classic method.

5. Conclusion

The developed assurance criterion enables a much deeper level of comparison between the mode shapes, as it shows various correlation degrees between the different mode shapes. It is possible then to have a more realistic idea of how interconnected the mode shapes are, added to a visualization that provides a common user of realizing this interconnection.

Conclusion

The mains conclusions that were apprehended from the three developed projects were:

- The curve description using the Fourier Transform method ensures reasonable results and ease to its user, therefore becoming a powerful tool to image reconstruction, when working with image processing. Further improvement might be done concerning methods for reconstruction of more general curves (open curves, for instance). This may need utilization of other image recognition methods, applying, for example, the methods which were mentioned in Chapter I, such as: Zernike Moment description, Wavelet Description. Using the developed method in mode shape recognition, it was simple to detect and reconstruct nodal lines, a useful feature to more detailed mode shape analyses.
- The developed function used in the plate experiment was very useful to determine the more wide and detailed approach that consists in using each mode shape extracted image format as input data for comparison in modal assurance criterion. This image based criterion shows mush more subtle differences between the mode shapes, ensuring a detailed correlation between them. Despite not showing the exact differences between a given mode shape pair, which can be done by visualization, this method also points towards a more direct way to mathematically establish an intimate correlation between two given images.
- When applying the method to turbine blade mode shapes, it was possible to see a result set that was very much alike the results obtained for the plate experiment. We may also infer from these results that a damaged turbine blade will also present a measurable mode shape correlation variation, as pointed by the results for the sane/damaged plate experiment. This can be used, for instance, in detecting damaged turbine blades, and also in establishing levels for tolerated damage. Further improvement may have to be done in specifying damage location methods.

<u>**References**</u>

[1] Avitabile, Peter "Experimental Modal Analysis, A Simple Non-Mathematical Presentation", University of Massachusetts Lowell, Lowell, Massachusetts

[2] Weizhuo Wang, John E Mottershead, and Cristinel Mares, **"**Mode-shape recognition and finite element model updating using the Zernike moment descriptor", Journal of Mechanical Systems and Signal Processing

[3] Britannica Concise Encyclopedia. 1994-2008 Encyclopædia Britannica, Inc. , accessed in July 7th, 2010.

[4] R. L. Cosgriff, "Identification of Shape," Report No 820-11 of the Ohio State University Research Foundation1960.

[5] C. T. Zahn and R. Z. Roskies, "Fourier Descriptors for Plane Closed Curves," *IEEE Transactions on Computers,* vol. C21, pp. 269-281, 1972.

[6] E. Persoon and K.-S. Fu, "Shape discrimination Using Fourier Descriptors," *IEEE Transactions on Systems Man and Cybernetics,* vol. SMC-7, pp. 170-179, 1977.

[7] Weizhuo Wang, John E Mottershead, and Cristinel Mares, "Shape descriptors for mode-shape recognition and model updating".

[8] Mark S. Nixon, Alberto S. Aguado "Feature Extraction and Image Processing Second edition" Copyright © 2008 Elsevier Ltd.

[9] C. H. Teh and R. T. Chin, "On image analysis by the methods of moments," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 10, pp. 496-513, 1988.

[10] R. J. Prokop and A. P. Reeves, "A survey of moment-based techniques for unoccluded object representation and recognition," *Graphical Models Image Processing,* vol. 54, pp. 438-460, 1992.

[11] A. Khotanzad and Y. H. Hong, "Invariant image recognition by zernike moments," *IEEE transaction of Pattern Analysis and Machine Intelligence,* vol. 12, pp. 489-498, 1990.

[12] I. Daubechies, *Ten Lectures on Wavelets*: SIAM: Society for Industrial and Applied Mathematics,1992.

[13] I. Lopez, W. Rooyakkers, & R. Vijgen, "Experimental Modal Analysis Of A Turbine Blade" , Eindhoven, December, 2004

[14] Allemang, Randall J., The Modal Assurance Criterion – Twenty Years of Use and Abuse, University of Cincinnati, Cincinnati, Ohio

[15] A. K. Jain, R. P. W. Duin, and J. Mao, "Statistical Pattern Recognition: A Review," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 22, pp. 4-37, 2000.

[16] L. d. F. Costa and J. R. M. Cesar, *Shape Analysis and Classification Theory and Practice*: CRC Press, 2001.

[17] R. J. Allemang and D. L. Brown, "A correlation coefficient for modal vector analysis," *Proceeding of 1st International Modal Analysis Conference,* vol. 1, pp. 110-116, 1982.

[18] Buhmann, Martin D. (2003), *Radial Basis Functions: Theory and Implementations*, Cambridge University Press, ISBN 978-0-521-63338-3.

[19] J. L. Rodgers and W. A. Nicewander. Thirteen ways to look at the correlation coefficient. The American Statistician, 42(1):59–66, February 1988.

Appendix

A1 - Function rbfcheck.m

```
unction maxdiff = rbfcheck(options)
ic;
odes   = options.('x');
 y     = options.('y');
= rbfinterp(nodes, options);

printf('RBF Check\n');
printf('max|y - yi| = %e \n', max(abs(s-y)) );

f (strcmp(options.('Stats'),'on'))
   fprintf('%d points were checked in %e sec\n', length(y), toc);
nd;
printf('\n');
```

A2 - Function rbfcreate.m

```
on options = rbfcreate(x, y, varargin)
EATE Creates an RBF interpolation
TIONS = RBFSET(X,Y,'NAME1',VALUE1,'NAME2',VALUE2,...) creates an
dial base function interpolation

FCREATE with no input arguments displays all property names and their possible
s.

EATE PROPERTIES

 Chirokov, alex.chirokov@gmail.com
eb 2006

t out possible values of properties.

rgin == 0) & (nargout == 0)
ntf('            x: [ dim by n matrix of coordinates for the nodes ]\n');
ntf('            y: [   1 by n vector of values at nodes ]\n');
ntf('  RBFFunction: [ gaussian  | thinplate | cubic | multiquadrics |
r} ]\n');
ntf('  RBFConstant: [ positive scalar     ]\n');
ntf('    RBFSmooth: [ positive scalar {0} ]\n');
ntf('        Stats: [ on | {off} ]\n');
ntf('\n');
rn;

= [
BFFunction      '
BFConstant      '
BFSmooth        '
tats            '
```

```
es = lower(Names);

ions = [];
 j = 1:m
ptions.(deblank(Names(j,:))) = [];


*************************************************************************
eck input arrays
*************************************************************************
Dim nXCount]=size(x);
Dim nYCount]=size(y);

(nXCount~=nYCount)
rror(sprintf('x and y should have the same number of rows'));
;

(nYDim~=1)
rror(sprintf('y should be n by 1 vector'));
;

ions.('x')           = x;
ions.('y')           = y;
*************************************************************************
fault values
*************************************************************************
ions.('RBFFunction') = 'linear';
ions.('RBFConstant') = (prod(max(x')-min(x'))/nXCount)^(1/nXDim);
prox. average distance between the nodes
ions.('RBFSmooth')   = 0;
ions.('Stats')       = 'off';

*************************************************************************
rgument parsing code: similar to ODESET.m
*************************************************************************

 1;
 finite state machine to parse name-value pairs.
rem(nargin-2,2) ~= 0
rror('Arguments must occur in name-value pairs.');

ectval = 0;                          % start expecting a name, not a
ue
le i <= nargin-2
rg = varargin{i};

f ~expectval
 if ~isstr(arg)
error(sprintf('Expected argument %d to be a string property name.', i));
 end

 lowArg = lower(arg);
 j = strmatch(lowArg,names);
 if isempty(j)                       % if no matches
   error(sprintf('Unrecognized property name ''%s''.', arg));
 elseif length(j) > 1                % if more than one match
```

```
     k = strmatch(lowArg,names,'exact');
     if length(k) == 1
       j = k;
     else
       msg = sprintf('Ambiguous property name ''%s'' ', arg);
       msg = [msg '(' deblank(Names(j(1),:))];
       for k = j(2:length(j))'
         msg = [msg ', ' deblank(Names(k,:))];
       end
       msg = sprintf('%s).', msg);
       error(msg);
     end
   end
   expectval = 1;                      % we expect a value next

 else
   options.(deblank(Names(j,:))) = arg;
   expectval = 0;
 end
 i = i + 1;
nd

f expectval
 error(sprintf('Expected value for property ''%s''.', arg));
nd



************************************************************************
 Creating RBF Interpolation
************************************************************************

witch lower(options.('RBFFunction'))
     case 'linear'
       options.('rbfphi')   = @rbfphi_linear;
     case 'cubic'
       options.('rbfphi')   = @rbfphi_cubic;
     case 'multiquadric'
       options.('rbfphi')   = @rbfphi_multiquadrics;
     case 'thinplate'
       options.('rbfphi')   = @rbfphi_thinplate;
     case 'gaussian'
       options.('rbfphi')   = @rbfphi_gaussian;
   otherwise
       options.('rbfphi')   = @rbfphi_linear;
nd

hi        = options.('rbfphi');

=rbfAssemble(x, phi, options.('RBFConstant'), options.('RBFSmooth'));

=[y'; zeros(nXDim+1, 1)];

%inverse
bfcoeff=A\b;
```

```
,S,V] = svd(A);

r i=1:1:nXCount+1
  if (S(i,i)>0) S(i,i)=1/S(i,i); end;
d;
fcoeff = V*S'*U*b;

ons.('rbfcoeff') = rbfcoeff;



strcmp(options.('Stats'),'on'))
fprintf('%d point RBF interpolation was created in %e sec\n', length(y),
;
fprintf('\n');


tion [A]=rbfAssemble(x, phi, const, smooth)
 n]=size(x);
ros(n,n);
i=1:n
for j=1:i
    r=norm(x(:,i)-x(:,j));
    temp=feval(phi,r, const);
    A(i,j)=temp;
    A(j,i)=temp;
end
A(i,i) = A(i,i) - smooth;

lynomial part
nes(n,1) x'];
[ A      P
  P' zeros(dim+1,dim+1)];

*************************************************************************
dial Base Functions
*************************************************************************
tion u=rbfphi_linear(r, const)


tion u=rbfphi_cubic(r, const)
*r.*r;

tion u=rbfphi_gaussian(r, const)
p(-0.5*r.*r/(const*const));

tion u=rbfphi_multiquadrics(r, const)
rt(1+r.*r/(const*const));

tion u=rbfphi_thinplate(r, const)
*r.*log(r+1);
```

```
ic;
hi        = options.('rbfphi');
bfconst   = options.('RBFConstant');
odes      = options.('x');
bfcoeff   = (options.('rbfcoeff'))';
```

A3 - Function `rbfinterp.m`

```
dim            n] = size(nodes);
dimPoints  nPoints] = size(x);

f  (dim~=dimPoints)
 error(sprintf('x should have the same number of rows as an array used
o create RBF interpolation'));
nd;

 = zeros(1, nPoints);
 = zeros(1, n);

or i=1:1:nPoints
   s=0;
   r =  (x(:,i)*ones(1,n)) - nodes;
   r = sqrt(sum(r.*r, 1));
     for j=1:n
          r(j) =  norm(x(:,i) - nodes(:,j));
     end

    s = rbfcoeff(n+1) + sum(rbfcoeff(1:n).*feval(phi, r, rbfconst));

   for k=1:dim
      s=s+rbfcoeff(k+n+1)*x(k,i);     % linear part
   end
   f(i) = s;
nd;

f  (strcmp(options.('Stats'),'on'))
   fprintf('Interpolation at %d points was computed in %e sec\n',
ength(f), toc);
nd;
```

A4 - Function `macD.m`

```
;
;   mac
;
;   Computes modal assurance criteria
;
;
;   mc=mac(phi1,phi2)
;
;
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
; This matlab source code was originally     %
; developed as part of "DIAMOND" at          %
; Los Alamos National Laboratory. It may     %
; be copied, modified, and distributed in    %
; any form, provided:                        %
;  a) This notice accompanies the files and  %
;     appears near the top of all source     %
;     code files.                            %
;  b) No payment or commercial services are  %
;     received in exchange for the code.     %
;                                            %
; Original copyright is reserved by the      %
; Regents of the University of California,   %
; in addition to Scott W. Doebling, Phillip  %
; J. Cornwell, Erik G. Straser, and Charles  %
; R. Farrar.                                 %
;%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ns,n]=size(t1);[ns1,n1]=size(t2);
ıs=min([ns,ns1]);
.f nargin < 3, Q=eye(ns,ns); end
:1=t1(1:ns,:);t2=t2(1:ns,:);mc=zeros(n,n1);
for i=1:n,
  for j=1:n1,

ıc(i,j)=(t1(:,i)'*Q*t2(:,j))^2/(t1(:,i)'*Q*t1(:,i)*t2(:,j)'*Q*t2(:,j));
  end
end
return
```