ROBUSTNESS MEASURES FOR SIGNAL DETECTION IN NON-STATIONARY NOISE USING DIFFERENTIAL GEOMETRIC TOOLS

A Dissertation

by

GUILLAUME JULIEN RAUX

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

December 2006

Major Subject: Electrical Engineering

ROBUSTNESS MEASURES FOR SIGNAL DETECTION IN NON-STATIONARY NOISE USING DIFFERENTIAL GEOMETRIC TOOLS

A Dissertation

by

GUILLAUME JULIEN RAUX

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Approved by:

Chair of Committee,	Don R. Halverson
Committee Members,	Deepa Kundur
	Jim Ji
	Yassin Hassan
Head of Department,	Costas N. Georghiades

December 2006

Major Subject: Electrical Engineering

ABSTRACT

Robustness Measures for Signal Detection
in Non-Stationary Noise
Using Differential Geometric Tools. (December 2006)
Guillaume Julien Raux, B.S., West Virginia University;
M.S., West Virginia University
Chair of Advisory Committee: Dr. Don Halverson

We propose the study of robustness measures for signal detection in non-stationary noise using differential geometric tools in conjunction with empirical distribution analysis. Our approach shows that the gradient can be viewed as a random variable and therefore used to generate sample densities allowing one to draw conclusions regarding the robustness. As an example, one can apply the geometric methodology to the detection of time varying deterministic signals in imperfectly known dependent non-stationary Gaussian noise. We also compare stationary to non-stationary noise and prove that robustness is barely reduced by admitting non-stationarity. In addition, we show that robustness decreases with larger sample sizes, but there is a convergence in this decrease for sample sizes greater than 14.

We then move on to compare the effect on robustness for signal detection between non-Gaussian tail effects and residual dependency. The work focuses on robustness as applied to tail effects for the noise distribution, affecting discrete-time detection of signals in independent non-stationary noise. This approach makes use of the extension to the generalized Gaussian case allowing the comparison in robustness between the Gaussian and Laplacian PDF. The obtained results are contrasted with the influence of dependency on robustness for a fixed tail category and draws consequences on

residual dependency versus tail uncertainty.

To Mon grand-père Georges.

ACKNOWLEDGMENTS

I would like to thank the professors on my committee for their assistance and insight into my research. I would like to give special thanks to Dr. Halverson. I have enjoyed working with him and greatly appreciate the many opportunities he has made available to me. Thanks for your help and comments for the preparation of this dissertation.

Finally, on a personal note, I would like to thank all my friends and family that kept encouraging me all along the tedious process of writing this dissertation and to whom I dedicate this work.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

LIST OF TABLES

TABLE Page

LIST OF FIGURES

FIGURE Page

FIGURE Page

FIGURE Page

FIGURE Page

CHAPTER I

INTRODUCTION

> It is a capital mistake to theorize before you have all the evidence. It biases the judgement. —Arthur Conan Doyle

A. Historical

In recent years, robustness measures have been widely discussed. Many of the techniques used in today's communications, signal processing, and control systems applications rely on techniques for detecting various signal parameters. Furthermore, it has become evident that the degree of robustness associated with the parameter detector is an important factor that effects overall system performance. Evidence of increased attention given to robustness issues in signal processing applications is given in [1], which is an excellent paper of over 200 publications which investigate robustness issues.

In this dissertation we consider the detection of a real signal parameter based on data which we allow to be dependent, and either stationary or non-stationary. While an analysis is fairly straight forward for certain cases (e.g., the detection of a signal in independent identically distributed -i.i.d.- Gaussian data), the situation becomes far more involved when the data possess statistical distributions which are not well known. In particular, the sampling rate that is slow enough to yield independent data may be consistent with an assumption of absolute stationarity and vice versa. Hence the motivation to seek procedures that are robust to the inexact statistical knowledge increased.

The journal model is *IEEE Transactions on Information Theory.*

B. Saddlepoint Technique

For an algorithm to be successful, it must possess a degree of robustness to the inexact knowledge, i.e., the algorithm's performance should not be too sensitive to inexact statistical knowledge. Much past work has been performed in engineering robustness research by applications of Huber-Strassen saddlepoint techniques [2, 3]. Even though this technique still plays an important role in today's research, it contains major limitations (being inherently non-quantitative) which have inspired an alternative approach using differential geometric tools. This new direction of research allows engineers to combine both performance and robustness into the development of an algorithm.

Classically, the noise models were stationary Gaussian with a parametric assumption (all necessary parameters assumed known), but it was quickly realized that some relaxation of such assumptions was necessary. For example, while a Gaussian model might be useful as a first step, it is certainly desirable to allow the entries in its covariance matrix to be imperfectly known and to admit residual non-stationarity.

For these reasons, there has been sustained interest in detection algorithms that feature robustness. This could be in the context of the algorithm performing well under a Gaussian assumption but with imperfectly known covariance matrix and residual non-stationarity. Classical robustness analysis (e.g. see [4, 5]) has built off the Huber-Strassen saddlepoint approach, and many useful results have been obtained, both for detection theory and, for other domains (such as estimation). There are some limitations to the employment of Huber-Strassen, however. The method is not naturally quantitative; one obtains an algorithm as a solution of a saddlepoint equation which is by definition: "the" robust solution. Algorithms which are very close to the solution are simply not robust, and there is not a natural measure of

how close they are to being robust. Secondly, the adaptation by practitioners in such fields as engineering are bound by Huber's concept of robustness as a subject of interest to statisticians who focused on outliers. For many practitioners, this may not be what they mean by robustness, since outliers might not be the chief problem. Third, the saddlepoint method restricts the type of uncertainty admitted through canonical models such as ϵ-contamination, and finally, the method resists admitting non-stationarity and dependency.

As a consequence, a new approach toward measuring robustness has been developed by Halverson, et. al (see examples, [6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]). This work is naturally quantitative and views the focus of robustness not on outliers but on perturbations away from a nominal. This may be of more interest to practitioners in fields such as engineering, where one first often makes a "seat of the pants" guess (i.e., to choose nominal values) and requires an algorithm that will tolerate an imperfect guess. In addition, the work readily admits non-stationarity and dependency. The general idea of this work is that the imperfectly known quantity (parameter vector or distribution) is allowed to vary about its nominal over a differentiable manifold which models a local "neighborhood" about the nominal. As the variation occurs, the performance (e.g., false alarm rate or detection probability for detectors, mean L_p error for estimators) changes. The greater the change: the less robustness; thus gradient provides a convenient measure. Early work [10, 13, 21, 22] focused on local robustness very close to the nominal, and so gradient at the nominal was a key element in robustness measure. Later work [6, 11] allowed for non-local neighborhoods, with [11] even allowing the robustness to be computed as a hybrid between the local and the non-local concepts, thus admitting an emphasis on outliers *a la Huber* if desired. While gradient was commonly employed, there was also consideration to context where gradient was insufficient, and the second-order measure

offered by Gaussian curvature was applied [8, 12, 23]. All of this later work features Euclidian models for the parameter manifold.

Recent work has explored the admission of non-Euclidian models. The reason for doing so is not to do more complex mathematics for its own sake, but to better model reality. For example, if a covariance matrix is imperfectly known, one approach would be to model this by simply allowing each entry to vary by, say, $\pm\delta$. If the nominal matrix is positive definite, then for sufficiently small δ the actual matrix will stay positive definite. But how small? This Euclidian model thus puts limits, possibly severe, on how much variation can be considered. But, a practitioner might need to take into account large variations. A Euclidian model will thus fail: it will admit matrices which are not positive definite and thus cannot be potential covariance matrices. The best approach would be to *impose* positive definiteness *a priori*, resulting in a non-Euclidian structure. Naturally, employing a non-Euclidian Riemannian manifold complicates the mathematics, but it provides a much more versatile tool to address the needs of the user. It is also a feasible and tractable approach; for example, in [8, 23] both gradient and curvature were employed to measure robustness for linear estimation using non-Euclidian model with a "biased" perturbation interpretation. In [9] the same general problem was expanded using gradient to include an "unbiased" perturbation interpretation, and this was compared to the "biased." In [9], the same general problem was expanded using gradient to include an "unbiased" perturbation interpretation, and this was compared to the "biased." Our proposed work will, for the first time, direct the non-Euclidian model toward detection theory.

C. Assumptions and Goals

In this dissertation, a variety of assumptions apply. These include:

- Constant additive signal in dependent noise
- Noise jointly Gaussian, later to be extended to specific non-Gaussian, such as Laplace
- Linear detector -of practical interest, e.g., the matched filter
- Covariance matrix entries lie on a non-Euclidian Riemannian differentiable manifold (we consider two canonical choices for examples)
- Gradient the measure of variability
- Sufficient smoothness conditions on all functions so that required operations (such as derivatives) exist
- Perturbations modeled by either the "biased" or "unbiased" approach of [9, 24]
- Robustness of the false alarm rate is studied, although the method easily extends to detection probability
- When comparing examples possessing parameter manifold of different dimensions, such as when comparing the stationary to the non-stationary, the manifolds are of comparable size if their integrated Gaussian curvature per unit volume is invariant

The goals of this dissertation include:

- To illustrate how non-Euclidean geometry can be applied toward measuring robustness of systems of engineering interest. While we specifically illustrate an application for signal detection with a matched filter, our methods will be seen to have broad potential application

- To compare the biased versus unbiased approach of [9, 24] to see if they yield similar results. If so, then the biased approach might be appealing because it is easier to compute
- To compare the robustness of the detector for stationary data versus non-stationary. Is there a loss of robustness due to non-stationarity?
- What is the effect of increasing sample size on robustness?
- What is the effect of a specific non-Gaussian noise (Laplace for example)?

D. Applications

Detection theory is an area of classical interest dating back at least to World War II. Simply put, it employs classical hypothesis testing to make a "signal with noise" or "noise only" decision. The signal can be random or not, but the noise is modeled as a random process. The "signal and noise" are not limited to electromagnetic phenomena, but can be interpreted as economic or medical trends – we are really referring simply to a choice between two hypotheses. Applications of classical interest include sonar (naval warfare; oil exploration); radar (military; remote sensing) and more recently, radio astronomy. In addition, there is now renewed interest in the area as part of the security arena (airline, explosive and pathogen detection).

E. Purpose and Overview of this Dissertation

Consider a performance function $P : R^m \rightarrow R$. For example, the covariance matrix entries may lie in a subset of R^m, and P might be the least squares error of a linear estimator employed with an imperfectly known variance matrix. Using the techniques described in [13, 10, 21, 22], one can easily associate the robustness of an algorithm

with changes in P as one moves away from the nominal point in R^m. The most convenient way to do this locally is to look at the rate of change of P as one moves in the manifold M spanned by the imperfectly known quantities. While potentially misleading, the use of slope is convenient and has been found to be a good indicator of the actual performance in many situations where the more sensitive additional measure provided by curvature [8, 12, 23] is not necessary.

As an example of the application of the above approach, consider applications which involve Gaussian-distributed data. The Gaussian covariance matrix is crucial to an analysis of such an application, but it is unrealistic to assume we will know the matrix perfectly. After all, why would mother nature be so kind to us? The set of possible entries for the matrix is imperfectly known but its constraint, such as being positive definite, might be known. This and other possible constraints will result in the entries lying in a manifold which may very well be non-Euclidian. More detail of this proposed work is presented in chapter III.

Now, we remark that the work in chapter III will employ gradient to generate a quantitative robustness measure and will also be based on signal detection. Chapter III will therefore be dedicated to the applications and tools of differential geometry to robustness while chapter II will be dedicated to detection theory and will also include discussion of applications, historical context, assumptions, and will provide more detailed comments of some of the elements of this introduction. Chapter IV will be dedicated to the extension of the previous case scenario using 3 samples. Extending the previous approach from 2 to 3 samples allows one to observe the effect of stationarity on robustness. Chapter V will be dedicated to generate a quantitative robustness measure based on signal detection for non-Gaussian noise for the 2 sample case scenario. This will be done using an extended Gaussian distribution and by varying the value of the exponential factor (allowing to deal with different noises).

Finally, chapter VI will present a complete summary of all the results obtained along this dissertation.

CHAPTER II

BACKGROUND AND FUNDAMENTALS

This chapter presents two main tasks in signal processing, which are detection and estimation of a signal leading to the making of a decision. It clearly shows how these two tasks are closely related and then moves on to make an in-depth presentation of the tools needed to form differential geometry for this research.

A. Estimation Theory in Signal Processing

In modern life, estimation theory can be found at the center of many electronic signal processing systems designed to extract *information*. The theory of estimation, originally developed within the area of statistics, is applied today in many different fields. A non-exhaustive list of the systems that incorporate this purpose is given as follows: radar, speech, sonar, communications, seismology, image analysis, etc..

All of these applications share the same common function of needing to estimate the values of a group of parameters. For example, within a radar system one might be interested in determining the position of an aircraft. To accomplish this, the transmission of an electromagnetic pulse is reflected off of the aircraft causing the antenna to receive an echo a few moments later. Clearly, if the the round trip delay can be measured, one can easily compute the distance between the plane and the antenna, even though the echo is decreased in amplitude due to the propagation losses and the corruption of the signal induced by the channel.

Another example is speech processing systems, such as speech recognition. This systems corresponds to the recognition of speech by a device like a computer. One of the many tasks the computer can execute is trying to recognize speech sounds like vowels or consonants. In order to do so, a computer tries to compare the spoken

vowel with waveforms stored in memory and then chooses the closest one to the spoken vowel, the one that minimizes the *distance measure*. The main difficulty here is when the pitch of the speakers's voice differs from the voice recorded during the training session.

Due to this problem inherent to the wave forms, one should try to choose different attributes that are less susceptible to variations. For example the spectral envelope will not change with the pitch since the Fourier transform of a periodic signal is a sampled version of the Fourier transform of one of the period of the signal. The period only affects the spacing between frequency samples, not the values.

In all of these systems, one faces the problem of the *extraction* of the parameter values based on *continuous* waveforms. The same problem arises in the use of digital computer. The equivalent problem is to extract different parameter values from a *discrete-time* waveform or a data set. Mathematically, we now have a N-point data set $\{x[0], x[1], ..., x[N-1]\}$ which depends on an unknown parameter θ.

If one wishes to determine θ based on the data set or to define an *estimator*, he will obtain:

$$\widehat{\theta} = g(x[0], x[1], ..., x[N-1]) \tag{2.1}$$

where g is some function.

One of the first people to address this kind of problem was Gauss in 1795 with the use of least squares data analysis to predict the movement of the planets (see [25]).

1. The Mathematical Estimation Problem

In order to determine good estimators the first step is to mathematically model the data. The data being random, we can describe it by its *probability density function*

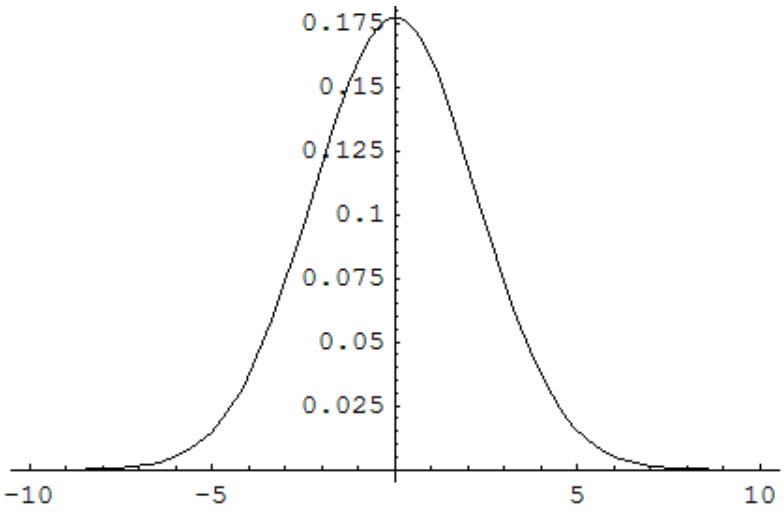

Fig. 1. Gaussian distribution with zero mean

(PDF). As such, the performance of the estimator can only be completely described *statistically*. The unknown parameter θ *parameterized* the PDF, i.e., within the class of PDFs each PDF is different due to the different value of θ.

For example, if N=1 and θ denotes the mean of a Gaussian random variable, then the PDF might be:

$$p(x[0];\theta) = \frac{1}{\sqrt{2\pi\sigma^2}} exp[-\frac{1}{2\sigma^2}(x[0]-\theta)^2] \tag{2.2}$$

which is shown in figure 1 for $\theta = 0$ and figure 2 for $\theta = 5$. The x-axis corresponds to x[0] and the y-axis corresponds to $p(x[0];\theta)$.

It is easy to realize that the value of the mean affects the probability of x[0], and consequently the specification of the PDF is critical in determining a good estimator.

In real life problems, we are not given a PDF but instead we have to choose one that is not only consistent with the problem constraints and any prior knowledge, but one that is also mathematically tractable.

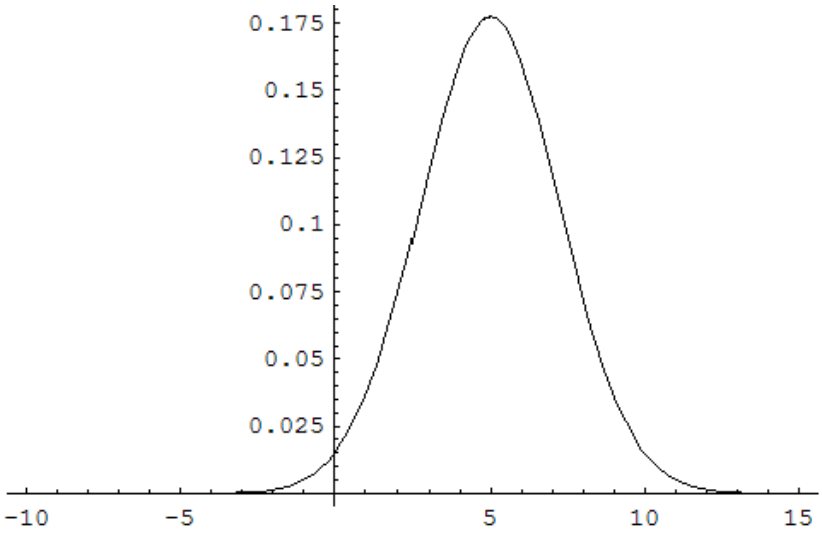

Fig. 2. Gaussian distribution with $\mu = 5$

2. Example

As an example, consider the DC level in White Gaussian Noise (WGN). Let's consider the observations: x[n]=A+w[n] where n=0, 1,...,N-1, where A is the parameter to be estimated. A reasonable model for w[n] is WGN or each sample of w[n] has the PDF $N \sim (0, \sigma^2)$ which denotes a Gaussian distribution with a mean of 0 and a variance of σ^2 and is uncorrelated with the other samples. Thus, the PDF becomes:

$$p(x[n];\theta) = \frac{1}{(2\pi\sigma^2)^{\frac{N}{2}}} exp[-\frac{1}{2\sigma^2}\sum_{n=0}^{N-1}(x[n]-A)^2] \tag{2.3}$$

Therefore a reasonable estimator for the average value of x[n] is:

$$\widehat{A} = \frac{1}{N}\sum_{n=0}^{N-1} x[n] \tag{2.4}$$

The performance of the estimator is critically dependent on the PDF assumptions. One can only hope that the estimator is *robust*, in that slight changes in the PDF do not severely affect the performance of the estimator.

An estimation based on PDFs is described as *classical* estimation because the parameters of interest are unknown but *deterministic*. When one incorporates the prior knowledge of the PDF, therefore assuming that the parameter is no longer deterministic, such an approach is called *Bayesian* estimation. The parameter we are attempting to to estimate is then viewed as a *realization* of the random variable θ. The data are now described by the *joint* PDF:

$$p(x;\theta) = p(x|\theta)p(\theta) \tag{2.5}$$

where $p(\theta)$ is the prior PDF, summarizing the knowledge about θ before any data are observed, and $p(x|\theta)$ is the conditional PDF, summarizing our knowledge provided by the data x conditioned on knowing θ.

Once the PDF has been specified, the problem is to determine an optimal estimator or a function of the data, knowing that the estimator may depend on other parameters. An estimator may be thought of as a *rule* that assigns a value to θ for each realization of x. The *estimate* of θ is the value of θ obtained for a *given* realization of x (see [26]).

B. Detection Theory in Signal Processing

Modern *detection theory* is fundamental to the design of electronic signal processing systems for *decision making* and the extraction of the information. Detection theory is mostly employed in weak signal and/or high noise situations where a decision must be made as to whether a noise-corrupted signal is present [27].

The making of the decision is based on the extraction of the information out of the available data [27]. The mathematical techniques needed to extract as much information from the data will provide the necessary interface between the need to

make a detection decision and the desire to optimize the accuracy of the decision. Ultimately the goal is to use the data as efficiently and accurately as possible.

Most of the results referenced above deal with the case where the signals are deterministic or of known form with random parameters. There are, however, applications in which non-homogeneous propagation media or incomplete knowledge of the signal source invalidates such an assumption regarding the signal.

In these cases, the signal would be best modeled as a random process. There has been a great amount of effort expended in this area; for example, the Gaussian signal case has been considered in many publications like [27], as has the detection of a random signal in Gaussian noise.

1. The Detection Problem

One of the simplest detection problems is to decide whether a signal embedded in noise is present, or if only the noise is present. An example of this problem would be the detection of a plane based on the signal received by a radar. This type of problem is called a *binary hypothesis testing problem*, since one needs to decide between two possible outcomes, signal plus noise present versus just noise present.

$$\begin{aligned} \mathcal{H}_0 &: \textit{signal not present} \\ \mathcal{H}_1 &: \textit{signal present} \end{aligned}$$

The detection decision then reduces to the choice of $\mathcal{H}_0$ and $\mathcal{H}_1$. In order to decide between $\mathcal{H}_0$ and $\mathcal{H}_1$, we have to rely on the available data, which may be collected either continuously or discretely. In the discrete time case, the decision is

between:

$$\begin{aligned} \mathcal{H}_0 &: F_0(.,...,.) \\ \mathcal{H}_1 &: F_1(.,...,.) \end{aligned}$$

where the indicated point distributions F_0 and F_1 apply to n samples and traditionally correspond respectively to "noise only" and "signal with noise" situations.

Further discussion of the decision theoretic background for continuous time detection theory as applied to radar detection in Gaussian noise appears in literature such as [27]. One should note that the random process involved in the previous discussion has historically been modeled as Gaussian. However, much of the noise encountered in real-life situations is highly non-Gaussian. In these situations, continuous time detection often becomes mathematically intractable. Because of this, and also due to the increasing use of discrete time systems, we will limit the remainder of our consideration of detection theory to discrete time case.

Let's consider a case where the noise is deterministic. We will further limit consideration to the common case where the signal is additive to the noise. The detection problem then becomes:

$$\begin{aligned} \mathcal{H}_0 &: F_0(.,...,.) \\ \mathcal{H}_1 &: F_1(.,...,.) \end{aligned} \tag{2.6}$$

where F_0 corresponds to n noise samples and F_1 corresponds to each of these samples respectively added to the i-th signal value s_i. We observe realizations $\{y_i\}_{i=1}^n$ of the observation random process $\{Y_i\}_{i=1}^n$ and the s_i are known constants.

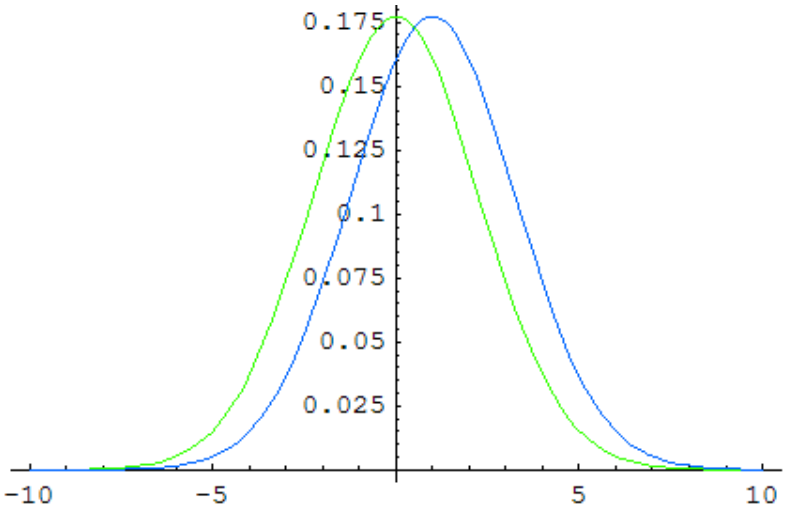

Fig. 3. Hypothesis testing: PDF of x[n] for signal present and signal absent

2. The Mathematical Detection Problem

The model of the detection problem has a form that will allow us to apply the theory of *statistical hypothesis testing* (see figure 3). As we did previously, one can consider, as an example, the detection of a DC level of amplitude A=1 embedded in WGN w[n] with variance σ^2. In order to simplify the discussion one can assume that we base the decision on the only sample available. Hence, one effectively needs to decide between the hypotheses x[0]=w[0] (noise) and x[0]=A+w[0] (signal plus noise). Since the main assumption is to consider the noise to be zero mean, one will need to decide that the signal is present if x[0]>1/2 and if noise only is present x[0]<1/2 -since $E\left[x[0]\right] = 0$ if only noise is present and $E\left[x[0]\right] = A$ if noise and signal are present-.

Clearly there will be an error whenever a signal is present and w[0]<1/2 or whenever noise only is present and w[0]>1/2. We cannot expect to make the correct decision all the time, but hopefully we can maximize the number of times that we make a good decision. The performance of any detector will depend upon how different the PDFs of x[0] are under each hypothesis. Multiple studies have already proven that

the detection performance improves as the "distance" between the PDFs increases or as A^2/σ^2, also called the signal-to-noise ratio (SNR), increases (see [27]).

This study illustrates the basic result that the detection performance depends on the *discrimination* between the two hypotheses or equivalently between the two PDFs. More formally, we model the detection problem as one of choosing between $\mathcal{H}_0$, which is termed the noise-only hypothesis, and $\mathcal{H}_1$, which is the signal present hypothesis.

The PDfs under each hypothesis are denoted by $p(x[0];\mathcal{H}_0)$ and $p(x[0];\mathcal{H}_1)$, which are for example:

$$p(x[0];\mathcal{H}_0) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}x[0]^2) \tag{2.7}$$

and

$$p(x[0];\mathcal{H}_1) = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}(x[0]-A)^2) \tag{2.8}$$

Ultimately, one will need to decide between the two PDFs as to which one is going to minimize the probability of making an error or equivalently maximize the probability of making a right decision.

Most of the results referenced above deal with the case where the signals are deterministic or of known form with random parameters. There are, however, applications in which non-homogeneous propagation media or incomplete knowledge of the signal source invalidates such an assumption regarding the signal. Such applications arise, for example, in sonar detection, radio astronomy, and seismic exploration.

In this cases, the signal would be best modeled as a random process. There has been a great amount of effort expended in this area; for example, the Gaussian signal case has been considered in many publications like [28, 29, 30], as has the detection of a random signal in Gaussian noise.

3. Detector Fidelity

In order to formalize the notion of fidelity of a detector, it has been found useful to employ two quantities, denoted by α and β, and defined by:

$$\begin{aligned} \alpha &= \textit{probability of choosing } \mathcal{H}_1 \textit{ when } \mathcal{H}_0 \textit{ is true.} \\ \beta &= \textit{probability of choosing } \mathcal{H}_1 \textit{ when } \mathcal{H}_1 \textit{ is true.} \end{aligned} \tag{2.9}$$

Classically, α has been called the "false alarm probability," and β the "detection probability". Ideally, one would like a detector to have the lowest α possible and the highest β possible. The problem is that there is a tradeoff between the two.

For example, if one designs a detector in an effort to make α small, it will bias the decision more toward $\mathcal{H}_0$ than for a detector designed to operate at a higher α level, with the result that β is reduced.

In view of this, there are several approaches which may be taken to formalize the notion of fidelity of a detector. For example, one could employ the *probability of error criterion* (minimize the probability of error) [31], the *Bayes criterion* (minimize the "average risk" associated with a weighted cost formulation) [32], the *minimax criterion* (minimize the maximum risk) [33], or finally the *Neyman-Pearson criterion* (constrain α and maximize β) [34].

One can easily remark that detectors designed under the first three criteria still possess an associated α and β; it can also be shown that many detectors reduce to the form of a Neyman-Pearson detector [35]. Also, α is usually associated with some sort of cost, and thus it is often desirable to constrain α to be no greater than some small value (e.g.$\alpha \leq 0.05$ or 5%). For this reason, the Neyman-Pearson criterion is especially popular and will be of immediate interest here.

The next step is to, when possible, design a detector with a maximal β for a

constrained α. In view of the Neyman-Pearson lemma [36], the optimal detector takes the following form:

$$\Lambda(y_1, ..., y_n) \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{2.10}$$

where Λ is a function of the observations and T is a deterministic constant. That is, the optimal detector chooses $\mathcal{H}_1$ if $\Lambda(y_1, ..., y_n) > T$, and chooses $\mathcal{H}_0$ if $\Lambda(y_1, ..., y_n) < T$. The Neyman-Pearson also gives the form of Λ. If P_i denotes the *probability measure* induced by the $Y_1, ..., Y_n$ under $\mathcal{H}_1$, and μ is the *dominating measure*, then

$$\Lambda = \frac{dP_1}{d\mu} / \frac{dP_0}{d\mu} \tag{2.11}$$

In this case the joint densities $P_i(y_1, ..., y_n)$ of $Y_1, ..., Y_n$ under $\mathcal{H}_i$ exist, then we have the more recognizable form.

$$\Lambda(y_1, ..., y_n) = \frac{P_1(y_1, ..., y_n)}{P_0(y_1, ..., y_n)} \tag{2.12}$$

This form is called the *likelihood function*. In certain cases, the form of the Neyman-Pearson detector simplifies. For example, if the noise process is first order stationary and "white", we then have, since $p_1(y_1, ..., y_n) = p_0(y_1 - s_1, ..., y_n - s_n)$,

$$\Lambda(y_1, ..., y_n) = \frac{p_0(y_1 - s_1)p_0(y_2 - s_2)...p_0(y_n - s_n)}{p_0(y_1)p_0(y_2)...p_0(y_n)} \tag{2.13}$$

where $p_0(.)$ denotes the univariate density of N_1. If we compare the above to the threshold value T, and take a natural logarithm of each side of the inequality, the Neyman-Pearson test becomes:

$$\sum_{i=1}^{n} \Big[ln\Big(p_0(y_i - s_i)\Big) - ln\Big(p_0(y_i)\Big)\Big] \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} \widehat{T} \tag{2.14}$$

where $\widehat{T} = \ln T$. The optimal detector, which is now called the *log-likelihood* ratio, reduces to figure 4.

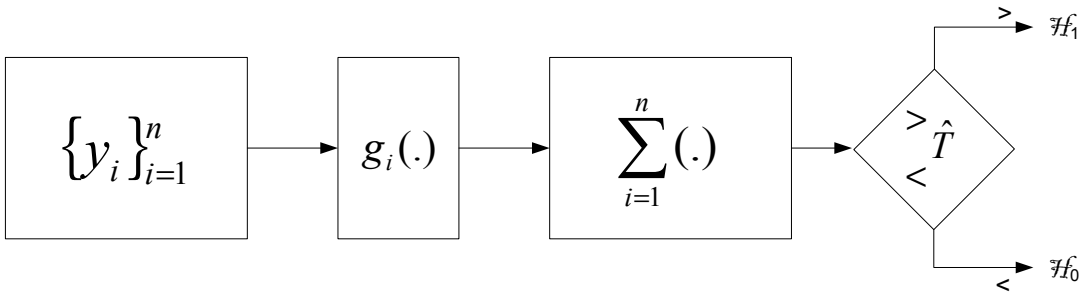

Fig. 4. Linear detector equivalent to a matched filter

where g_i is a time varying zero memory nonlinearity. If the noise is Gaussian, the nonlinearities g_i become linear, yielding a matched filter as in the continuous time case. Also, if the s_i are equal (the "sure" signal case), then the nonlinearity is time invariant.

If the noise is nonwhite, the Neyman-Pearson detector unfortunately is not of such a simple form as above. Because of modern, high speed sampling, dependency between the samples is often no longer negligible, and in these cases a Neyman-Pearson approach to detector design often becomes intractable because of inexact knowledge of the required n-th order densities of the noise, as well as inability to determine the statistics of the likelihood ratio Λ.

C. Introduction to Differential Geometry and General Relativity

We wish to remind the reader that differential geometry is a very abstract and challenging field. For a full understanding of those concepts, one should refer himself to the proper mathematical references (see [37] for example). However, in the interest of acquainting the interest of the reader, we now present a short heuristic description of differential geometry.

Differential Geometry is the language of modern physics as well as an area of

mathematical delight. Typically, one considers sets which are manifolds (that is, locally resemble Euclidean space) and which come equipped with a measure of distances. In particular, this includes classical studies of the curvature of curves and surfaces.

In mathematics, differential topology is the field dealing with differentiable functions on differentiable manifolds. It arises naturally from the study of the theory of differential equations. Differential geometry is the study of geometry using calculus. Together they make up the geometric theory of differentiable manifolds - which can also be studied directly from the point of view of dynamical systems.

1. Preliminaries: Distance, Open Sets, Parametric Surfaces and Smooth Function

In order to be able to speak about smooth manifolds, a review of the topology is necessary. A *n-dimensional Euclidian* space corresponds to:

$$E_n = (y_1, y_2, ...y_n | y_i \in \mathcal{R})$$

where R is the set of real numbers. Thus, E_1 represents just a line, where E_2 represents an Euclidian plane, and E_3 represents a 3-dimensional Euclidian space.

The *norm*, also called "magnitude", $\|y\|$ of $y = (y_1, y_2, ..., y_n)$ in E_n is defined to be:

$$\|y\| = \sqrt{y_1^1 + y_2^2 + ... + y_n^2}$$

which can be viewed as its distance from the origin. Therefore the distance between two points , y and z, will be:

$$z = \|z - y\| = \sqrt{(z_1 - y_1)^2 + (z_2 - y_2)^2 + ... + (z_n - y_n)^2}$$

A subset U of E_n is called *open* if, for every y in U, all points of E_n within some

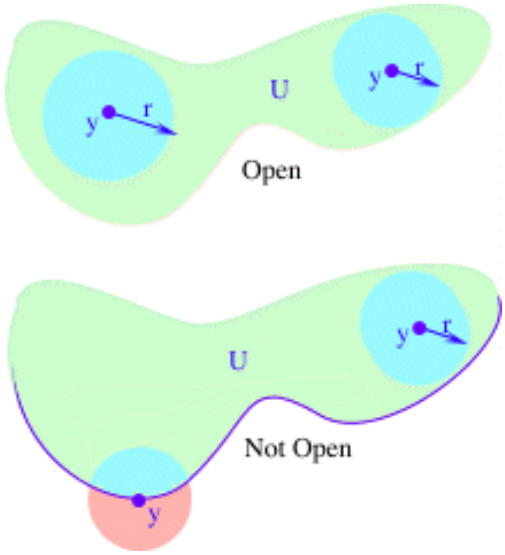

Fig. 5. Representation of open and not open sets

positive distance r of y are also in U. In figure 5, the top set U is a set that does not include any boundary points and is therefore *open*. Thus, given any point y in U, we can find a little ball centered at y that lies entirely within U.

The lower set is not open since it includes on its lower part some boundary points. For example, if one chooses y on the boundary, then it is impossible to find a little ball centered at y entirely contained within U.

Intuitively, one can visualize an open set as a solid region minus its boundary. If the boundary is included then we will get a *closed set*. The closed set is formally defined as the complement of the open set.

Let $M \subset E_s$. A subset $U \subset C$ is called open in M if, for every y in U, all points within some positive distance r of y are also in U. For the parametric paths and surfaces in E_3, from now on, the three coordinates of 3-space will be referred to as $y_1,\ y_2,\ and\ y_3$ [38].

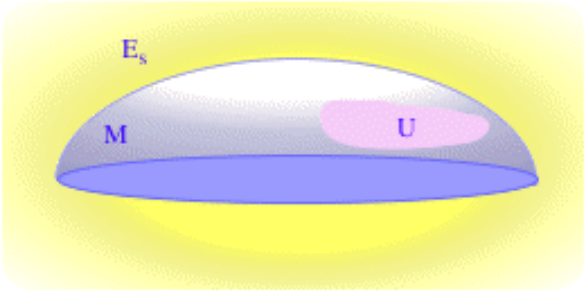

Fig. 6. Open set in M

In figure 6, M is the hemisphere, E_s is 3-dimensional space and U is a small "patch" on M that excludes its boundary. Thus U is not open in E_s, since there are points in E_s arbitrarily close to U that lie outside U. However, it is open in M, since given any point y in U, all points of M within a small enough distance from y are still in U.

This will help define a smooth path in E_3 as a set of three smooth (infinitely differentiable) real-valued functions of a single real variable t:

$$y_1 = y_1(t),\ y_2 = y_2(t),\ \textit{and}\ y_3 = y_3(t)$$

The *parameter* of the curve is the term that usually describes the variable t. The path is *non-singular* if the vector $(\frac{dy_1}{dt},\ \frac{dy_2}{dt},\ \ \frac{dy_3}{dt})$ is nowhere zero.

A smooth path in E_n is defined as a collection of smooth functions $y_i = y_i(t)$, where i goes from 1 to n. A smooth surface immersed in E_3 is a collection of three smooth real-valued function of two variables x^1 and x^2 (see figure 7).

$$\begin{aligned} y_1 &= y_1(x^1, x^2) \\ y_2 &= y_2(x^1, x^2) \\ y_3 &= y_3(x^1, x^2) \end{aligned}$$

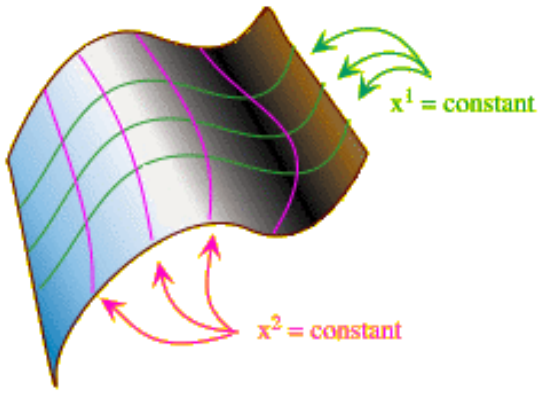

Fig. 7. Smooth surface immersed in E_3

or just

$$y_i = y_i(x^1, x^2), \ \textit{where } i = (1, 2, 3)$$

Then x^1 and x^2 are called the *parameters* or *local coordinates*. A unit sphere ($y_1^2 + y_2^2 + y_3^2 = 1$), when using spherical coordinates, is a good example that illustrates this concept (see figure 8).

$$\begin{aligned} y1 &= \sin(x^1)\cos(x^2) \\ y2 &= \sin(x^1)\sin(x^2) \\ y3 &= \cos(x^1) \end{aligned}$$

where x^1 *and* x^2 are the polar coordinates (the angles are shown on the figure).

Therefore, the parametric equations of a surface show us how to obtain a point on the surface once we know the two local coordinates (or parameters). In other words, this operation allows us to specify a function $E_2 \rightarrow E_3$. Thus, in order to obtain the local coordinates from the Cartesian coordinates y_1, y_2 *and* y_3, one will need to solve for the local coordinates x^i as a function of y_j. For example, for the

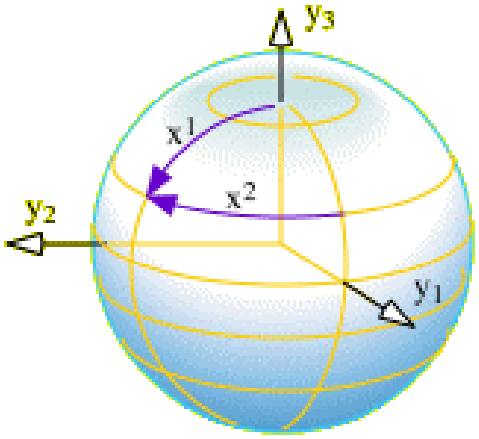

Fig. 8. The unit sphere

sphere case, we will get:

$$x^1 = \cos^{-1}(y_3)$$

$$x^2 = \begin{cases} \cos^{-1}(y_1/\sqrt{y_1^2+y_2^2}) & \text{if } y_2 \geq 0 \\ 2\pi - \cos^{-1}(y_1/\sqrt{y_1^2+y_2^2}) & \text{if } y_2 < 0 \end{cases}$$

This technique presents the advantage of allowing us to give each point on much of the sphere *two unique coordinates*, x^1 and x^2. Even though there is a continuity problem as y_2 approaches 0, since then x^2 "jumps" from 0 to 2π, and at the poles ($y_1 = y_2 = 0$) since the function is not even defined, we still can restrict the portion of the sphere to an open subset of the sphere by having:

$$0 < x^1 < \pi \text{ and } 0 < x^2 < 2\pi$$

x^1 and x^2 are called the *coordinates functions*.

A chart of a surface S is a pair of functions $x = (x^1(y_1, y_2, y_3), x^2(y_1, y_2, y_3))$ which specify each of the local coordinates (parameters) x^1 and x^2 as smooth functions of a general point (global or ambient coordinates) on the surface (see figure 9).

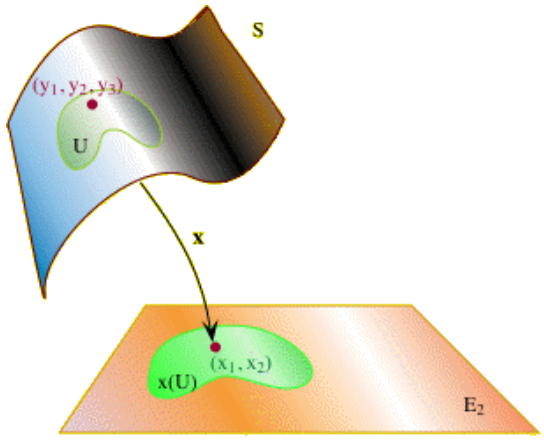

Fig. 9. Chart

2. Smooth Manifolds and Scalar Fields

An *open cover* of $M \subset E_s$ is a collection $\{U_\alpha\}$ of open sets in M such that $M = \cup_\alpha U_\alpha$. The following are two examples that can be used to illustrate these concepts: E_s can be covered by open balls and the unit sphere in E_s can be covered by the collection $\{U_1, U_2\}$ (see figure 10) where:

$$U_1 = \{(y_1, y_2, y_3) | y_3 > -\frac{1}{2}\}$$

$$U_2 = \{(y_1, y_2, y_3) | y_3 < \frac{1}{2}\}$$

A subset M of E_s is called an *n-dimensional smooth manifold* if there is a collection $\{U_\alpha; x^1_\alpha, x^2_\alpha, ..., x^n_\alpha\}$ where:

- The U_α form an open cover of **M**
- Each x^r_α is a C^∞ real-valued function defined on U (that is, $x^r_\alpha : U_\alpha \rightarrow E_n$) given by x(u)=$(x^1_\alpha(u), x^2_\alpha(u), ..., x^n_\alpha(u))$ is one-to-one (that is , to each point U_α is assigned a *unique* set of n coordinates). The tuple $(U_\alpha; x^1_\alpha, x^2_\alpha, ..., x^n_\alpha)$ is called a *local chart of M*. The collection of all charts is called a *smooth atlas of M*.

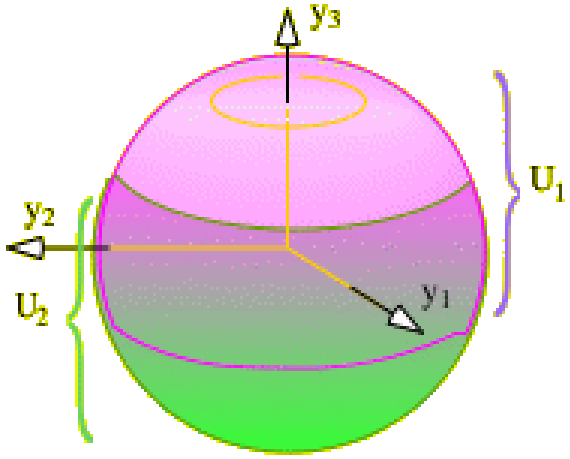

Fig. 10. Unit sphere covered by the collection $\{U_1, U_2\}$

Further, U_α is called a *coordinate neighborhood*.

- If (U, x^j) and $(V, \overline{x}^j)$ are two local charts of M, and if $U \cap V \neq 0$, then one can write $x^i = x^i\overline{x}^j$ and its inverse $\overline{x}^k = \overline{x}^k(x^l)$ for each i and k, where all functions are C^∞ (all the functions are smooth). This ensemble of functions is called the *change-of-coordinates* transformation (see figure 11). One should note that x^i should always be thought as the *local coordinates* (or parameters) of the manifold. It is easy to parameterize each of the open sets U by using the inverse function of x, which assigns to each point in some neighborhood of E_n a corresponding point onto the manifold.

Also it is important to notice that the third condition implies that:

$$det\left(\frac{\delta\overline{x}^i}{\delta x^j}\right) \neq 0$$

$$det\left(\frac{\delta x^i}{\delta\overline{x}^j}\right) \neq 0$$

since the matrices associated to those determinants must be invertible.

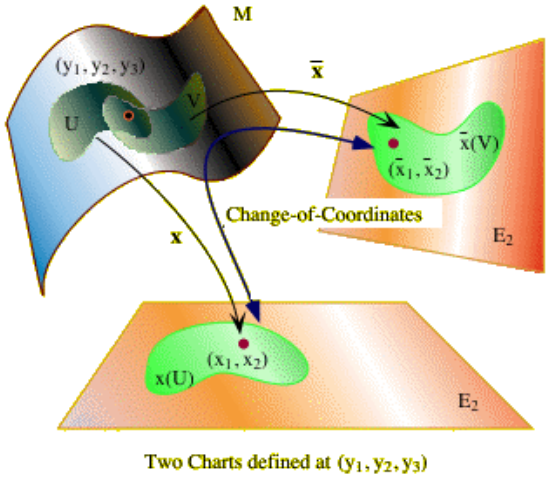

Fig. 11. Change of coordinates using two charts

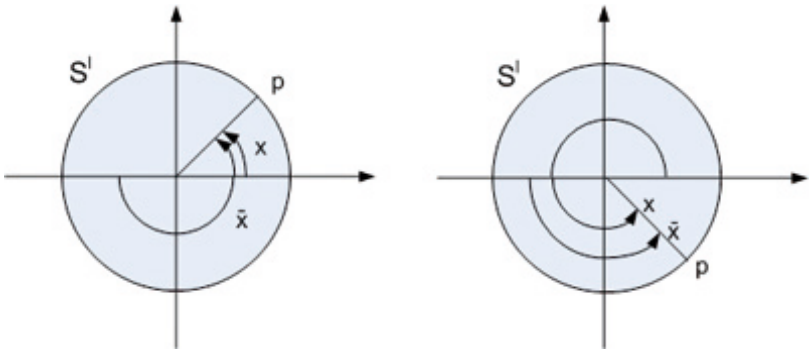

Fig. 12. 1-Dimensional manifold

The following example is a good illustration of the previous definitions. Let's have S^l the unit circle, with the exponential map. Hence, this unit circle is a 1-dimensional manifold. Here is a possible structure (with two charts as shown in figure 12),

One has:

$$x : S^1 - \{(1,0)\} \to E_1$$

$$x : S^1 - \{(-1,0)\} \to E_1,$$

with 0¡x, $\overline{x} < 2\pi$, and the change of coordinate maps are given by:

$$\overline{x} = \begin{cases} x + \pi & \text{if } x < \pi \\ x - \pi & \text{if } x > \pi \end{cases}$$

and

$$x = \begin{cases} \overline{x} + \pi & \text{if } \overline{x} < \pi \\ \overline{x} - \pi & \text{if } \overline{x} > \pi \end{cases}$$

Notice the symmetry between x and $\overline{x}$. Also, notice that these change of coordinate functions are only defined when $\theta \neq 0, \pi$.

3. Tangent Vectors and the Tangent Space

In order for us to fully understand the concept of *vector tangent to smooth manifold*, we first need to introduce the concept of *smooth paths* on M.

A smooth path in the smooth manifold M is a smooth map $r : (-a, a) \rightarrow M$, where $r(t) = (y_1(t), y_2(t), ..., y_s(t))$. We say that r is a smooth path through $m \in M$ if $r(t_0) = m$ for some t_0. We can specify a path in M by its coordinates: $y_1 = y_1(t), y_2 = y_2(t), ..., y_s = y_s(t)$.

Since the ambient and local coordinates are functions of each other, we can also express a path in terms of its local coordinates: $x^1 = x^1(t), x^2 = x^2(t), ..., x^n = x^n(t)$.

A *tangent vector* is an operator that maps functions on the manifold.

$$t : \{f : f \in \mathfrak{F}\} \tag{2.15}$$

A good illustration of this is the following example (see figure 13); Let M be the surface $y_3 = y_1^1 + y_2^2$, which can be parameterize by:

$$y_1 = x^1$$

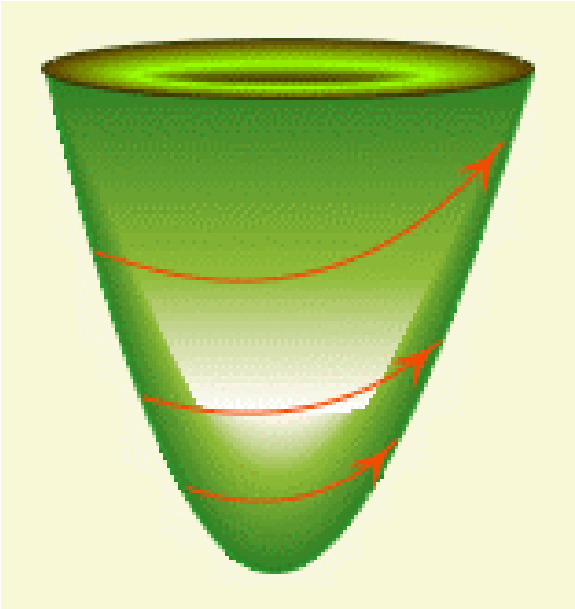

Fig. 13. Tangent vector to a cone. Path on M

$$y_2 = x^2$$

$$y_3 = (x^1)^2 + (x^2)^2$$

This corresponds to the single chart $(U = M; x^1, x^2)$, where

$$x^1 = y_1 \; and \; x^2 = y_2$$

To specify the tangent vector, let's first specify a path in M, such as:

$$y_1 = \sqrt{t}\sin(t)$$

$$y_2 = \sqrt{t}\cos(t)$$

$$y_3 = t$$

giving us the path shown in figure 13 and figure 14.

We then can obtain a tangent vector field along the path by taking the appro-

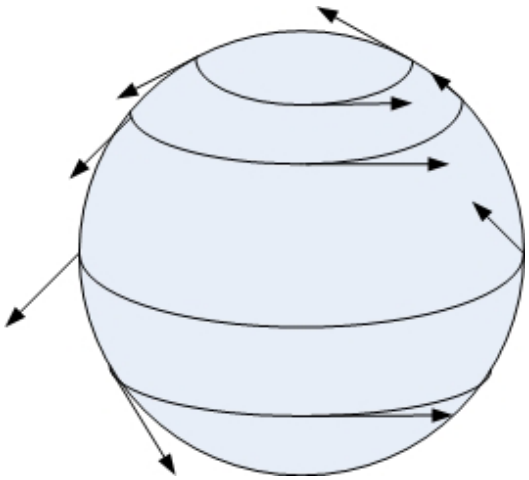

Fig. 14. Tangent vector to a sphere. Path on M

priate derivatives:

$$(\frac{dy_1}{dt}, \frac{dy_2}{dt}, \frac{dy_3}{dt}) = (\sqrt{t}\cos t + \frac{\sin t}{2\sqrt{t}}, -\sqrt{t}\sin t + \frac{\cos t}{2\sqrt{t}}, 1)$$

In order to get the actual tangent vectors at points in M, one needs to evaluate this at a fixed point t_0. Also, it is easy to realize that we can express the coordinates x^i in terms of t.

If M is an n-dimensional manifold, and $m \in M$, then *the tangent space at m* (see figure 15) is the set T_m of all tangent vectors at m. The above constructions turn T_m into a *vector space*.

4. Contravariant and Covariant Vector Fields

A *contravariant vector* at $m \in M$ is a collection v^i of n quantities (defined for each chart at m) which transform according to the formula:

$$\overline{v}^i = \frac{\delta \overline{x}^i}{\delta x^j} v^j$$

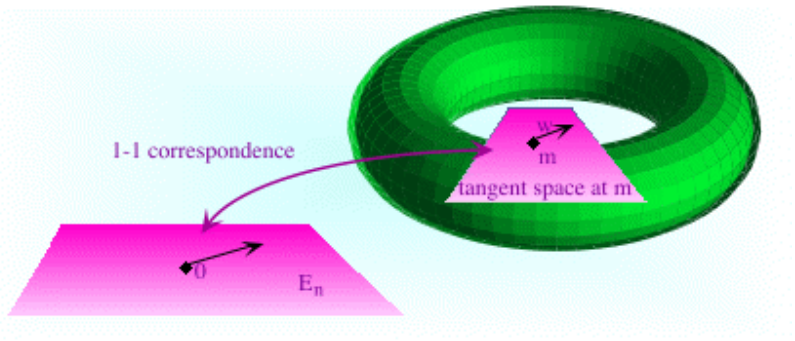

Fig. 15. Tangent space at m

It goes along with the fact that contravariant vectors are just tangent vectors.

At each point m in a manifold M, we have the n vectors $\frac{\delta}{\delta x^1}, \frac{\delta}{\delta x^2}, ..., \frac{\delta}{\delta x^n}$, where the typical vector $\frac{\delta}{\delta x^i}$ was obtained by taking the derivative of the path. Hence,

$$\frac{\delta}{\delta x^i} = vector\ obtained\ by\ differentiating\ the\ path\ x^j = \begin{cases} t + constant & \text{if j=i} \\ constant & \text{if } j \neq i \end{cases}$$

where the constants are chosen to make $x^i(t0)$ correspond to m for some t_0.

Note that a tangent filed is a field on (part of) a manifold, and as such, it is not, in general, constant. The only things that are constant are its coordinates under the specific chart x. The corresponding coordinates under another chart $\overline{x}$ are $\delta\overline{x}^j/\delta x^i$, are not constant in general. The vector field looks like figure 16. If one wants to *path together* local vector fields, making them not local anymore, one will need to extend them to the whole of M. In order to do so, one will need to make them zero near the boundary of the coordinate patch. The following procedure will allow this.

If $m \in M$ and x is any chart of M, lat x(m) = y and let D be a ball of some radius r centered at y entirely contained in the image of x. Now define a vector field

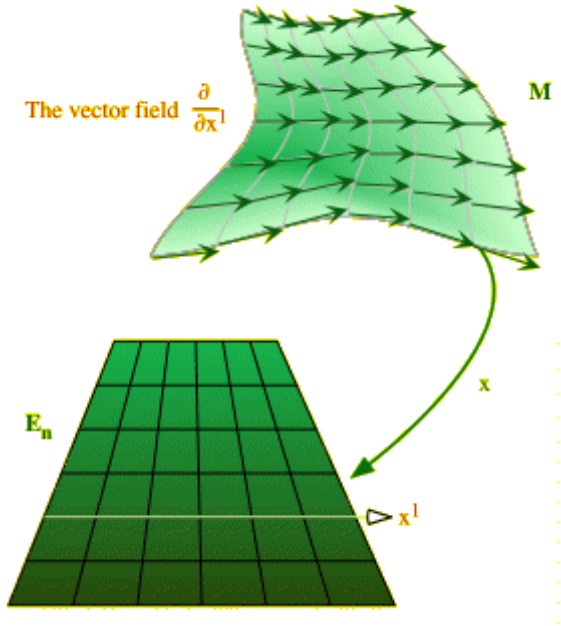

Fig. 16. The vector field $\delta/\delta x^i$

on the whole of M by:

$$w(p) = \begin{cases} \delta/\delta x^j e^{-R^2} & \text{if p is in D} \\ 0 & \text{otherwise} \end{cases}$$

where R=$(|x(p) - y|)/(r - |x(p) - y|)$. The figure 17 shows what this field looks like on M. The fact that $\overline{V}^i$ is a smooth function of the $\overline{x}^i$ now follows from the fact that

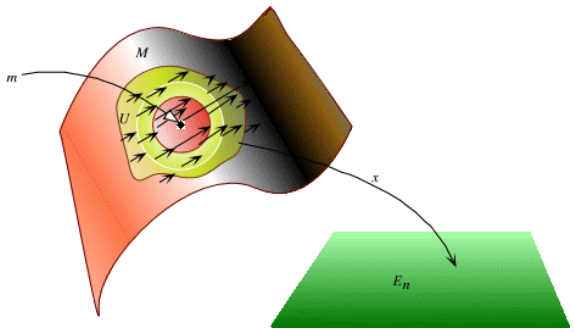

Fig. 17. Field on M

all the partial derivatives of all orders vanish as you leave the domain of x.

A *covariant vector field* C on M associates with each chart x a collection of n smooth functions $C_i(x^1, x^2, ..., x^n)$ which satisfy (covariant vector transformation rule):

$$\overline{C}_i = C_j \frac{\delta x^j}{\delta \overline{x}^i}$$

Geometrically, a contravariant vector is a vector that is tangent to the manifold. Hence, a *smooth 1-form*, also called a smooth cotangent vector field on the manifold M (or on an open subset U of M) is a function F that assigns to each tangent vector field V on M (or on the subset U) a scalar field F(V) which is smooth (see figure 18). It has the following properties:

$$F(V + W) = F(V) + F(W)$$

$$F(\alpha V) = \alpha F(V)$$

for every pair of tangent vector fields V and W, and every scalar α. The equivalent in linear algebra is that F is a linear transformation from the vector space of smooth tangent vector fields on M to the the vector space of smooth scalar fields on M.

5. Tensor Fields

Suppose that v = (v1, v2, v3) and w = (w1, w2, w3) are vector fields on E_3. Then their *tensor product* is defined to consist of the nine quantities $v_i w_j$. Thus, let V and W be contravariant, and let C and D be covariant. Then:

$$\overline{v}^i \overline{w}^j = \frac{\delta \overline{x}^i}{\delta x^k} V^k \frac{\delta \overline{x}^j}{\delta x^m} W^m = \frac{\delta \overline{x}^i}{\delta x^k} \frac{\delta \overline{x}^j}{\delta x^m} V^k W^m$$

and similarly,

$$\overline{v}^i \overline{c}_j = \frac{\delta \overline{x}^i}{\delta x^k} \frac{\delta \overline{x}^j}{\delta x^m} V^k C_m$$

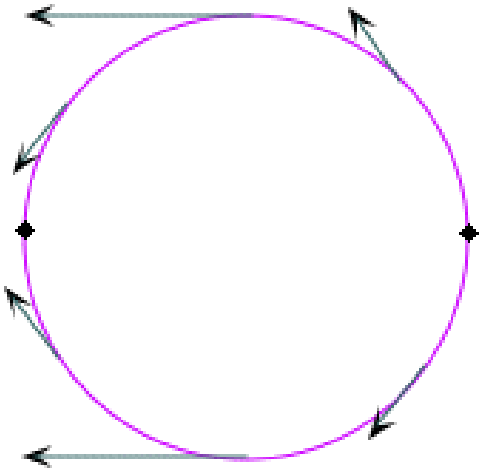

Fig. 18. Smooth cotangent vector field

and,

$$\overline{c}_i \overline{d}_j = \frac{\delta x_k}{\delta \overline{x}^j} \frac{\delta x_m}{\delta \overline{x}^j} C_k D_m$$

These product fields are called respectively "tensors" of type (2,0), (1,1), and (0,2).

A tensor field of type (2,0) on the n-dimensional smooth manifold M associates with each chart x a collection of n^2 smooth functions $T^{ij}(x^1, x^2, ..., x^n)$ which satisfy the transformation rules shown below. Similarly, we define tensor fields of type (0,2), (1,1), and, more generally, a tensor field of type (m,n).

It is important to note that a tensor field of type (1,0) is just a contravariant vector field, while a tensor field of type (0,1) is a covariant vector field. Similarly, a tensor field of type (0,0) is a scalar field. Also type (1,1) tensors correspond to linear transformations in linear algebra.

The *Kronecker Delta Tensor*, given by:

$$\delta^i_j = \begin{cases} 1 & \text{if j=i} \\ 0 & \text{otherwise} \end{cases}$$

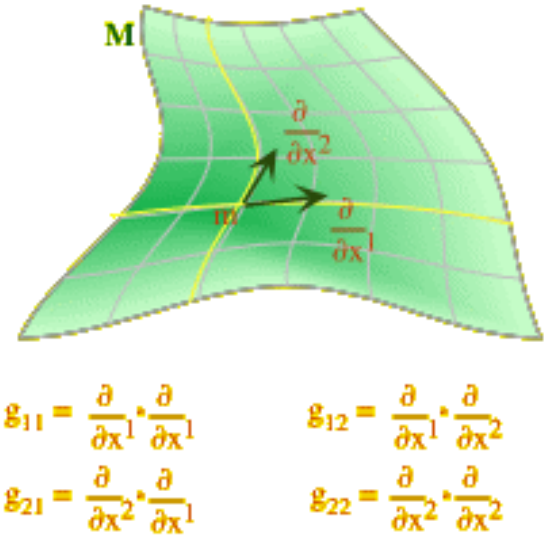

Fig. 19. A metric tensor

is a tensor field of type (1,1). Indeed, one has $\delta^i_j = \frac{\delta x^i}{\delta x^j}$ and the latter quantities transform according to the rule:

$$\overline{\delta}^i_j = \frac{\delta \overline{x}^i}{\delta x^k} \frac{\delta x^m}{\delta \overline{x}^j} \delta^k_m$$

whence they constitute a tensor field of type (1,1).

A *metric tensor* (see figure 19) is defined by a set of quantities g_{ij} where:

$$g_{ij} = \frac{\delta}{\delta x^i} \frac{\delta}{\delta x^j}$$

This is a tensor of type (0,2). This tensor is called "the metric tensor inherited from the imbedding of M in Es."

6. Riemannian Manifolds

A smooth inner product on a manifold M is a function $< -, - >$ that associates to each pair of smooth contravariant vector fields X and Y a scalar (field) $< X, Y >$, satisfying the following properties.

- Symmetry: $< X, Y >=< Y, X >$ for all X and Y.

  Also, $< aX, bY >$ = ab $< X, Y >$ for all X and Y, and scalars a and b

- Bilinearity: $< X, Y + Z >=< X, Y > + < X, Z >$.

- Non-degeneracy: If $< X, Y >= 0$ for every Y, then X = 0

A manifold endowed with a smooth inner product is called a *Riemannian manifold*. It is important to realize that if x is any chart, and p is any point in the domain of x, then :

$$< X, Y >= X^i Y^j < \frac{\delta}{\delta x^i}, \frac{\delta}{\delta x^j} >$$

This gives us smooth functions

$$g_{ij} =< \frac{\delta}{\delta x^i}, \frac{\delta}{\delta x^j} >$$

such that

$$< X, Y >= g_{ij} X^i Y^j$$

and which constitutes the coefficients of a type (0,2) symmetric tensor. This tensor is called the *fundamental tensor* or metric tensor of the Riemannian manifold.

Here are some things we can do with a Riemannian manifold. If X is a contravariant vector field on M, then the square norm norm of X is defined by:

$$\|X\|^2 =< X, X >= g_{ij} X^i X^j$$

Unlike in regular algebra, $\|X\|^2$ can be negative. If $\|X\|^2 < 0$, X is called *timelike*; if $\|X\|^2 > 0$, X is called *spacelike*, and if $\|X\|^2 = 0$ X is called *null* (see figure 20). If X is not spacelike, it can then be defined as:

$$\|X\| = (\|X\|^2)^{1/2} = (g_{ij} X^i X^j)^{1/2}$$

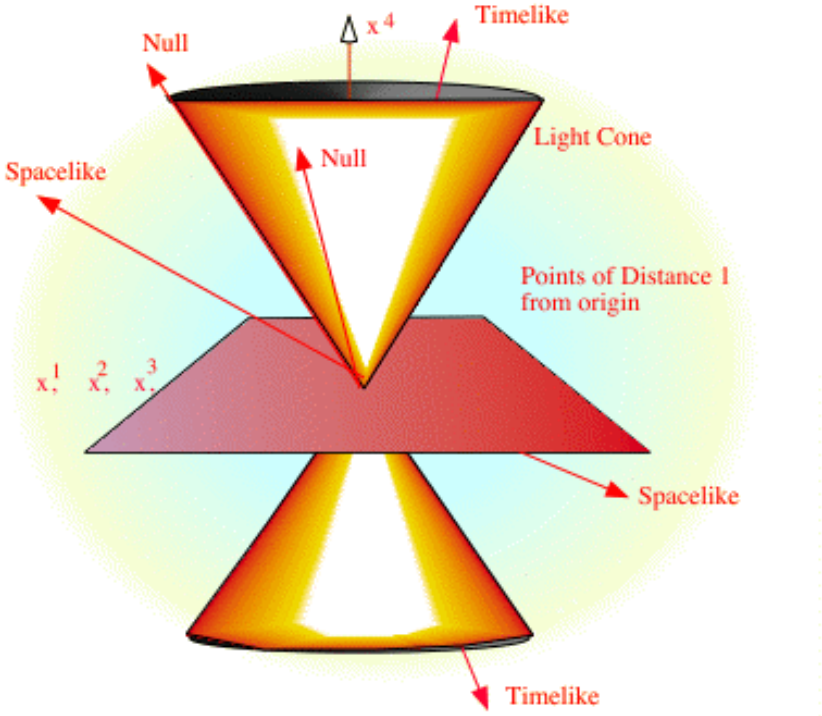

Fig. 20. X as a timelike, spacelike, and null

One of the useful applications for these Riemannian manifolds is *arc length*. If C is a non-null path in M, then its length is defined as follows: First break the path into segments S where each of them lie in some coordinate neighborhood, and then define the length of S by:

$$L(a,b) = \int_a^b (\pm g_{ij} \frac{dx^i}{dt} \frac{dx^j}{dt})^{1/2}$$

where the sign ± 1 is chosen as $+1$ if the curve is space-like and -1 if it is time-like. In other words, we are defining the arc-length differential form by:

$$ds^2 = \pm g_{ij} dx^i dx^j$$

Another useful application is the *parameterizations by arc length.* In order to so, one needs to let C be a non-null path $x^i = x^i(t)$ in M. Then, one needs to fix a point t = a on this path, and define a new function s (arc length) by s(t) = L(a, t) = length of path from t = a to t. Then s is an invertible function of t, and, using s as a parameter, $\|dx^i/ds\|^2$ is constant, and equals 1 if C is space-like and -1 if it is time-like.

CHAPTER III

DISTRIBUTIONAL APPROACH TOWARD APPLICATIONS TO MEASURING BIASED AND UNBIASED DETECTION ROBUSTNESS

A robustness measure technique, based on differential geometric tools is presented in this chapter. The first part of the chapter shows that when one models imperfectly known quantities as elements of a non-Euclidian manifold, the use of tangent space is the most effective method of measuring robustness for a variety of communications and signal processing algorithms. Through the use of different types of parameter surfaces, the second part of the chapter illustrates the previous technique.

A. Introduction

Engineers do not always deal with exact knowledge of both design and analysis. Real world applications of algorithms are often made more difficult by the presence of inexact knowledge of certain quantities. Often, such knowledge pertains to the statistics of an underlying random process such as elements of the covariance matrix. For an algorithm to be successful, it must posses a degree of robustness to this inexact knowledge (i.e., the algorithm's performance should not be too sensitive to inexact statistical knowledge). The recognition of the importance of robustness allows engineers to combine both performance and robustness into the development of an algorithm [15, 16, 17, 18, 19, 20]. Much past work has been performed in engineering robustness research by Huber-Strassen [2, 3] on saddlepoint techniques. Even though this technique still plays an important role in today's research, it contains a major limitation through the absence of a natural quantitative way to evaluate the performance of the technique. This is a major problem because often the saddlepoint solution (defined as the "robust" solution) is difficult to obtain in practical

applications, and thus alternative algorithms were considered.

More recently, a new group of researchers (Halverson, Bakich, Wise...) has started to use differential geometric tools to develop more natural quantitative measures of the degree of robustness. This new direction of research allows engineers to combine both performance and robustness into the development of an algorithm. The thrust of this work is to extend the differential geometric approach toward robustness in different ways.

It is important to note that past work made use of both Euclidian and curved space to model the inexact knowledge. Much of this past work viewed the corresponding perturbations *only* in Euclidian space [10, 6, 7, 12, 13, 11]. For example, if one is considering a strictly increasing nominal distribution function, this distribution function can be allowed to vary locally in a region of m-dimensional Euclidian space, where m is ultimately allowed to approach infinity. There are, however, other situations where the imperfectly known quantities must be constrained to lie on a non-Euclidian manifold.

As an example of practical scenarios where non-Euclidian manifolds may arise, consider applications which involve Gaussian-distributed data. The Gaussian covariance matrix is crucial to an analysis of such an application; but it is unrealistic to assume we will know the matrix perfectly. After all, why would mother nature be so kind to us? The set of possible entries for the matrix is imperfectly known but its constraint, such as being positive definite, might be known. This and other possible constraints will result in the entries lying in a manifold which may very well be non-Euclidian.

B. Development

Consider a performance function $P : R^m \rightarrow R$. For example, the covariance matrix entries may lie in a subset of R^m and P might be the least squares error of a linear estimator employed with data processing an imperfectly known variance matrix. Using the techniques described in [21, 22, 13, 10], one can easily associate the robustness of an algorithm with changes in P as one moves away from the nominal point in R^m.

The most convenient way to do this locally is to look at the rate of change of P as one moves in the manifold M spanned by the imperfectly known quantities, where $\widetilde{C} \subset R^m$, where $\widetilde{C}$ is a set containing all the local coordinates. While potentially misleading, the use of slope is convenient and has been found to be a good indicator of the actual performance, in many situations where the more sensitive additional measure provided by curvature [23, 8, 12] is not necessary.

In the next section, we present the detection problem under consideration and the techniques introduced by [2] to obtain the design of the robust detector.

C. Preliminaries and Problem Statement

As an example of the application of the above approach, consider detection in zero mean noise with two samples. The noise samples are realizations of the random variables X and Y, where X and Y are jointly Gaussian distributed random variables with covariance matrix C:

$$C = \begin{pmatrix} a & c \\ c & b \end{pmatrix}$$

where $a = E(X^2)$, $b = E(Y^2)$ and $c = E(XY)$. In order to ensure a positive definite matrix, we need, $c^2 < ab$. Performance function is the false alarm rate; similar methods could be obtained for detection probability by simply shifting the underlying

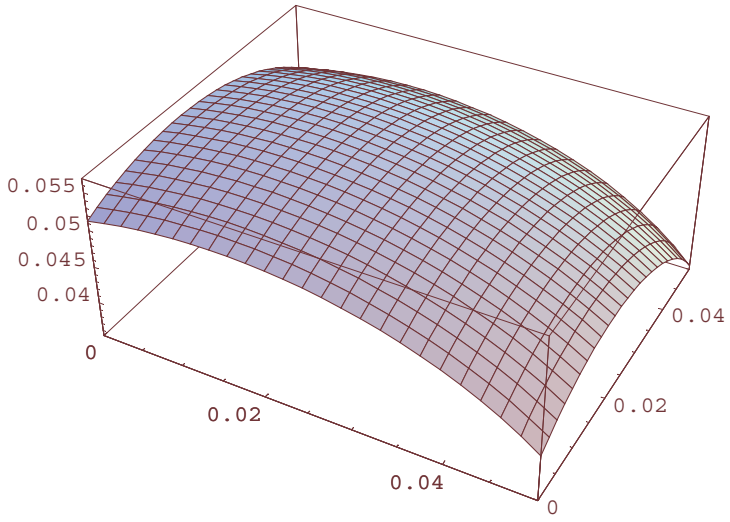

Fig. 21. Upper part of the manifold

distribution.

Supposed (a,b,c) are close to the nominal (a_0, b_0, c_0) according to (for reasons discussed later):

$$(a - a_0)^2 + (b - b_0)^2 + 2(c - c_0)^2 = \epsilon \tag{3.1}$$

where equation 3.1 represents a surface parameter of a 3D manifold centered at (a_0, b_0, c_0). The manifold of vectors of covariance entries is a paraboloidal bowl (an ellipsoidal "egg" shape) which is vertically constrained via $c^2 < ab$ (see figures 21 and 22). Combining the constraint of the previous equation together with the need of a positive definite covariance matrix results in (a,b,c) lying in a two dimensional manifold.

The following section is dedicated to the derivation of closed form expressions for the distribution function of the test statistic under each hypothesis.

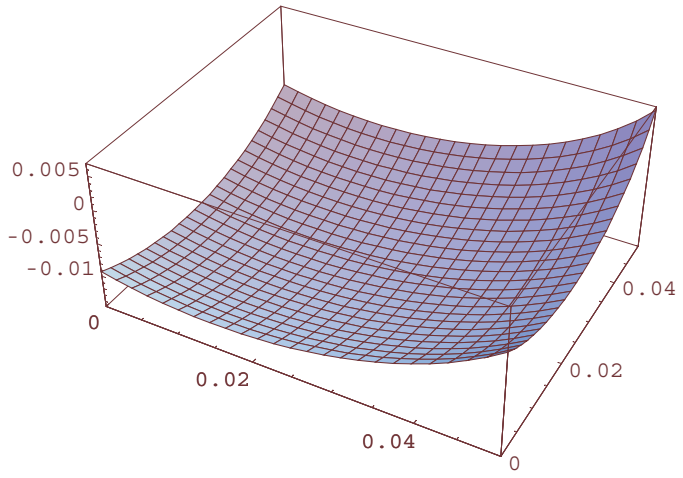

Fig. 22. Lower part of the manifold

D. Applications

The discrete time detection problem under consideration reduces to a hypothesis test of the following form:

$$\mathcal{H}_0 : \text{Noise}$$
$$(\text{or } \mathcal{H}_0 : P_0)$$
$$\mathcal{H}_1 : \text{Noise} + \text{Signal}$$
$$(\text{or } \mathcal{H}_1 : P_1)$$

Noise denotes the distribution function P_0 under the null hypothesis and *Noise + Signal* denotes the distribution P_1 under the signal present hypothesis. Figure 23 illustrates the previous statement for the corresponding densities, assuming they exist.

Based on the realizations $\{y_i\}_{i=1}^n$ of the random variables $\{Y_i\}_{i=1}^n$ which have distribution function $P_j (j = 0, 1)$, the detector attempts to decide between the two hypotheses: $\mathcal{H}_0$ (mean equal to zero) and $\mathcal{H}_1$ (mean equal to two) (see figure 23).

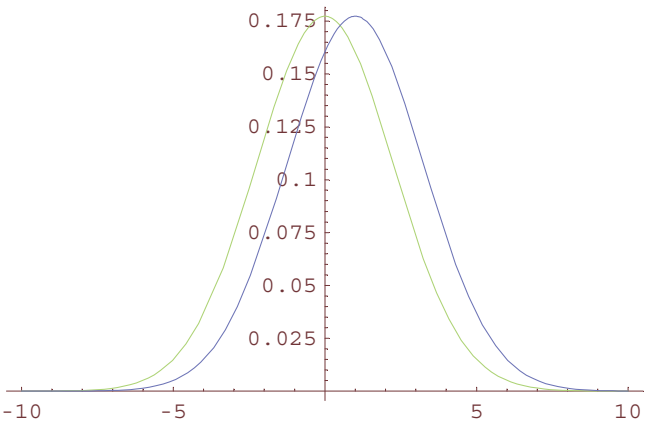

Fig. 23. Hypothesis testing: signal present and signal abscent

As is customary, let α denote the false alarm probability and let β denote the detection probability. If the underlying distribution of the noise were known then one could use the Neyman-Pearson optimal detector [26], therefore:

$$\Lambda(y_1, y_2) = \frac{f_1(y_1, y_2)}{f_0(y_1, y_2)} \tag{3.2}$$

where:

- *Under* $\mathcal{H}_1$, we have ($\mathcal{N}_1 \sim \mathcal{N}(0, \sigma)$):

  ($y_1 \sim \mathcal{N}_1 + s_1$ and $y_2 \sim \mathcal{N}_2 + s_2$) with joint distribution $\mathcal{N}([s_1 s_2]^T, \sigma_1^2, \sigma_2^2, \rho)$.

  It is easy to realize that: $\sigma_1^2 = E[(y_1 - s_1)^2] = E[(\mathcal{N}_1 + s_1 - s_1)^2] = E[\mathcal{N}_1^2] = a$, $\sigma_2^2 = E[(y_2 - s_2)^2] = E[(\mathcal{N}_2 + s_2 - s_2)^2] = E[\mathcal{N}_2^2] = b$, and $\rho = (E[y_1 y_2] - E[y_1]E[y_2])/(\sigma_1 \sigma_2) = (E[(\mathcal{N}_1 + s_1)(\mathcal{N}_2 + s_2) - s_1 s_2])/(\sigma_1 \sigma_2) = c/(\sqrt{ab})$.

  Therefore, we obtain:

$$\begin{aligned} f_1(y_1, y_2) &= \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big[-\frac{1}{2(1-\rho^2)} \\ &* \Big[\frac{(y_1 - s_1)^2}{a} - 2\rho\frac{(y_1 - s_1)(y_2 - s_2)}{\sqrt{ab}} + \frac{(y_2 - s_2)^2}{b}\Big]\Big] \end{aligned} \tag{3.3}$$

- *Under* $\mathcal{H}_0$ this time, we have:

$(y_1 \sim \mathcal{N}_1, y_2 \sim \mathcal{N}_2)$ which is equivalent to:

$\mathcal{N}(0, \sigma_1^2, \sigma_2^2, \rho)$ where $\sigma_1^2 = \sigma_{\mathcal{N}_1}^2 = a$, $\sigma_2^2 = \sigma_{\mathcal{N}_2}^2 = b$, and $\rho = E[y_1 y_2]/(\sigma_1\sigma_2) = E[\mathcal{N}_1\mathcal{N}_2]/(\sigma_1\sigma_2) = c/\sqrt{ab}$.

Therefore:

$$f_0(y_1, y_2) = \frac{1}{2\pi\sqrt{1-\rho^2}} \exp\Big[-\frac{1}{2(1-\rho^2)}[\frac{y_1^2}{a} - 2\rho\frac{(y_1)(y_2)}{\sqrt{ab}} + \frac{y_2^2}{b}]\Big] \tag{3.4}$$

Also, by doing the ratio, one can obtain:

$$\begin{aligned}
\Lambda(y_1, y_2) &= \frac{\exp\Big[-\frac{1}{2(1-\rho^2)}[\frac{(y_1-s_1)^2}{a} - 2\rho\frac{(y_1-s_1)(y_2-s_2)}{\sqrt{ab}} + \frac{(y_2-s_2)^2}{b}]\Big]}{\exp\Big[-\frac{1}{2(1-\rho^2)}[\frac{y_1^2}{a} - 2\rho\frac{(y_1)(y_2)}{\sqrt{ab}} + \frac{y_2^2}{b}]\Big]} \\
&= \exp[\frac{y_1^2}{a} - \frac{2y_1s_1}{a} + \frac{s_1^2}{a} - \frac{y_1^2}{a} - 2\frac{\rho}{\sqrt{ab}}[y_1y_2 - y_2s_1 - y_1s_2 + s_1s_2 - y_1y_2] \\
&+ \frac{y_2^2}{b} - 2\frac{y_2s_2}{b} + \frac{s_2^2}{b} - \frac{y_2^2}{b}]
\end{aligned} \tag{3.5}$$

After few obvious simplifications, one can easily obtain the following equation:

$$\exp[-2\frac{s_1y_1}{a} + \frac{s_1^2}{a} + 2\frac{\rho}{\sqrt{ab}}[s_1y_2 + s_2y_1 - s_1s_2] - 2\frac{s_2y_2}{b} - \frac{s_2^2}{b}] \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{3.6}$$

and now taking the natural log of the previous expression and if a,b,s_1 and $s_2 \geq 0$, one can obtain the simplify expression for the threshold:

$$(-\frac{s_1}{a} + \frac{\rho s_2}{\sqrt{ab}})y_1 + (-\frac{s_2}{b} + \frac{\rho s_1}{\sqrt{ab}})y_2 \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} \widetilde{T} \tag{3.7}$$

Now if one replaces ρ by its expression (see previous page) and after scaling the equation, one can obtain:

$$(cs_2 - bs_1)y_1 + (cs_1 - as_2)y_2 \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} \widehat{T}^* \tag{3.8}$$

The next goal is to find a solution to $\alpha = P(\Lambda < T|Noise)$ where Λ=AY_1+BY_2 and $Y_1 \sim \mathcal{N}(0, a)$, $Y_2 \sim \mathcal{N}(0, b)$. Thus, one will need to compute the variance for both A

and B.

$$\begin{aligned}\sigma_\Lambda^2 &= E[\Lambda^2] - E[\Lambda]^2 \\ &= E[A^2Y_1^2] + 2E[ABY_1Y_2] + E[B^2Y_2^2] \\ &= (cs_2 - bs_1)^2 a + 2c(cs_2 - bs_1)(cs_1 - as_2) \\ &+ (cs_1 - as_2)^2 b \end{aligned} \tag{3.9}$$

where $c = c_0 \pm \sqrt{(\epsilon^2 - (a-a_0)^2 - (b-b_0)^2)/(2)}$ (see figures 21 and 22).

For the computation of the value of the threshold $\widehat{T}$, we will employ nominal values, chosen, for example, according to the following table I. The computations are as followed:

$$\begin{aligned}\alpha &= Pr(\Lambda > \widehat{T}|Noise) \\ &= \int_{\widehat{T}}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_\Lambda^2} e^{-\frac{y^2}{2\sigma_\Lambda^2}} dy \\ &= \frac{1}{2} - erf(\frac{\widehat{T}}{\sigma_\Lambda}) \end{aligned} \tag{3.10}$$

Table I. Nominal values for the 1st example

Point	Value
a_0	1.5
b_0	2
c_0	0.5

In order to get a value for the threshold $\widehat{T}$, one needs to compute the nominal variance ($\sigma^2_{\Lambda_0} = 11.344$). We then obtain, for α=0.05:

$$\begin{aligned} 0.05 &= \frac{1}{2} - erf(\frac{\widehat{T}}{3.3680}) \\ 0.45 &= erf(.2969045\widehat{T}) \\ or\ 0.2969045\widehat{T} &= 1.64399 \\ \widehat{T} &= 5.5371 \end{aligned} \tag{3.11}$$

E. Derivation of Formulas Used for Robustness Analysis

In this section, we first define and then illustrate the derivation of Varma and Halverson [24, 9] for expressions for the slopes that we are going to use for the robustness analysis, namely slope with biased perturbations and slope with unbiased perturbations. We remark that the way a perturbation about the nominal in a manifold is modeled is not unique, and [24, 9] describes two important ways (biased and unbiased) that thus may occur.

1. General Worst Case Slope to Riemannian Manifold

Consider the manifold M and consider a point P on M, with local coordinates $(u_i)_i$. Consider the Riemannian metric g on the tangent space T_pM. If $\overrightarrow{X}$ and $\overrightarrow{Y}$ are tangent vectors at P, then:

$$\overrightarrow{X} = \sum_i X_i \frac{\partial}{\partial u_i} \tag{3.12}$$

$$\overrightarrow{Y} = \sum_i Y_i \frac{\partial}{\partial u_i} \tag{3.13}$$

therefore:

$$\langle \overrightarrow{X}, \overrightarrow{Y} \rangle = \sum_i g_{ij} X_i Y_j \tag{3.14}$$

where:

$$g_{ij} = g\Big(\frac{\partial}{\partial u_i}, \frac{\partial}{\partial u_j}\Big) \tag{3.15}$$

Thus, g is a covariant tensor of order 2. For tangent vector, $\overrightarrow{X} = \sum_i X_i \frac{\partial}{\partial u_i}$, by varying $\overrightarrow{X}$ we wish to maximize directional derivative of function h at P,

$$D_{\overrightarrow{X}} h(P) = \sum_i X_i \frac{\partial h}{\partial u_i}\Big|_P \tag{3.16}$$

subject to the condition that $\overrightarrow{X}$ have a unit length.

$$\langle \overrightarrow{X}, \overrightarrow{Y} \rangle = 1 \tag{3.17}$$

and

$$\sum_i g_{ij} X_i Y_j = 1 \tag{3.18}$$

Using the Lagrange multiplier method, we get:

$$J = \sum_i X_i \frac{\partial h}{\partial u_i}\Big|_P - \lambda \sum_{i,k} g_{jk} X_j X_k \tag{3.19}$$

thus:

$$\forall i,\ \frac{\partial J}{\partial X_i} = \frac{\partial}{\partial X_i}\Big(\sum_i X_i \frac{\partial h}{\partial u_i}\Big|_P - \lambda(\sum_{j,k} g_{jk} X_j X_k)\Big)$$

then

$$\frac{\partial}{\partial X_i}\Big(\sum_i X_i \frac{\partial h}{\partial u_i}\Big|_P - \lambda(\sum_{j\neq k} g_{jk} X_j X_k) - \lambda(\sum_j g_{jj} X_j X_j)\Big) = 0$$

hence

$$\frac{\partial h}{\partial u_i}\Big|_P - \lambda \sum_{k\neq i} g_{ik} X_k - \lambda \sum_{j\neq i} g_{ij} X_j - 2\lambda g_{ii} X_i = 0 \tag{3.20}$$

Using the fact that $G = (g_{ij})$ is symmetric, we have then:

$$\frac{\partial h}{\partial u_i}\Big|_P - 2\lambda \sum_{j\neq k} g_{ij} X_j - 2\lambda g_{ii} X_i = 0 \tag{3.21}$$

or:

$$\frac{\partial h}{\partial u_i}\Big|_P - 2\lambda \sum_j g_{ij} X_j = 0 \tag{3.22}$$

$$D_{\overrightarrow{X}} h(P) - 2\lambda.1 = 0 \tag{3.23}$$

$$\lambda = \frac{1}{2} D_{\overrightarrow{X}} h(P) \tag{3.24}$$

$$\frac{\partial h}{\partial u_i}\Big|_P = D_{\overrightarrow{X}} h(P). \sum_j g_{ij} X_j \tag{3.25}$$

$$\begin{pmatrix} \partial h/\partial u_1 \\ \vdots \\ \partial h/\partial u_m \end{pmatrix} = D_{\overrightarrow{X}} h G \overrightarrow{X}$$

where G is a $m * m$ matrix of the following form:

$$G = \begin{pmatrix} g_{11} & g_{12} & \cdots \\ g_{21} & g_{22} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix}$$

and where $\overrightarrow{X}$ is a column vector of the following form:

$$X_1 = \begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix}$$

Therefore:

$$\begin{pmatrix} \partial h/\partial u_1 \\ \vdots \\ \partial h/\partial u_m \end{pmatrix} = D_{\overrightarrow{X}} h.G \begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix}$$

Since $D_{\overrightarrow{X}}h$ is a scalar, we will obtain:

$$D_{\overrightarrow{X}}h\Big(\frac{\partial h}{\partial u_1}\dots\frac{\partial h}{\partial u_m}\Big)\begin{pmatrix} X_1 \\ \vdots \\ X_m \end{pmatrix} = \Big(\frac{\partial h}{\partial u_1}\dots\frac{\partial h}{\partial u_m}\Big)G^{-1}\begin{pmatrix} \frac{\partial h}{\partial u_1} \\ \vdots \\ \frac{\partial h}{\partial u_m} \end{pmatrix} \tag{3.26}$$

$$(D_{\overrightarrow{X}}h)^2 = \nabla h G^{-1}\nabla h^T \tag{3.27}$$

Hence:

$$(D_{\overrightarrow{X}}h)\Big|_{Extreme} = \sqrt{\nabla h G^{-1}\nabla h^T} \tag{3.28}$$

While it might not be obvious from the above expression, the value of $(D_{\overrightarrow{X}}h)\Big|_{Extreme}$ does not depend on the choice of the coordinates, provided the underlying manifold remains fixed. However, this follows from the classical interpretation of directional derivative as a limit of ∇h over the arc length, where the answer computes to be independent of the curve chosen as long as the tangent to the curve at P is fixed.

2. Slope with Unbiased Perturbations

If the parameter surface is defined by c=f(a,b), we can use $a = u_1$ and $b = u_2$. Then we obtain:

$$\frac{\partial}{\partial a} = (1, 0, \frac{\partial c}{\partial a}) \tag{3.29}$$

$$\frac{\partial}{\partial b} = (1, 0, \frac{\partial c}{\partial b}) \tag{3.30}$$

Inheriting the inner product from R^3

$$g_{11} = 1 + \Big(\frac{\partial c}{\partial a}\Big)^2 \tag{3.31}$$

$$g_{22} = 1 + \Big(\frac{\partial c}{\partial b}\Big)^2 \tag{3.32}$$

$$g_{12} = g_{21} = \frac{\partial c}{\partial a}.\frac{\partial c}{\partial b} \tag{3.33}$$

We then remark that this choice of coordinates is for convenience, and the same result will be obtained for an alternative choice. Thus, the result is without bias. If the performance function is $P = h(a, b, c) \equiv h(a, b)$, we then obtain:

$$\left(D_{\overrightarrow{X}}h\right)\Big|_{Extreme} = \sqrt{\nabla h G^{-1} \nabla h^T} \tag{3.34}$$

$$\nabla h = \left(\frac{\partial h}{\partial a} \quad \frac{\partial h}{\partial b}\right) \tag{3.35}$$

and finally:

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{12} & g_{22} \end{pmatrix}$$

3. Slope with Biased Perturbations

In the case where the slope is biased with perturbations, the variations are translated to the Euclidean space. It is with these variations that the bias enters the system, because the manifold has been altered. Thus, $g_{ij} = \delta_{ij}$, where δ_{ij} is the *Kronecker delta*.

This time we have:

$$G = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

G becomes the identity matrix. Using the new matrix G into equation 4.58, we obtain:

$$\left(D_{\overrightarrow{X}}h\right)\Big|_{Extreme} = \sqrt{\nabla h G^{-1} \nabla h^T} = \sqrt{(\frac{\partial h}{\partial a})^2 + (\frac{\partial h}{\partial b})^2} \tag{3.36}$$

The equation (3.36) is the well known version of the directional derivative applying on functions on affine space.

F. Calculations

All the results are obtained in this section with the use of Matlab simulations. After paying closer attention to the computations, one can easily realize that the integrals $(\partial\alpha)/(\partial a)$ and $(\partial\alpha)/(\partial b)$ do not have a closed form.

$$\frac{\partial\alpha}{\partial a} = \frac{1}{\sqrt{2\pi}} \int_{\widehat{T}}^{\infty} [-\frac{1}{2}(\sigma^2)_{\Lambda}^{\frac{-3}{2}} exp(-\frac{y^2}{2\sigma_{\Lambda}^2}) + \frac{y^2}{2(\sigma_{\Lambda}^2)^{\frac{5}{2}}} exp(-\frac{y^2}{2\sigma_{\Lambda}^2})] \frac{\partial\sigma_{\Lambda}^2}{\partial a} dy \tag{3.37}$$

$$\frac{\partial\alpha}{\partial b} = \frac{1}{\sqrt{2\pi}} \int_{\widehat{T}}^{\infty} [-\frac{1}{2}(\sigma^2)_{\Lambda}^{\frac{-3}{2}} exp(-\frac{y^2}{2\sigma_{\Lambda}^2}) + \frac{y^2}{2(\sigma_{\Lambda}^2)^{\frac{5}{2}}} exp(-\frac{y^2}{2\sigma_{\Lambda}^2})] \frac{\partial\sigma_{\Lambda}^2}{\partial b} dy \tag{3.38}$$

Especially the following part:

$$\int_{\widehat{T}}^{\infty} \frac{1}{const} y^2 exp^{-\frac{y^2}{const}} dy \tag{3.39}$$

In order to determine a solution to those equations, numerical analysis is used[1].

1. Detail of Derivations

$$\frac{\partial f}{\partial a} = \frac{\partial c}{\partial a} = (a_0 - a)(2(\epsilon^2 - (a - a_0)^2 - (b - b_0)^2))^{-\frac{1}{2}} \tag{3.40}$$

$$\frac{\partial f}{\partial b} = \frac{\partial c}{\partial b} = (b_0 - b)(2(\epsilon^2 - (a - a_0)^2 - (b - b_0)^2))^{-\frac{1}{2}} \tag{3.41}$$

$$\frac{\partial h}{\partial a} = \frac{\partial \alpha}{\partial a} \tag{3.42}$$

[1]The Matlab 7.0 source codes for the numerical analysis are included in the appendix

and

$$\Delta h = \Big(\frac{\partial h}{\partial a}\frac{\partial h}{\partial b}\Big) \tag{3.43}$$

Now, let's give the detail of the computations of $(\partial\sigma^2_\Lambda)/(\partial a)$ and $(\partial\sigma^2_\Lambda)/(\partial b)$. Breaking the differentiation into three different parts, we have:

- For $(\partial\sigma^2_\Lambda)/(\partial a)$:

$$\begin{aligned}\frac{\partial 1^{st} term}{\partial a} &= \frac{\partial}{\partial a} a\Big(cs_2 - bs_1\Big)^2 \\ &= \Big(cs_2 - bs_1\Big)^2 + 2a(\frac{\partial f}{\partial a}s_2)(cs_2 - bs_1)\end{aligned} \tag{3.44}$$

$$\begin{aligned}\frac{\partial 2^{nd} term}{\partial a} &= 2\frac{\partial f}{\partial a}(cs_2 - bs_1)(cs_1 - as_2) + 2c(\frac{\partial f}{\partial a}s_2)(cs_1 - as_2) \\ &+ 2c(cs_2 - bs_1)(\frac{\partial f}{\partial a}s_1 - s_2)\end{aligned} \tag{3.45}$$

$$\begin{aligned}\frac{\partial 3^{rd} term}{\partial a} &= \frac{\partial}{\partial a} b(cs_1 - as_2)^2 \\ &= 2b(\frac{\partial f}{\partial a}s_1 - s_2)(cs_1 - as_2)\end{aligned} \tag{3.46}$$

- For $(\partial\sigma^2_\Lambda)/(\partial b)$:

$$\begin{aligned}\frac{\partial 1^{st} term}{\partial b} &= \frac{\partial}{\partial b} a\Big(cs_2 - bs_1\Big)^2 \\ &= 2a(\frac{\partial f}{\partial b}s_2 - s_1)(cs_2 - bs_1)\end{aligned} \tag{3.47}$$

$$\begin{aligned}\frac{\partial 2^{nd} term}{\partial b} &= 2\frac{\partial f}{\partial b}(cs_2 - bs_1)(cs_1 - as_2) + 2c(\frac{\partial f}{\partial b}s_2 - s_1)(cs_1 - as_2) \\ &+ 2c(cs_2 - bs_1)(\frac{\partial f}{\partial b}s_1)\end{aligned} \tag{3.48}$$

$$\begin{aligned}\frac{\partial 3^{rd} term}{\partial b} &= \frac{\partial}{\partial b} b(cs_1 - as_2)^2 \\ &= (cs_1 - as_2)^2 + 2b(\frac{\partial f}{\partial b} s_1)(cs_1 - as_2) \end{aligned} \tag{3.49}$$

2. Sample Densities and Weighting Factors

a. Sample Densities

The gradient over a region of the manifold is not constant. It very much varies over the whole region. Therefore instead of computing the average of the values over the region, we seek a method that quickly illustrates the nature of the variations. If we regard gradient as a random variable, then vital information would be carried by its density. The following procedure is used throughout in the analysis:

- We first decide on the nominal covariance matrix for (X,Y). We also need to assume additional information about the parameters (a,b,c) which compose the covariance matrix expressed by the parameter surface.
- The actual parameters are unlikely to vary from the nominal by a large amount. After choosing an area local to a nominal, we vary the parameters around this region, located on the parameter surface. Then, the gradient values for both, the unbiased and the biased, perturbations are calculated for the new system state (point on the performance surface).
- A sample density for these gradient values is obtained.

b. Weighting Factor

Consider first the sphere in 3-D. If n points are equally arranged along circles of constant latitude, they gradually become more close together as one approaches the pole (figure 24). For this reason when computing a sample density for gradient, such

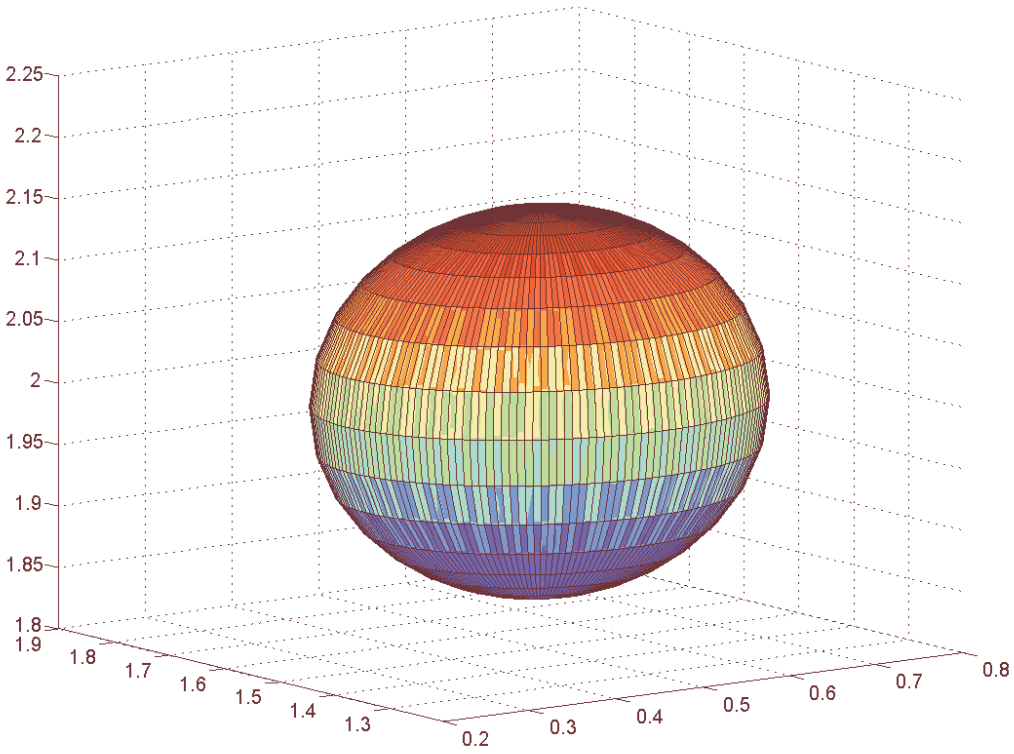

Fig. 24. Point selection over the whole ellipsoid

points are over represented as we approach the pole. When counting them their number should be given reduced "weight". The same issue arises with an ellipsoid. The same "compactification" of points at poles occurs.

Now, let's consider an ellipsoid, where one views the "poles" generated by rotating an ellipse ($Ax^2 + By^2 = C$) about the long axis (see figure 25). The same "compactification" of points at poles occurs, but compared to the sphere, the weighting is more complicated and is affected by three considerations:

- As before, directly proportional to the radius of the circle of constant latitude, r=y
- If one measures the area of a circular strip generated by varying θ by $\pm(\Delta\theta)/2$,

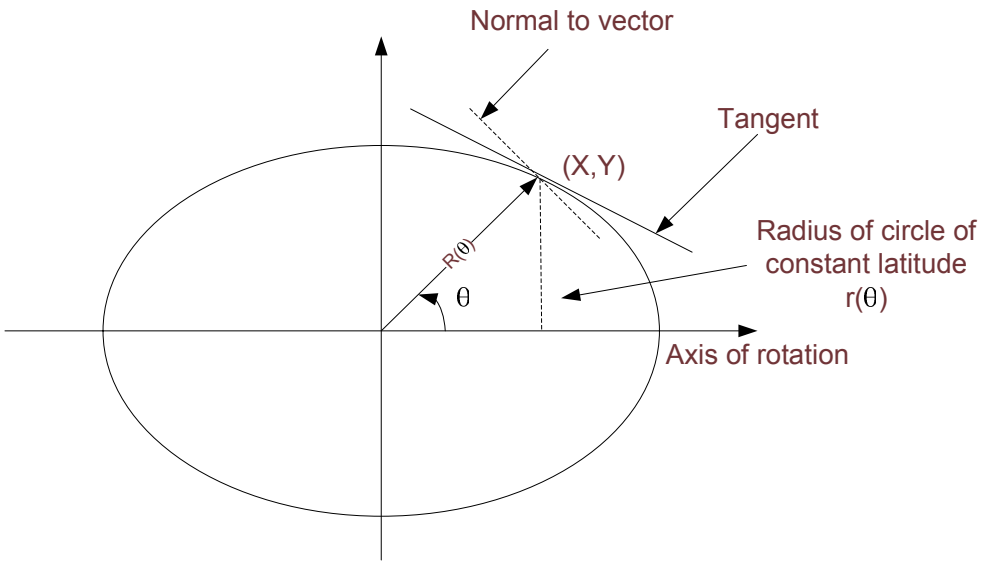

Fig. 25. Rotation along the axis for an ellipsoid

we see that the area of strip depends not only on its radius r, but is directly proportional to (for "width") the non-constant distance from the origin R, and

- The "width" of the strip also differs from the sphere case because it is not constantly normal to θ vector (see figure 25). In the sphere, R is constant and we have normality so the strip "width" depends only on $\Delta\theta$ and r (see figure 26)

As a consequence of the last two points, we see that the weighting factor is also directly proportional to $R(\theta)$ and the ratio $(s_1 + s_2)/(2s)$ (see figure 27). As the deviation from normality increases, and the strip widens, thus increasing the area and partially compensating for smaller r as we move to the pole. It is also easy to realize that the weighting factor could be computed using the volume element. Let's consider the ellipsoid, $a^2 + b^2 + 2c^2 = \epsilon$ or equivalently $c = \pm\sqrt{(\epsilon - a^2 - b^2)/2}$. The

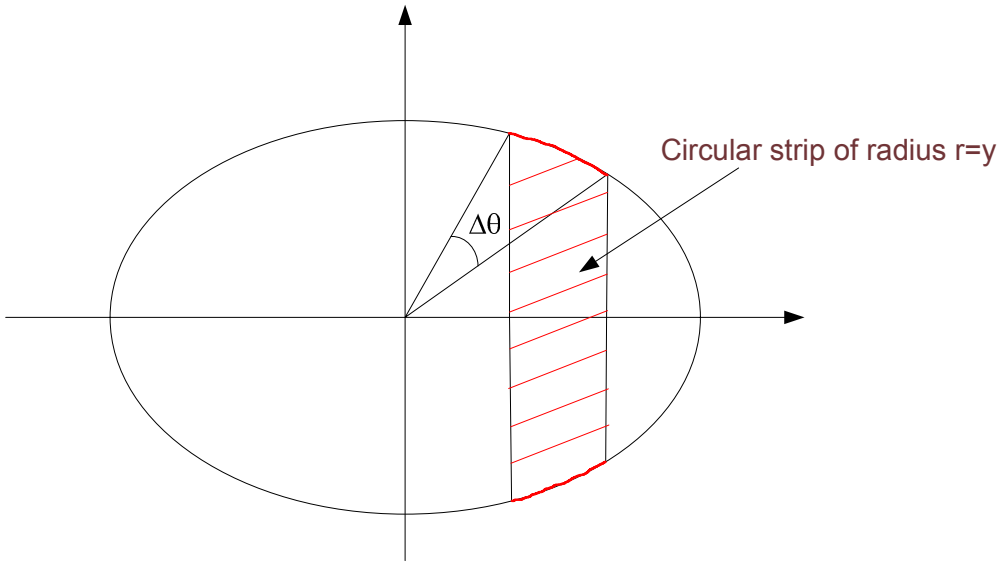

Fig. 26. Circular strip of radius r=y

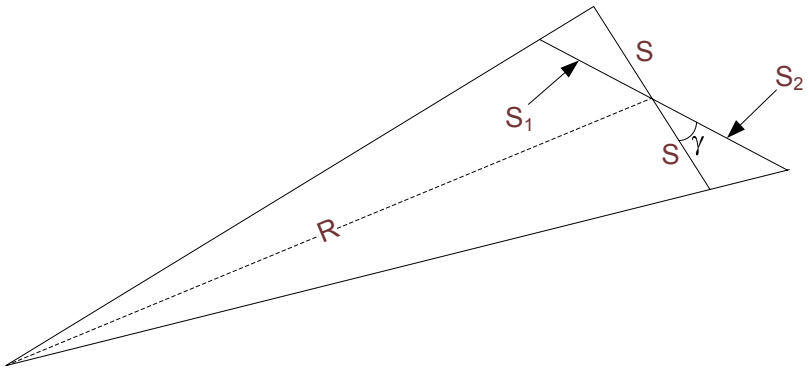

Fig. 27. Weighting factor as a function of the location on the ellipsoid

volume element is then equal to:

$$Volume\ element\ = \sqrt{1 + (\frac{\partial c}{\partial a})^2 + (\frac{\partial c}{\partial b})^2} dadb \tag{3.50}$$

where $\sqrt{1 + (\frac{\partial c}{\partial a})^2 + (\frac{\partial c}{\partial b})^2}$ represents the weight. We then obtain:

$$weight^2 = 1 + (\frac{\partial c}{\partial a})^2 + (\frac{\partial c}{\partial b})^2 \tag{3.51}$$

Now, replacing c by its corresponding expression, and taking the partial of c with respect to a and b, one can obtain:

$$weight = \sqrt{\frac{\epsilon - 1/2a^2 - 1/2b^2}{\epsilon - a^2 - b^2}} \tag{3.52}$$

This weighting factor is expressed in the cartesian coordinates and therefore needs to be expressed, for obvious reasons, in polar coordinates. One will then need to relate the previous expression to the "latitude" θ.

Let's note that $a^2 + b^2 = \epsilon - 2c^2$ and using figure 28, one can obtain the following expression:

$$\begin{aligned} \frac{c}{\sqrt{a^2 + b^2}} &= \tan\theta \\ a^2 + b^2 &= \frac{c^2}{\tan\theta} \\ \frac{c^2}{tan^2\theta} &= a^2 + b^2 = \epsilon - 2c^2 \\ c &= \pm\sqrt{\frac{\epsilon}{2 + \cot^2\theta}} \end{aligned} \tag{3.53}$$

Therefore, when replacing into the equation 3.52, one can obtain:

$$weight = \sqrt{\frac{\epsilon - \frac{1}{2}\frac{\epsilon}{1+2\tan^2\theta}}{\epsilon - \frac{\epsilon}{1+2\tan^2\theta}}} \tag{3.54}$$

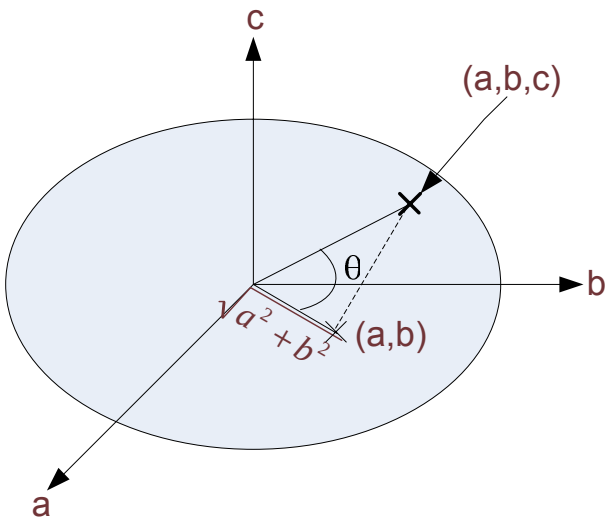

Fig. 28. (a,b,c) related to the latitude θ

which simplifies into:

$$weight = \sqrt{1 + \frac{1}{4}\cot^2\theta} \tag{3.55}$$

where θ is the latitude. However note that the previous weighting factor scales da.db. This assumes rectangular partition in the (a,b) plane, therefore $da.db$ must be converted to $d\theta.d\phi$ where ϕ is the longitude and θ is the latitude. Now, looking at the ellipse, the weighting factor in polar coordinates will then be equal to multiplying (3.55) by the conversion factor of da.db to $d\theta.d\phi$:

$$weighting\ factor = 2\epsilon\sqrt{\sin^2\theta\cos^2\theta + \frac{1}{4}\cos^4\theta} \tag{3.56}$$

The weighting factor is represented in figure 29 for $-90° < \theta < 90°$.

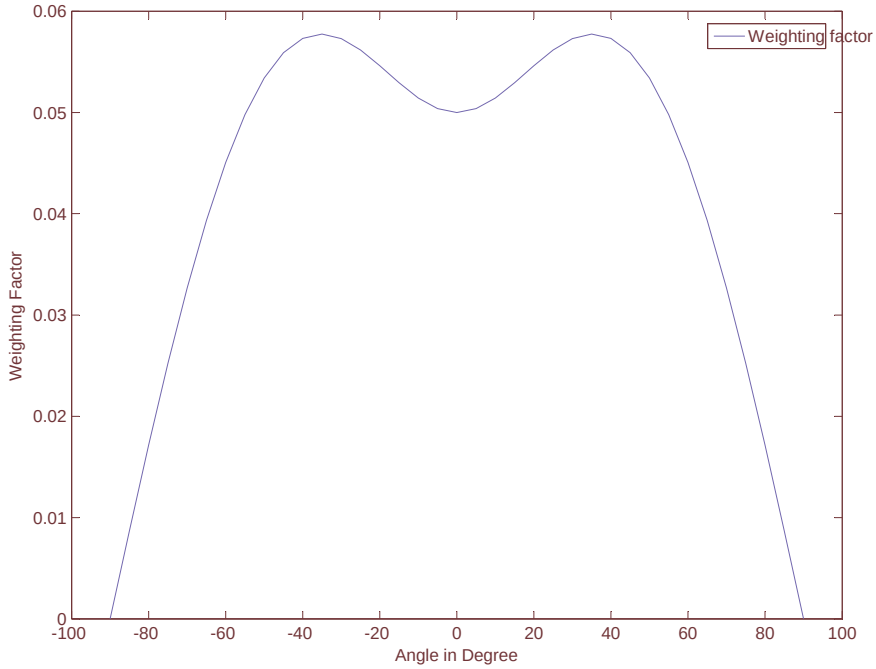

Fig. 29. Weighting factor as a function of θ

c. Sample Size and Bin-Size

The sample densities are all based on bin-sizes of $5 * 10^{-3}$. The sample densities are obtained by adding, for each bin, the weighting factors corresponding to each of the tangent slopes. Figures 30 and 31 illustrate the results. The means and variances are similar (less than 1.5% difference in mean for the unbiased case for example), allowing one to use only 700 sets of points and therefore run faster simulations without affecting the simulation results.

Table II illustrates well the similarity between the two cases scenario.

The sample size includes 700 sets of data points. For each set of three points (c=f(a,b)) the corresponding tangent slope to the surface is computed. The following procedure is used all along in the analysis:

- We first decide on the nominal covariance matrix for (X,Y). We also need to

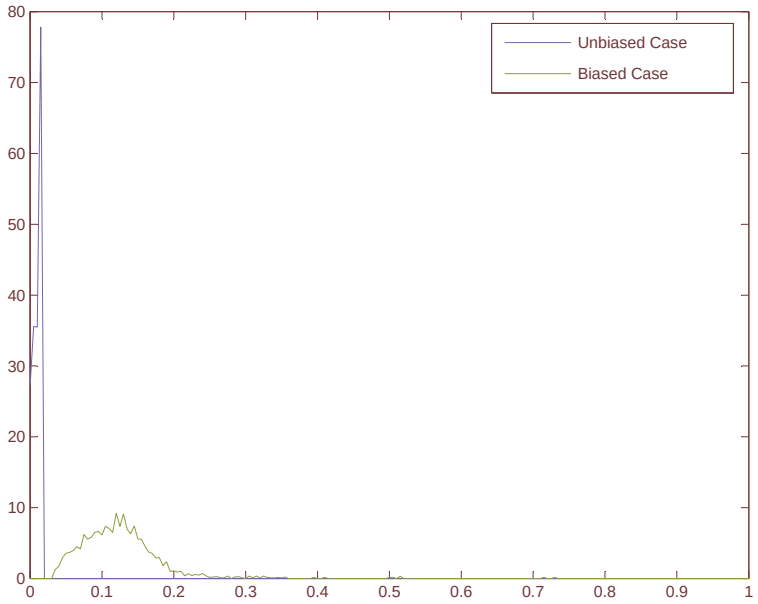

Fig. 30. Sample density with 7000 set of points

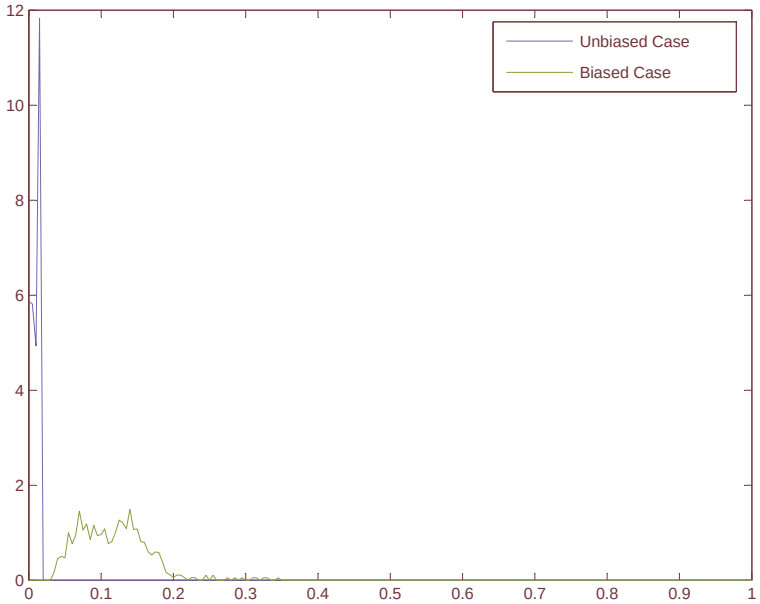

Fig. 31. Sample density with 700 set of points

Table II. Difference in mean and variance for 7000 points and 700 points

	Mean		Variance	
	Unbiased	Biased	Unbiased	Biased
7000 points	0.012153	0.13135	0.000147	0.01725
700 points	0.011492	0.11931	0.001623	0.14503

assume additional information about the parameters (a,b,c) which compose the covariance matrix expressed by the parameter surface.

- The actual parameters are unlikely to vary from the nominal by a large amount. After choosing an area local to a nominal, we vary the parameters around this region, located on the parameter surface. Then, the gradient values for both the unbiased and the biased, perturbations are calculated for the new system state (point on the performance surface).

- Different values of nominal parameters are tested. Also, the effect of the bin-size over the sample densities is implemented. The last case scenario treated is when the variation from the nominal is bigger than usual.

We treated two different examples. The first example corresponds to the use where the covariance matrix lies on a ball centered on the nominal. Such a parameter surface is bounded with positive Gaussian curvatures. The second example uses a new parameter surface; the correlation coefficient is set up to stay constant, thus yielding a saddle surface with negative curvature.

G. Computation of Gradient

For example 1 we employ coordinates (a,b) and express the tangent vectors $(\partial/\partial a, \partial/\partial b)$ in terms of the Euclidean vectors. Inherently the inner product from $\mathcal{R}^3$, it is routine to obtain $g_{11} = 1 + (\partial c/\partial a)^2$, $g_{22} = 1 + (\partial c/\partial b)^2$ and $g_{12} = g_{21} = \partial c/\partial a * \partial c/\partial b$ for the unbiased case. For example 2, the same method is obtained, except coordinates $\widetilde{a}$ and $\widetilde{b}$ are employed. To complete the calculation of 3.28, employing false alarm rate α as the performance function, we easily obtain:

$$\frac{\partial \alpha}{\partial a} = \frac{1}{\sqrt{2\pi}} \int_T^\infty [-\frac{1}{2}(\sigma_\Lambda^2)^{\frac{-3}{2}} exp(\frac{-y^2}{2\sigma_\Lambda^2}) + \frac{y^2}{2(\sigma_\Lambda^2)^{\frac{5}{2}}} exp(\frac{-y^2}{2\sigma_\Lambda^2})] \frac{\partial \sigma_\Lambda^2}{\partial a} dy \tag{3.57}$$

where T is the detector threshold and σ_Λ^2 is the variance of the test statistic. Computations for the coordinates yield the same form: all requiring the analog of $(\partial\sigma_\Lambda^2)/(\partial a)$, which is a long but realistic calculation. The threshold is chosen using the nominal values (a_0, b_0, c_0) for α=0.05.

H. Examples

1. Example 1

In this first example, the parameters (a,b,c) are related by:

$$\left\| \begin{pmatrix} a & c \\ c & b \end{pmatrix} - \begin{pmatrix} a_0 & c_0 \\ c_0 & b_0 \end{pmatrix} \right\| = \|E\| = \sqrt{\epsilon} \tag{3.58}$$

For this example ($2 \star 2$ matrix), let $\|E\|$ represent the value of the natural norm of the generic linear group G1(2,IR) applied to E, also known as the *Frobenius Norm.* Then:

$$\|E\|^2 = (a - a_0)^2 + (b - b_0)^2 + 2(c - c_0)^2 \tag{3.59}$$

representing an ellipsoidal "egg" shape. We then impose the constraint that $\|E\|^2 = \epsilon$. Various ϵ can be considered, but we always employ an ϵ small enough so that all parameters (a,b,c) correspond to a positive definite matrix. We remark that the proper interpretation of the parameter surface is that the surfaces induces the "nominal" (a_0, b_0, c_0) and not the reverse. The nominal is simply a convenient center of gravity serving as a kind of average value, for choosing the detector threshold, for example.

The next step is to compute all the sets of points. The (a,b,c) points are bounded by the need to be on the manifold (see figure 32). In order to do so, we first need to divide the select manifold onto "strips" of different latitudes (every 5° for example) and different longitudes (every 10° for example). Therefore on each "slice" or latitude of the manifold, we will have 36 points. In order not to introduce any biased-ness into the sample density approach, a weighting factor is assigned to each latitude of the manifold (see figure 29).

Let's have ϵ be the tolerance on variance of the covariance (or the deviation of the covariance matrix from its nominal). Note that ϵ is also directly used to verify if the covariance matrix is positive definite (through the computation of each set of coordinates a, b and c).

2. Histograms

Histograms of the gradient sample density over the manifold are used to visualize if the detection algorithm is or is not robust. The more "compact" the histogram, the more it shows the absence of big slopes, and the less variable is the gradient. If in addition the point of "compactification" is of small value, then the detector might be called "robust."

All the scenarios that one could potentially encounter are represented in table

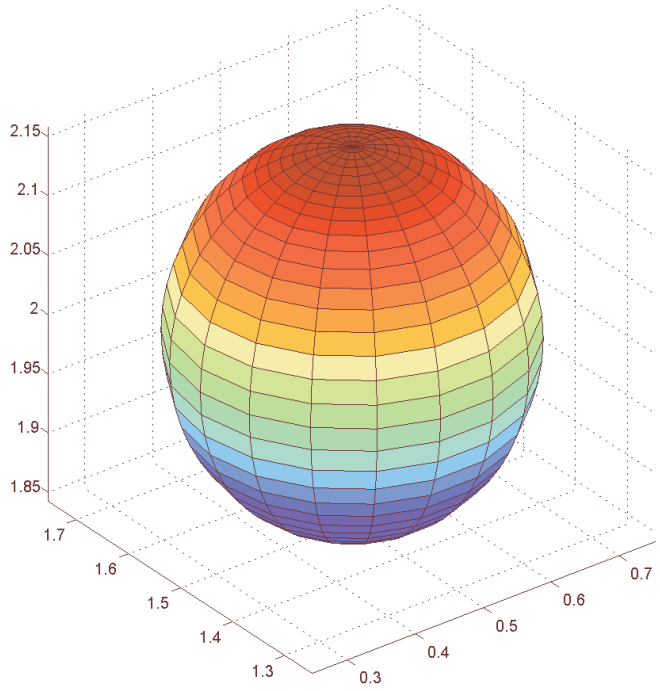

Fig. 32. Picking the right set of points

III. There are seven sets of two graphs that cover all those cases (see figures 33 to 39). For all the histograms the bin-size is 0.005. Each of the sample distributions (see table III) are based on 700 points each.

Table IV gives the corresponding means and variances for both the biased and unbiased case, for each case scenario. The weighted mean is used to combine average values from samples of the same population with different sample sizes:

$$\bar{x} = \frac{\sum_{i=1}^{n} w_i.x_i}{\sum_{i=1}^{n} w_i} \tag{3.60}$$

The weights w_i represent the bounds of the partial sample. In other applications they represent a measure for the reliability of the influence upon the mean by respective values.

In statistics, the concept of variance can also be used to describe a set of data.

Table III. All the different cases scenario for the 1st surface parameter

Case	a_0	b_0	c_0	s_1	s_2	ϵ	Threshold
1	1.5	2	0.5	1.5	0.5	0.05	5.5371
2	1.5	2	0.5	1.5	0.5	0.5	5.5371
3	1.5	2	0.5	1.5	0.5	0.005	5.5371
4	1.5	2	-0.5	1.5	0.5	0.05	6.4659
5	0.5	3	0.25	1.5	0.5	0.05	5.0253
6	1.5	2	0.5	0.5	0.5	0.05	2.1553
7	1.5	2	0.5	1.5	2	0.05	7.4662

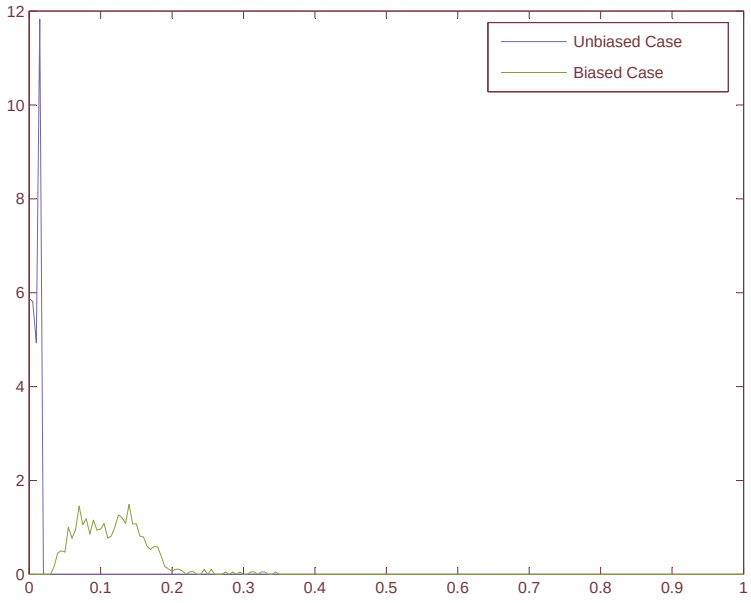

Fig. 33. Sample densities for the first surface parameter: case 1

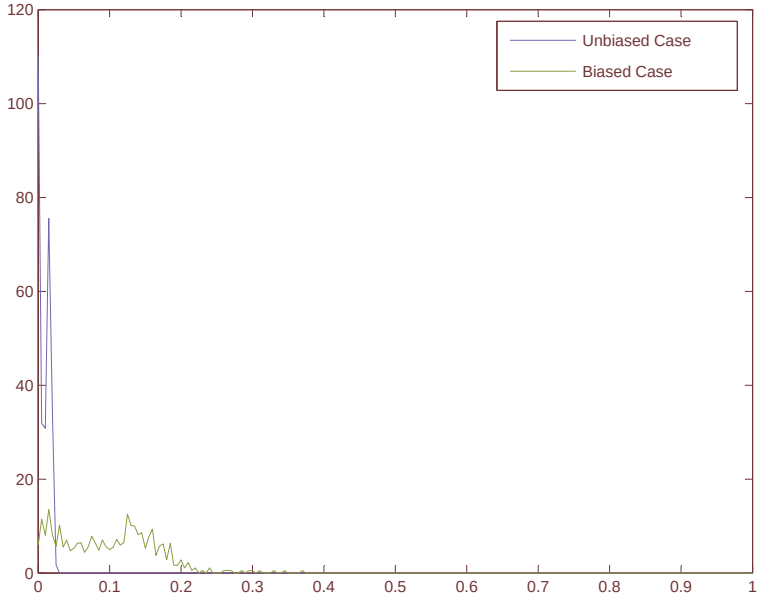

Fig. 34. Sample densities for the first surface parameter: case 2

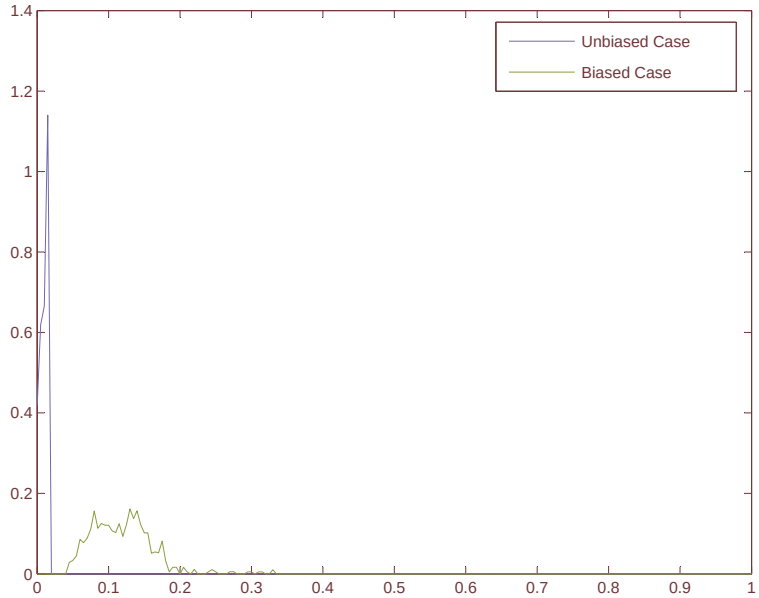

Fig. 35. Sample densities for the first surface parameter: case 3

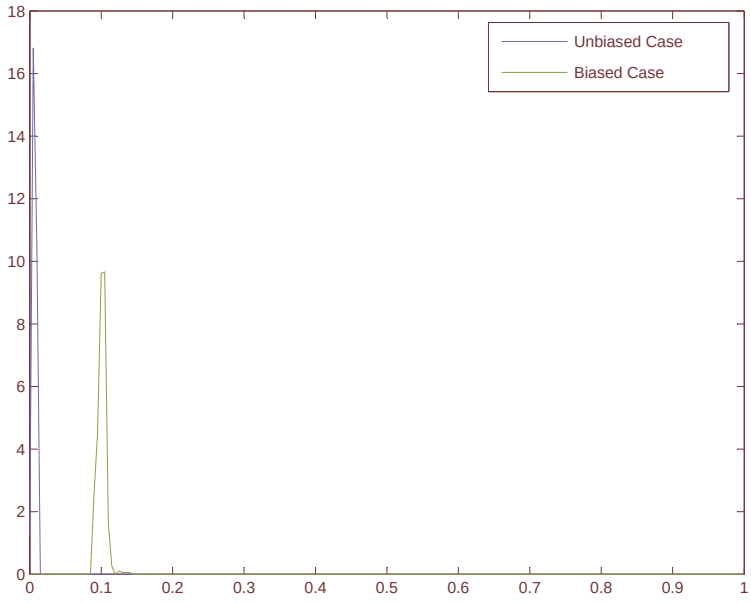

Fig. 36. Sample densities for the first surface parameter: case 4

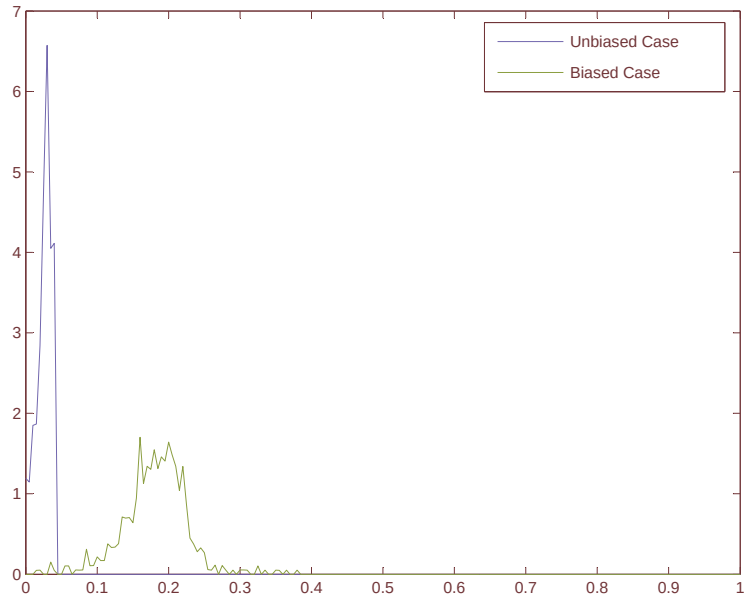

Fig. 37. Sample densities for the first surface parameter: case 5

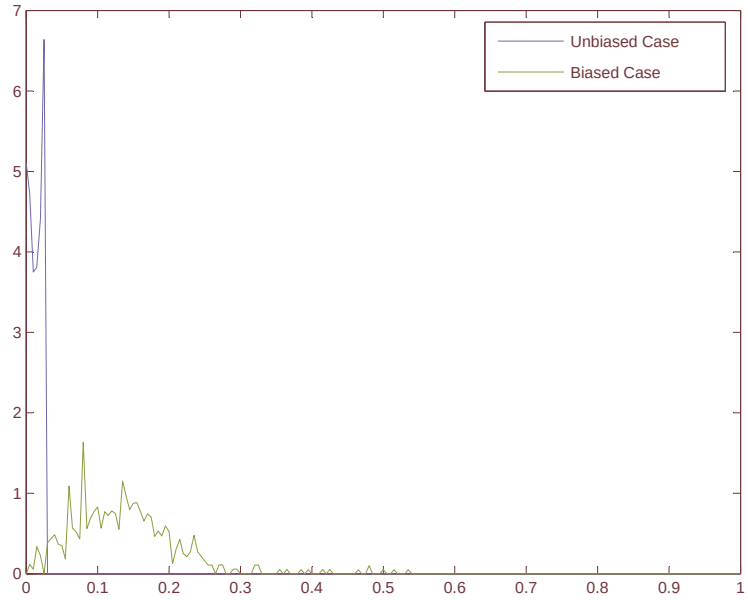

Fig. 38. Sample densities for the first surface parameter: case 6

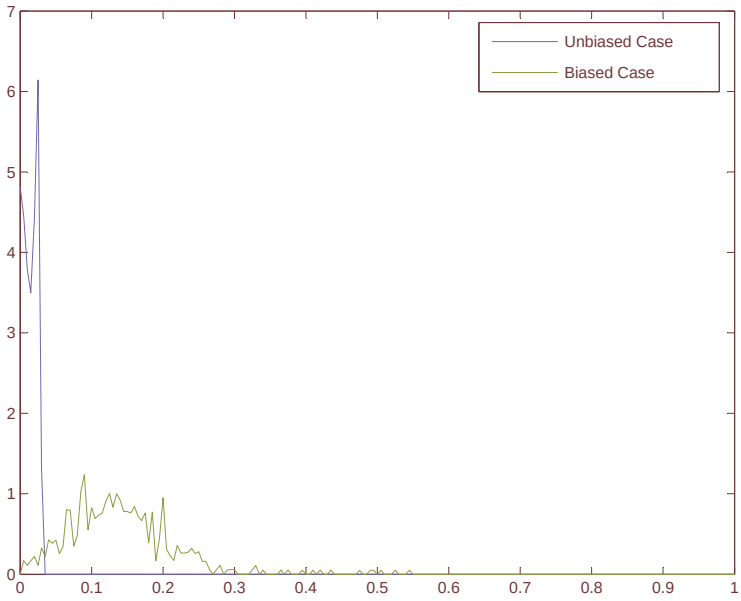

Fig. 39. Sample densities for the first surface parameter: case 7

Table IV. Means and variances for each case scenario for the 1st surface parameter

	Mean		Variance	
	Biased	Unbiased	Biased	Unbiased
case 1	0.11931	0.011492	0.4529	0.004623
case 2	0.10078	0.010683	0.5986	0.048921
case 3	0.12165	0.011942	0.0327	0.000348
case 4	0.10351	0.009103	0.2950	0.002491
case 5	0.18399	0.028262	0.9893	0.025275
case 6	0.14021	0.015596	0.7075	0.009061
case 7	0.14368	0.016364	0.7377	0.009955

When the set of data is a population, it is called the population variance. If the set is a sample, we call it the sample variance. The population variance of a population y_i where i=1,2,...,N is given by:

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(y_i - \mu)^2 \tag{3.61}$$

where μ is the population mean.

3. Example 2

In this example a new parameter surface is used. This time, the correlation coefficient is set up to stay constant. We have now:

$$\lambda = \frac{c}{\widetilde{a}\widetilde{b}} \tag{3.62}$$

where λ is the correlation coefficient that takes a value between $-1 < \lambda < 1$. Let's have $\tilde{a} = \sqrt{a}$ and $\tilde{b} = \sqrt{b}$, then $\tilde{a}, \tilde{b}, c$ can be used as parameters of the Gaussian,

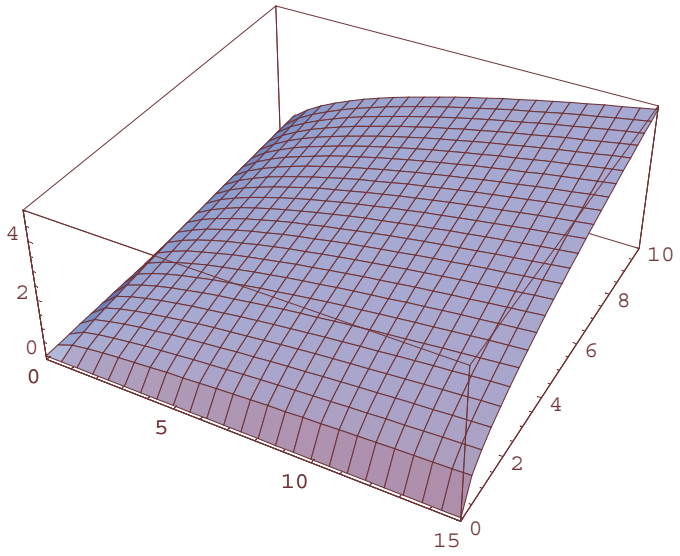

Fig. 40. Representation of the saddle surface for $\lambda = -0.2$

where $\tilde{a}, \tilde{b}$ are now the standard deviations instead of the variances. Note that the previous equation represents a true saddle surface parameter. Such a model might be appropriate in situations where one had much more confidence in the values of the correlation coefficient than the specific (a,b,c) values that are related to it. Various λ can be considered.

For a fixed and known value of λ, we will need to compute the weight factor.

$$\begin{aligned} weight\ factor &= \sqrt{1 + (\frac{\partial c}{\partial \tilde{a}})^2 + (\frac{\partial c}{\partial \tilde{b}})^2} \\ &= \sqrt{1 + \lambda^2 \tilde{a}^2 + \lambda^2 \tilde{b}^2} \end{aligned} \tag{3.63}$$

The surface parameter for the two different values of λ are represented in figures 40 and 41.

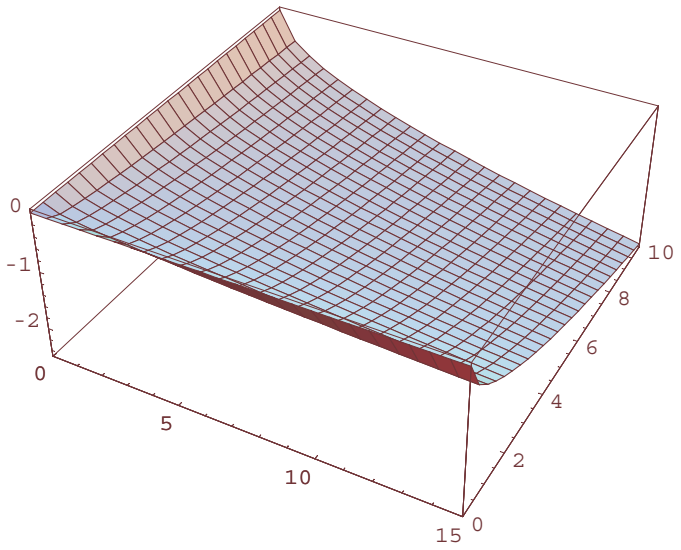

Fig. 41. Representation of the saddle surface for $\lambda = +0.2$

4. Histograms

Like the previous section, we first need to determine sets of points $(\tilde{a}, \tilde{b}, c)$ that respect the equation of the true saddle. In order to accomplish this, we partition the $(\tilde{a}, \tilde{b})$ plane using squares out as far as possible (the saddle surface has no boundaries). For each case scenario, (see table V) the focus of our interest is on both biased and unbiased, cases. All the sample distributions are based on 1500 points each.

By varying the nominal values, one will be able to visualize the robustness of the detection scheme. The changes will be made one value at the time, to improve the chance of visualizing the parameter's effect on the robustness of the detection scheme.

Note that naturally, the G matrix used to compute the tangent slopes is different from the "egg" surface. This is because $c(\tilde{a}, \tilde{b})$ has changed. Due to this change, we will now need to compute $\partial\alpha/\partial\tilde{a}$ and $\partial\alpha/\partial\tilde{b}$. The Chain's rule makes this computa-

Table V. All the different cases scenario for the 2nd surface parameter

case #	λ	$\tilde{a}$	$\tilde{b}$	s_1	s_2	$\tilde{c}_0$
1st	0.2	0.5	1	1.5	2	0.75
2nd	-0.2	0.5	1	1.5	2	-0.75
3rd	0.2	1.5	1	1.5	2	0.3
4th	0.2	0.5	1/4	1.5	2	0.025
5th	0.2	1.5	1	0.5	2	0.3
6th	0.2	1.5	1	0.5	1/2	0.3
7th	0.2	0.5	1	1.5	2	0.1

tion possible.

$$\begin{aligned} \frac{\partial \alpha}{\partial \tilde{a}} &= \frac{\partial \alpha}{\partial a} \times \frac{\partial a}{\partial \tilde{a}} \\ &= \Big(\frac{\partial \alpha}{\partial a}\Big|_{a=\tilde{a}^2} \Big) 2\tilde{a} \end{aligned} \tag{3.64}$$

$$\begin{aligned} \frac{\partial \alpha}{\partial \tilde{b}} &= \frac{\partial \alpha}{\partial b} \times \frac{\partial b}{\partial \tilde{b}} \\ &= \Big(\frac{\partial \alpha}{\partial b}\Big|_{b=\tilde{b}^2} \Big) 2\tilde{b} \end{aligned} \tag{3.65}$$

The reader should be aware that for the seventh case scenario, the value of ϵ is now equal to 0.005 instead of 0.05. Also, for all the cases L = 1 (there were found no differences in results with an increase or decrease of the value of L).

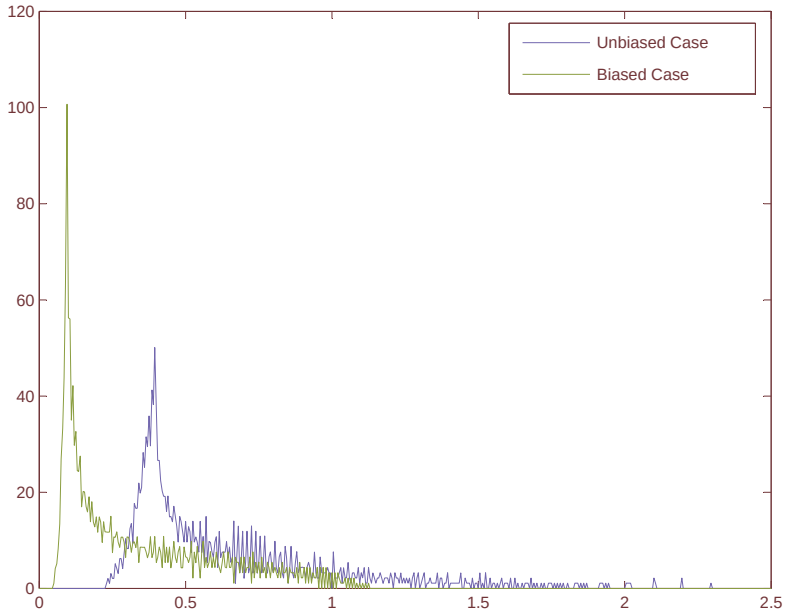

Fig. 42. Sample densities for the second surface parameter: case 1

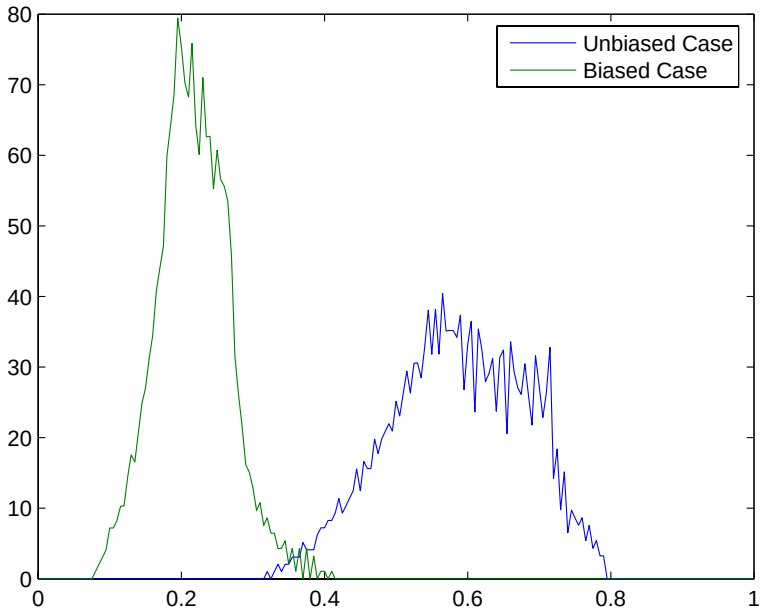

Fig. 43. Sample densities for the second surface parameter: case 2

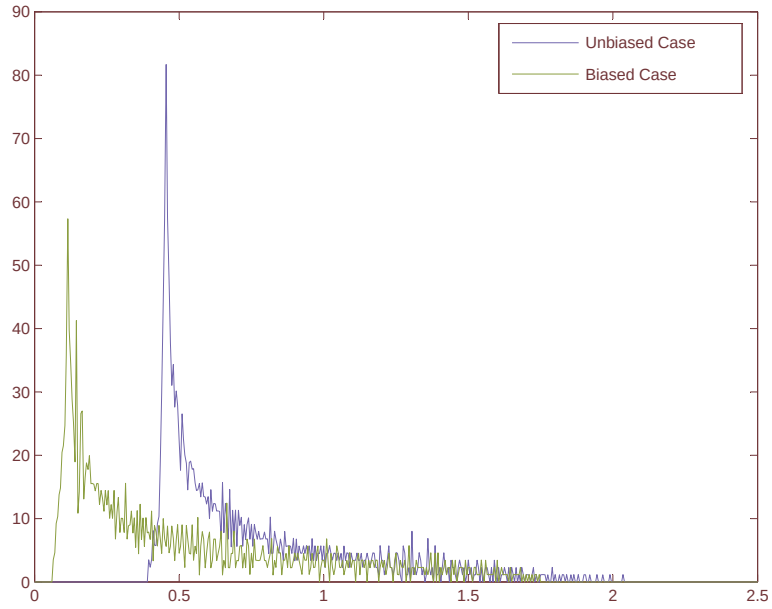

Fig. 44. Sample densities for the second surface parameter: case 3

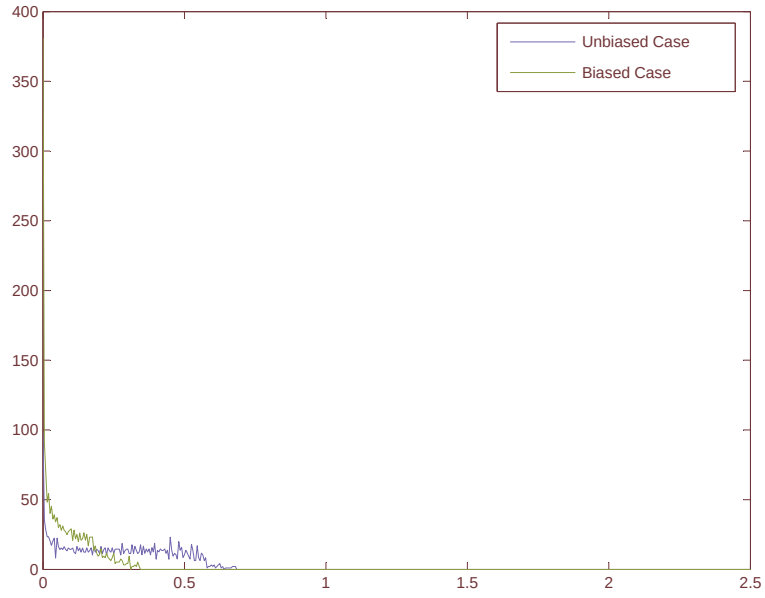

Fig. 45. Sample densities for the second surface parameter: case 4

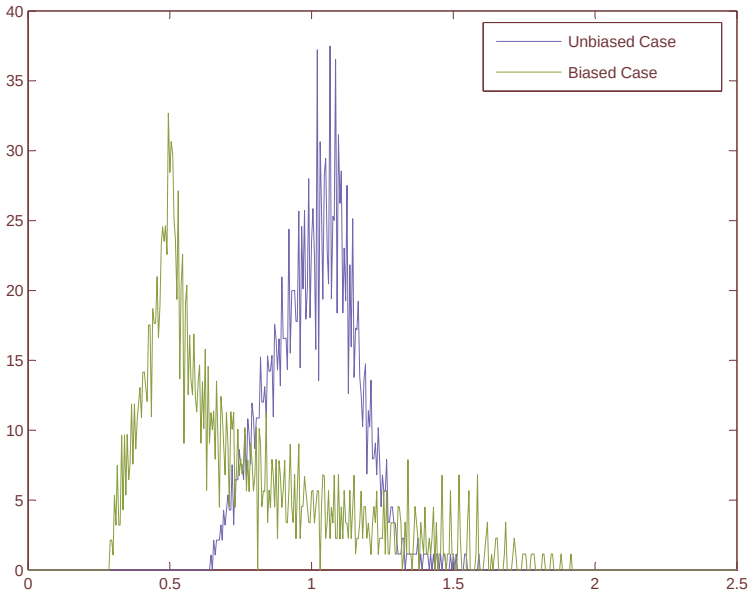

Fig. 46. Sample densities for the second surface parameter: case 5

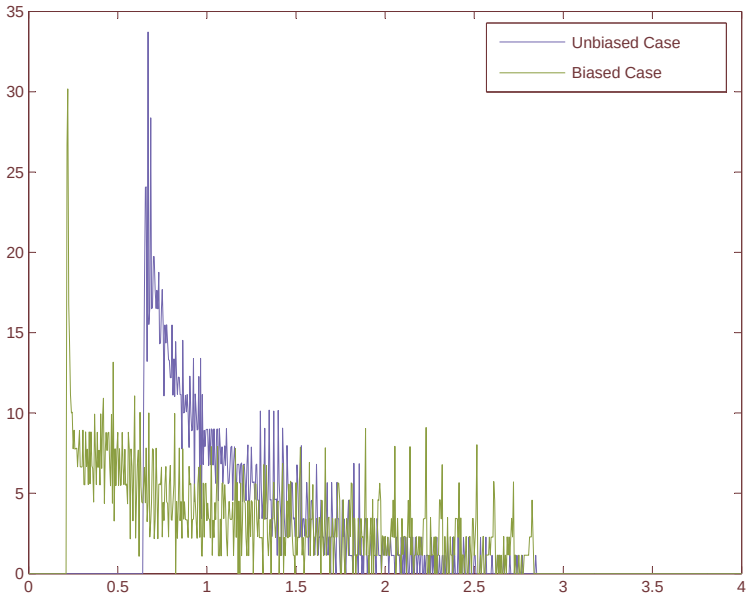

Fig. 47. Sample densities for the second surface parameter: case 6

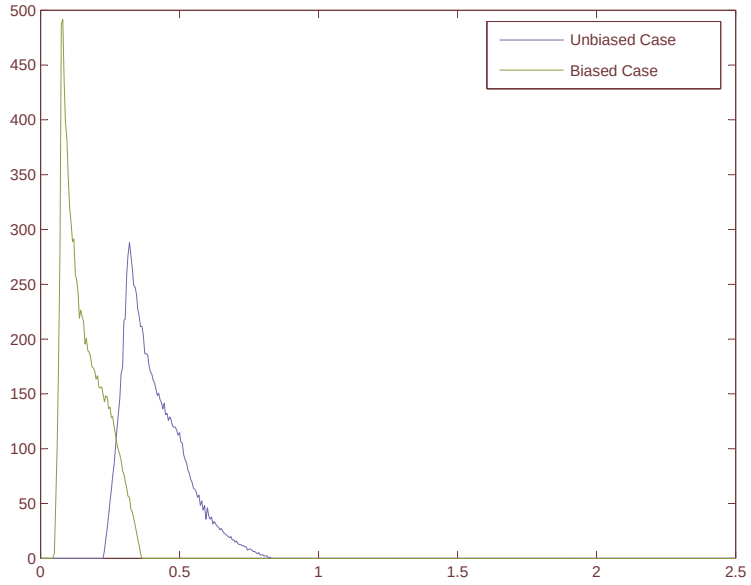

Fig. 48. Sample densities for the second surface parameter: case 7

I. Conclusion

Histograms of the gradient distribution over the manifold are used to visualize if the detection algorithm is or is not robust. The more "compact" the histogram, the more it shows the absence of big slopes, and the less variable is the gradient. If in addition the point of "compactification" is of small value, then the detector might be called "robust" (see figures 42 to 48).

In the case of the "egg" shape which corresponds to a closed parameter surface with a positive curvature, the sample densities obtained for the unbiased case are showing a high degree of robustness (absence of large slopes). The results look very consistent with sample densities compact around zero, and similar means/variances. Note that the biased results are not comparable. Since the unbiased approach is philosophically preferable in terms of telling the "true" robustness story, our results show that the extra difficulty in computing unbiased histograms is worth the effort.

In the case of the "saddle" surface, which this time corresponds to an open surface parameter with a negative curvature, the sample densities obtained show an absence of robustness in the scheme. Again, results of the biased approach are not comparable. To the view of those results we can conclude that the statement about robustness of an algorithm can only be made if it is associated with a parameter surface. The results depend completely on the choice of the parameter surface.

The next step of this work is an extension of our research: the case where three samples are involved. While the extension of this work from two to three samples sounds trivial, it will allow us to draw conclusions regarding the effect of non-stationarity on robustness (a related work would be the study of the effect of stationary noise –high frequency sampling for example– on robustness). For this work we will employ the unbiased approach exclusively.

CHAPTER IV

APPLICATION TO STATIONARITY VERSUS NON STATIONARITY USING A 6 DIMENSION MODEL

Many of the techniques used in telecommunications, image, speech, and radar signal processing rely on various degrees of measuring performance. The measure of robustness, is also based on the received data; for example the sampling speed has a direct effect on the dependency of the data. This chapter starts by extending the previous example from two samples/three dimensions to three samples/six dimensions. It then introduces the necessary computations one needs to make in order to state on the measure of robustness. Finally, the end of this chapter is dedicated to the effect of a larger sample size approach on robustness.

A. Stationary Versus Non Stationary

Statistically speaking, a stationary process (or strictly stationary process) is a stochastic process in which the probability density function of some random variable X does not change over time or position. As a result, parameters such as the mean and variance of the data set also do not change over time or position. For example, the measurement of white noise is considered a stationary process. Alternatively, the measurement of a slow sampling process is not stationary.

The linear detection of a random variable Y in terms of X, where X and Y are jointly Gaussian distributed random variables with covariance matrix K, using three

samples and non-stationary noise, becomes:

$$K = \begin{pmatrix} a & d/\sqrt{2} & f/\sqrt{2} \\ d/\sqrt{2} & b & e/\sqrt{2} \\ f/\sqrt{2} & e/\sqrt{2} & c \end{pmatrix}$$

The normalization by $\sqrt{2}$ is done to make the higher dimensional analysis easier; it does not compromise the robustness analysis.

The parameter surface obtained using the Frobenius norm, for the non-stationary case, would be a 5-sphere in 6D:

$$(a-a_0)^2 + (b-b_0)^2 + (c-c_0)^2 + (d-d_0)^2 + (e-e_0)^2 + (f-f_0)^2 = \epsilon \tag{4.1}$$

The same covariance matrix, using stationary data this time, would then be:

$$K = \begin{pmatrix} a & d/\sqrt{2} & f/\sqrt{2} \\ d/\sqrt{2} & a & d/\sqrt{2} \\ f/\sqrt{2} & d/\sqrt{2} & a \end{pmatrix}$$

The corresponding surface parameter would now be an 2-dimensional ellipsoid with a non-circular cross section in 3D:

$$3(a-a_0)^2 + 2(d-d_0)^2 + (f-f_0)^2 = \epsilon \tag{4.2}$$

1. The Neyman-Pearson Test Statistic

Under $\mathcal{H}_1$, we obtain:

$$f_{\underset{1}{Y}}(y_1, y_2, y_3) = \frac{1}{(2\pi)^{3/2}|K|^{1/2}} exp\Big(-\frac{1}{2}(y_1 - s_1\ y_2 - s_2\ y_3 - s_3)K^{-1}(y_1 - s_1\ y_2 - s_2\ y_3 - s_3)^T\Big) \tag{4.3}$$

Under $\mathcal{H}_0$, we obtain:

$$f_{\underset{0}{Y}}(y_1, y_2, y_3) = \frac{1}{(2\pi)^{3/2}|K|^{1/2}} exp\Big(-\frac{1}{2}(y_1\ y_2\ y_3)K^{-1}(y_1\ y_2\ y_3)^T\Big) \tag{4.4}$$

Therefore the Neyman-Pearson test statistics becomes:

$$\Lambda(y_1, y_2, y_3) = \frac{f_{\underset{1}{Y}}(y_1, y_2, y_3)}{f_{\underset{0}{Y}}(y_1, y_2, y_3)} \tag{4.5}$$

After few computations, the test statistic can easily be transformed into:

$$\Lambda(y_1, y_2, y_3) = \frac{exp\Big(-\frac{1}{2}(y_1 - s_1\ y_2 - s_2\ y_3 - s_3)K^{-1}(y_1 - s_1\ y_2 - s_2\ y_3 - s_3)^T\Big)}{exp\Big(-\frac{1}{2}(y_1\ y_2\ y_3)K^{-1}(y_1\ y_2\ y_3)^T\Big)} \tag{4.6}$$

Therefore the new test statistic comes down to the computation of $YK^{-1}Y^T$ for both hypotheses (under $\mathcal{H}_1$ and under $\mathcal{H}_0$).

2. The Stationary Case

The inverse of the K matrix is equal to:

$$K^{-1} = 1/(det\ K) \begin{pmatrix} a^2 - (d^2)/2 & (-ad)/(\sqrt{2}) + (df)/2 & (d^2)/2 - (af)/\sqrt{2} \\ (-ad)/\sqrt{2} + (df)/2 & a^2 - (f^2/2 & (-ad)/\sqrt{2} + (df)/2 \\ d^2/2 - (af)/\sqrt{2} & (-ad)/\sqrt{2} + (df)/2 & a^2 - (d^2)/2 \end{pmatrix}$$

Under $\mathcal{H}_1$, we then obtain:

$$\begin{aligned} YK^{-1}Y^T &= \frac{1}{det\ K}\Big[(a^2 - \frac{d^2}{2})(y_1 - s_1)^2 + (a^2 - \frac{f^2}{2})(y_2 - s_2)^2 + (a^2 - \frac{d^2}{2})(y_3 - s_3)^2 \\ &+ 2(\frac{df}{2} - \frac{ad}{\sqrt{2}})(y_1 - s_1)(y_2 - s_2) + 2(\frac{d^2}{2} - \frac{af}{\sqrt{2}})(y_1 - s_1)(y_3 - s_3) \\ &+ 2(\frac{df}{2} - \frac{ad}{\sqrt{2}})(y_2 - s_2)(y_3 - s_3)\Big] \end{aligned} \tag{4.7}$$

Under $\mathcal{H}_0$, we obtain:

$$\begin{aligned} YK^{-1}Y^T &= \frac{1}{det\ K}\Big[(a^2-\frac{d^2}{2})y_1^2+(a^2-\frac{f^2}{2})y_2^2+(a^2-\frac{d^2}{2})y_3^2 \\ &+ 2(\frac{df}{2}-\frac{ad}{\sqrt{2}})y_1y_2+2(\frac{d^2}{2}-\frac{af}{\sqrt{2}})y_1y_3+2(\frac{df}{2}-\frac{ad}{\sqrt{2}})y_2y_3\Big] \end{aligned} \tag{4.8}$$

The Neyman-Pearson test statistic is then equal to:

$$\Lambda(y_1,y_2,y_3)=(\underline{y}-\underline{m})K^{-1}(\underline{y}-\underline{m})^T-\underline{y}K^{-1}\underline{y}^T \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{4.9}$$

After few obvious simplifications, one can easily obtain the following equation:

$$\begin{aligned} \Lambda(y_1,y_2,y_3) &= \frac{1}{2}\Big(2s_1(a^2-\frac{d^2}{2})+2s_2(\frac{df}{2}-\frac{ad}{\sqrt{2}})+2s_3(\frac{d^2}{2}-\frac{af}{\sqrt{2}})\Big)y_1 \\ &+ \frac{1}{2}\Big(2s_1(\frac{df}{2}-\frac{ad}{\sqrt{2}})+2s_2(a^2-\frac{f^2}{2})+2s_3(\frac{df}{2}-\frac{ad}{\sqrt{2}})\Big)y_2 \\ &+ \frac{1}{2}\Big(2s_1(\frac{d^2}{2}-\frac{af}{\sqrt{2}})+2s_2(\frac{df}{2}-\frac{ad}{\sqrt{2}})+2s_3(\frac{a^2}{2}-\frac{d^2}{2})\Big)y_3 \\ &+ \frac{1}{2}\Big(s_1^2(\frac{d^2}{2}-a^2)+s_2^2(\frac{f^2}{2}-a^2)+s_3^2(\frac{d^2}{2}-a^2)+2s_1s_2(\frac{ad}{\sqrt{2}}-\frac{df}{2}) \\ &+ 2s_1s_3(\frac{af}{\sqrt{2}}-\frac{d^2}{2})+2s_2s_3(\frac{ad}{\sqrt{2}}-\frac{df}{2})\Big) \end{aligned} \tag{4.10}$$

It is interesting to notice that the Neyman-Pearson test statistic now looks like:

$$\Lambda = AY_1+BY_2+CY_3+D \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{4.11}$$

One can also "push" the D term into the test statistic without affecting the final result, obtaining the next expression:

$$\Lambda = AY_1+BY_2+CY_3 \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T^* \tag{4.12}$$

where A,B and C are constant terms and where $Y_1 \sim \mathcal{N}(0,a)$, $Y_2 \sim \mathcal{N}(0,b)$ and $Y_3 \sim \mathcal{N}(0,c)$. The next logical step is to find a solution to $\alpha = P(\Lambda < T|Noise)$ of the test statistic. In order to do so, one needs to compute the variance σ_Λ^2. Starting

from the definition of the variance, one can easily express the variance as a function of a,d,f,A,B and C. The variance is then equal to:

$$\begin{aligned}
\sigma_\Lambda^2 &= E[\Lambda^2] - E[\Lambda]^2 \\
&= E[(AY_1^2 + BY_2^2 + CY_3^2)^2] - E[AY_1^2 + BY_2^2 + CY_3^2]^2 \\
&= A^2E[Y_1^1] + B^2E[Y_2^2] + C^2E[Y_3^2] + 2ABE[Y_1Y_2] + 2ACE[Y_1Y_3] \\
&+ 2BCE[Y_2Y_3] \\
&= a(A^2 + B^2 + C^2) + 2AB\frac{d}{\sqrt{2}} + 2AC\frac{f}{\sqrt{2}} + 2BC\frac{d}{\sqrt{2}} \qquad (4.13)
\end{aligned}$$

where $E[Y_1Y_2]$=d, $E[Y_1Y_3]$=e, $E[Y_2Y_3]$=f.

For the computation of the value of the threshold $\widehat{T}$, one will employ nominal values, chosen according to the following table:

Table VI. Nominal values for the stationary case

Point	Value
a_0	1.5
d_0	2
f_0	0.5

For α the computations are as follows:

$$\begin{aligned}
\alpha &= Pr(\Lambda > \widehat{T}|Noise) \\
&= \int_{\widehat{T}}^{\infty} \frac{1}{\sqrt{2\pi}\sigma_\Lambda^2} e^{-\frac{y^2}{2\sigma_\Lambda^2}} dy \\
&= \frac{1}{2} - erf(\frac{\widehat{T}}{\sigma_\Lambda}) \qquad (4.14)
\end{aligned}$$

In order to get a value for the threshold $\widehat{T}$, one needs to compute the nominal variance. Using the nominal values one can obtain $\sigma^2_{\Lambda_0} = 11.344$. Replacing the nominal variance by its value one will then obtain the threshold value, where α=0.05:

$$\begin{aligned} 0.05 &= \frac{1}{2} - erf(\frac{\widehat{T}}{3.3680}) \\ 0.45 &= erf(.2969045\widehat{T}) \\ or\ 0.2969045\widehat{T} &= 1.64399 \\ \widehat{T} &= 5.5371 \end{aligned} \tag{4.15}$$

3. The Non-Stationary Case

For the non-stationary case, the inverse of the K matrix is then equal to:

$$K^{-1} = 1/(det\ K) \begin{pmatrix} bc - e^2/2 & ef/2 - (dc)/\sqrt{2} & (de)/2 - (bf)/\sqrt{2} \\ (ef)/2 - (cd)/\sqrt{2} & ac - f^2/2 & (df)/2 - (ae)/\sqrt{2} \\ (de)/2 - (bf)/\sqrt{2} & (df)/2 - (ae)/\sqrt{2} & ab - d^2/2 \end{pmatrix}$$

This time, under $\mathcal{H}_1$, we obtain:

$$\begin{aligned} YK^{-1}Y^T &= \frac{1}{det\ K}\Big[(bc - \frac{e^2}{2})(y_1 - s_1)^2 + (ac - \frac{f^2}{2})(y_2 - s_2)^2 + (\frac{ef}{2} - \frac{dc}{\sqrt{2}})(y_3 - s_3)^2 \\ &+ 2(\frac{de}{2} - \frac{bf}{\sqrt{2}})(y_1 - s_1)(y_3 - s_3) + 2(\frac{ef}{2} - \frac{dc}{\sqrt{2}})(y_1 - s_1)(y_2 - s_2) \\ &+ 2(\frac{df}{2} - \frac{ae}{\sqrt{2}})(y_2 - s_2)(y_3 - s_3)\Big] \end{aligned} \tag{4.16}$$

Also, under $\mathcal{H}_0$, we obtain:

$$\begin{aligned} YK^{-1}Y^T &= \frac{1}{det\ K}\Big[(bc - \frac{e^2}{2})y_1^2 + (ac - \frac{f^2}{2})y_2^2 + (\frac{ef}{2} - \frac{dc}{\sqrt{2}})y_3^2 \\ &+ 2(\frac{de}{2} - \frac{bf}{\sqrt{2}})y_1y_3 + 2(\frac{ef}{2} - \frac{dc}{\sqrt{2}})y_1y_2 + 2(\frac{df}{2} - \frac{ae}{\sqrt{2}})y_2y_3\Big] \end{aligned} \tag{4.17}$$

The Neyman-Pearson test statistic then becomes:

$$\Lambda(y_1, y_2, y_3) = (\underline{y} - \underline{m})K^{-1}(\underline{y} - \underline{m})^T - \underline{y}K^{-1}\underline{y}^T \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{4.18}$$

Using the previous simplifications, one can easily obtain the following equation:

$$\begin{aligned}
\Lambda(y_1, y_2, y_3) &= \Big(2s_1(bc - \frac{e^2}{2}) + 2s_2(\frac{ef}{2} - \frac{dc}{\sqrt{2}}) + 2s_3(\frac{de}{2} - \frac{bf}{\sqrt{2}})\Big)y_1 \\
&+ \Big(2s_1(\frac{ef}{2} - \frac{dc}{\sqrt{2}}) + 2s_2(ac - \frac{f^2}{2}) + 2s_3(\frac{df}{2} - \frac{ae}{\sqrt{2}})\Big)y_2 \\
&+ \Big(2s_1(\frac{de}{2} - \frac{bf}{\sqrt{2}}) + 2s_2(\frac{df}{2} - \frac{ae}{\sqrt{2}}) + 2s_3(\frac{ab}{2} - \frac{d^2}{2})\Big)y_3 \\
&+ \Big(s_1^2(\frac{e^2}{2} - bc) + s_2^2(\frac{f^2}{2} - ac) + s_3^2(\frac{d^2}{2} - ab) + 2s_1s_3(\frac{bf}{\sqrt{2}} - \frac{de}{2}) \\
&+ 2s_1s_2(\frac{dc}{\sqrt{2}} - \frac{ef}{2}) + 2s_2s_3(\frac{ae}{\sqrt{2}} - \frac{df}{2})\Big)
\end{aligned} \tag{4.19}$$

The Neyman-Pearson test statistic for the non-stationary case looks like the following equation:

$$\Lambda = AY_1 + BY_2 + CY_3 + D \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T \tag{4.20}$$

Again, "pushing" the D term into the test statistic, the final result will not be affected. Therefore:

$$\Lambda = AY_1 + BY_2 + CY_3 \underset{\mathcal{H}_0}{\overset{\mathcal{H}_1}{\gtrless}} T^* \tag{4.21}$$

where A,B and C are constant terms and where $Y_1 \sim \mathcal{N}(0, a)$, $Y_2 \sim \mathcal{N}(0, b)$ and $Y_3 \sim \mathcal{N}(0, c)$. The next logical step is to compute a solution to $\alpha = P(\Lambda < T|Noise)$ for the test statistic. One will again need to compute the nominal variance σ_Λ^2:

$$\begin{aligned}
\sigma_\Lambda^2 &= E[\Lambda^2] - E[\Lambda]^2 \\
&= E[(AY_1^2 + BY_2^2 + CY_3^2)^2] - E[AY_1^2 + BY_2^2 + CY_3^2]^2 \\
&= aA^2 + bB^2 + cC^2 + 2AB\frac{d}{\sqrt{2}} + 2AC\frac{f}{\sqrt{2}} + 2BC\frac{e}{\sqrt{2}}
\end{aligned} \tag{4.22}$$

where $E[Y_1Y_2]$=d, $E[Y_1Y_3]$=e, $E[Y_2Y_3]$=f.

B. Matching ϵ_{2D} and ϵ_{5D}

For two-dimensional surfaces embedded in R^3, there are two kinds of curvature: Gaussian curvature and Mean curvature. To compute these at a given point of the surface, one needs to consider the intersection of the surface with a plane containing a fixed normal vector at the point. This intersection is a plane curve and has a curvature. If we vary the plane, this curvature will change. Furthermore, there are two extremal values - the maximal and the minimal curvature, called the principal curvatures, k1 and k2, the extremal directions are called principal directions. Here we adopt the convention that a curvature is taken to be positive if the curve turns in the same direction as the surface's chosen normal, otherwise negative.

The Gaussian curvature [39], named after Carl Friedrich Gauss, is equal to the product of the principal curvatures, k1*k2. It is positive for spheres, negative for one sheet hyperboloids, and zero for planes. It determines whether a surface is locally convex (when it is positive) or locally saddle (when it is negative).

The above definition of Gaussian curvature is extrinsic in that it uses the surface's embedding in R^3, normal vectors, external planes etc. Gaussian curvature is, however, in fact an intrinsic property of the surface. This means it does not depend on the particular embedding of the surface. Intuitively, this means that ants living on the surface could determine the Gaussian curvature. Formally, Gaussian curvature only depends on the Riemannian metric of the surface (see [40]).

The matching of the values of ϵ is necessary to enable one to compare the obtained results. This can be done by looking at the average Gauss-Kronecker curvature over

a surface. The curvature at a point on a manifold is given by:

$$K_n = \lambda_1 ... \lambda_n \tag{4.23}$$

where n is the dimension of the manifold and the λ_i are the *principle curvatures*. The average curvature is then equal to:

$$K_{n,Av} = \frac{\int_M K_n dV_n}{volume\ M} \tag{4.24}$$

where the volume is actually the "surface area" of the surface parameter. Since the curvature is an n-dimensional concept ($K_n = \prod_{i=1}^{n} \lambda_i$), we can equate:

$$(K_{5,Av})^{1/5} = (K_{2,Av})^{1/2} \tag{4.25}$$

Since n=5 has $m_5 = sphere\ of\ radius\sqrt{\epsilon_5}$, we have $K_n = (\frac{1}{\sqrt{\epsilon_5}})^5$, so:

$$\begin{aligned} K_{5,Av} &= \frac{\int_{m_5} \epsilon_5^{-5/2} dV_5}{volume\ m_5} \\ &= \frac{\epsilon_5^{-5/2} \int_{m_5} dV_5}{volume\ m_5} \\ &= \frac{\epsilon_5^{-5/2}\ volume\ m_5}{volume\ m_5} \\ &= \epsilon_5^{-5/2} \end{aligned} \tag{4.26}$$

For the two dimensions, we have to use the Gauss-Bonnet theorem [41, 42, 43] because the manifold is non spherical. The Gauss-Bonnet theorem in differential geometry is an important statement about surfaces which connects their geometry (in the sense of curvature) to their topology (in the sense of the Euler characteristic).

Suppose M is a compact two-dimensional orientable Riemannian manifold with boundary ∂M. Denote by K the Gaussian curvature at points of M, and by k_g the

geodesic curvature at points of ∂M. Then, the Gauss-Bonnet yields:

$$\int_M K dA + \int_{\partial M} k_g dS = 2\pi\chi(M) \tag{4.27}$$

where $\chi(M)$ is the Euler characteristic of M.

The theorem applies in particular if the manifold does not have a boundary, in which case the integral $\int_{\partial M} k_g \, ds$ can be omitted. Therefore, for two dimensions:

$$\chi(M) = \frac{1}{2\pi} \int_{M_2} K_2 dV_2 \tag{4.28}$$

Since M_2 (2 dimension ellipsoid) is diffeomorphic to a 2-sphere, $\chi(M_2) = 2$, thus:

$$2 = \frac{1}{2\pi} \int_{M_2} K_2 dV_2 \tag{4.29}$$

or

$$4\pi = \int_{M_2} K_2 dV_2 \tag{4.30}$$

Also, we know that:

$$\begin{aligned} K_{2,Av} &= \frac{\int_{M_2} K_2 dV_2}{vol\ M_2} \\ &= \frac{4\pi}{vol\ M_2} \\ &= \frac{4\pi}{surface\ area\ of\ (3a^2 + 2d^2 + f^2 = \epsilon_2)} \\ &= \frac{4\pi}{A_2(\epsilon_2)} \end{aligned} \tag{4.31}$$

Finally we obtain:

$$(\epsilon_5^{-5/2})^{1/5} = (\frac{4\pi}{A_2(\epsilon_2)})^{1/2} \tag{4.32}$$

or

$$\epsilon_5 = \frac{A_2(\epsilon_2)}{4\pi} \tag{4.33}$$

where $A_2(\epsilon_2)$ is the surface area of $3a^2 + 2d^2 + f^2 = \epsilon_2$.

C. Surface Area

If we express the surface area as a function of f, we have:

$$f = (\epsilon - 3a^2 - 2d^2)^{1/2} \tag{4.34}$$

So we have for 1/2 of the area:

$$\frac{1}{2}area = \int\limits_{(a,b)} \int\limits_{region} \sqrt{1 + (\frac{\partial f}{\partial a})^2 + (\frac{\partial f}{\partial d})^2} dadd \tag{4.35}$$

The (a,b) region is $\{(a, b) : \epsilon = 3a^2 + 2d^2\}$. If one wants to quickly verify the results, one can use, for example, the geometric mean and approximate the ellipsoid into a sphere of radius $r = (\sqrt{\epsilon}.\sqrt{\epsilon/2}.\sqrt{\epsilon/3})^{1/3}$. The total area being equal to $4\pi r^2$.

So, one has:

$$\frac{1}{2}area = \int\limits_{-\sqrt{\epsilon/3}}^{\sqrt{\epsilon/3}} \int\limits_{-\sqrt{\frac{\epsilon-3a^2}{2}}}^{\sqrt{\frac{\epsilon-3a^2}{2}}} \sqrt{1 + (\frac{\partial f}{\partial a})^2 + (\frac{\partial f}{\partial d})^2} dadd \tag{4.36}$$

After few simplifications, one will obtain:

$$\frac{1}{2}area = \int\limits_{-\sqrt{\epsilon/3}}^{\sqrt{\epsilon/3}} \int\limits_{-\sqrt{\frac{\epsilon-3a^2}{2}}}^{\sqrt{\frac{\epsilon-3a^2}{2}}} \sqrt{\frac{6a^2 + 2d^2 + \epsilon}{-3a^2 - 2d^2 + \epsilon}} dadd \tag{4.37}$$

If $\epsilon = 0.05$, for example, one gets 1/2 *area* equals to 3.16.

D. Weighting Factors and Point Selection

For the stationary case, with f as a function of a and d one has for the weighting factor:

$$Weighting\ Factor = \sqrt{(1 + (\frac{\partial f}{\partial a})^2 + (\frac{\partial f}{\partial d})^2)} \tag{4.38}$$

So after few simplifications, one has:

$$Weighting\ Factor = \Big[\frac{\epsilon + 6(a-a_0)^2 + 2(d-d_0)^2}{\epsilon - 3(a-a_0)^2 - 2(d-d_0)^2}\Big]^{1/2} \tag{4.39}$$

One should note that the weighting factor is independent of f, so if one decides to compute the upper half of $f = f_0 \pm (...)^{1/2}$, one will get the same weighting factor for the lower part (a,d) too.

For the points selection, let's again express f as a function of (a,d). Remember that it does not matter which side we choose (upper or lower part of the ellipsoid in $\mathcal{R}^3$) because the weighting factor will equalize (independent of f). Hence, using the Frobenius norm one has:

$$f = f_0 + (\epsilon - 3(a-a_0)^2 - 2(d-d_0)^2)^{1/2} \tag{4.40}$$

For the point selection, we will first need to shift the center of the ellipsoid at the origin. we then will have $\widehat{a} = a - a_0$, $\widehat{d} = d - d_0$ and $\widehat{f} = f - f_0$. Inside the $(\widehat{a}, \widehat{d})$ plane (where $\widehat{f} = 0$), we will have:

$$3\widehat{a}^2 - 2\widehat{d}^2 < \epsilon \tag{4.41}$$

The previous equation is represented in figure 49. The grid needs to be at the interior of the ellipse allowing one to find sets of $(\widehat{a}, \widehat{d})$ points. From this, the next step is to get the (a,d) points via $a = \widehat{a} + a_0$ and $d = \widehat{d} + d_0$. Then using equation 4.40 to get f, one can produce sets of (a,d,f) points for the upper half of the ellipsoid. The same procedure is used to produce sets of points for the lower part of the ellipsoid.

For the non-stationary case, the point assignment and the weighting factor are done simultaneously (for a sphere in 6D). One will first start with a sphere in 3D and then expand our reasoning to a sphere in 6D. For a sphere in 3D, centered at the

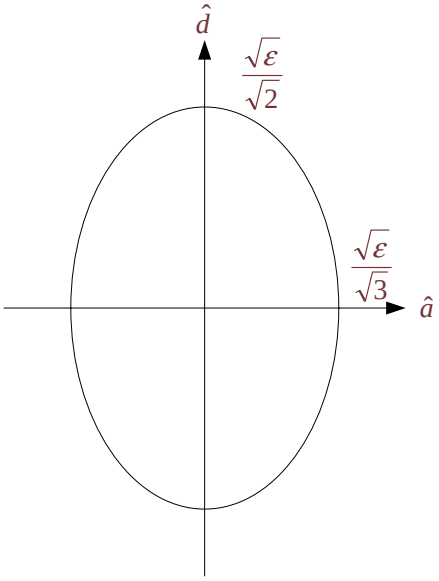

Fig. 49. Selection of the sets of points for the stationary case

origin, one has (see figure 50) a radius in the (x_1, x_2) plane of $\sqrt{\epsilon}\cos\phi_2$. Hence:

$$\begin{aligned} x_1 &= (\cos\phi_1)\sqrt{\epsilon}\cos\phi_2 = \sqrt{\epsilon}\cos\phi_2\cos\phi_1 \\ x_2 &= (\sin\phi_1)\sqrt{\epsilon}\cos\phi_2 = \sqrt{\epsilon}\cos\phi_2\sin\phi_1 \\ x_3 &= \sqrt{\epsilon}\sin\phi_2 \end{aligned} \tag{4.42}$$

where $0 \leq \phi_1 < 2\pi$ and $-\pi/2 \leq \phi_2 \leq \pi/2$.

The weighting factor is then equal to the surface area element for the sphere in polar coordinates (see figure 51). As ϕ_2 is incremented by $d\phi_2$, it introduces an increment in latitude of $\sqrt{\epsilon}d\phi_2$. As ϕ_1 is incremented, it induces an increment in longitude of $(\sqrt{\epsilon}\cos\phi_2)d\phi_1$. After noticing that all the increments are orthogonal, the surface area element is equals to:

$$\begin{aligned} Surface\ area\ element &= \sqrt{\epsilon}d\phi_2\sqrt{\epsilon}\cos\phi_2 d\phi_1 \\ &= \epsilon\cos\phi_2 d\phi_1 d\phi_2 \end{aligned} \tag{4.43}$$

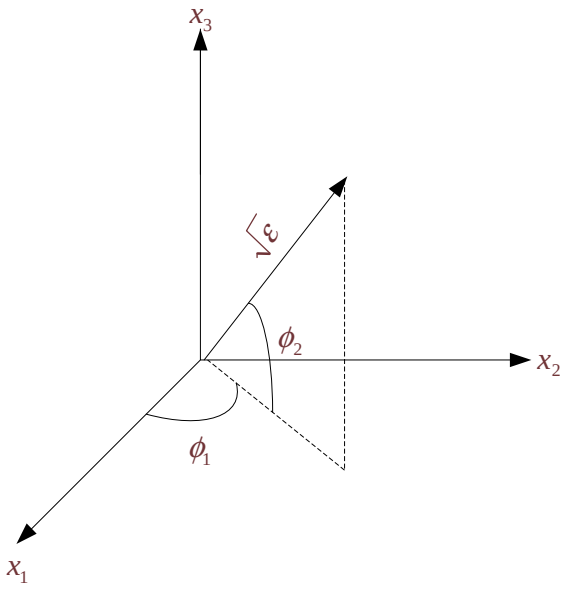

Fig. 50. Selection of the sets of points for the non-stationary case

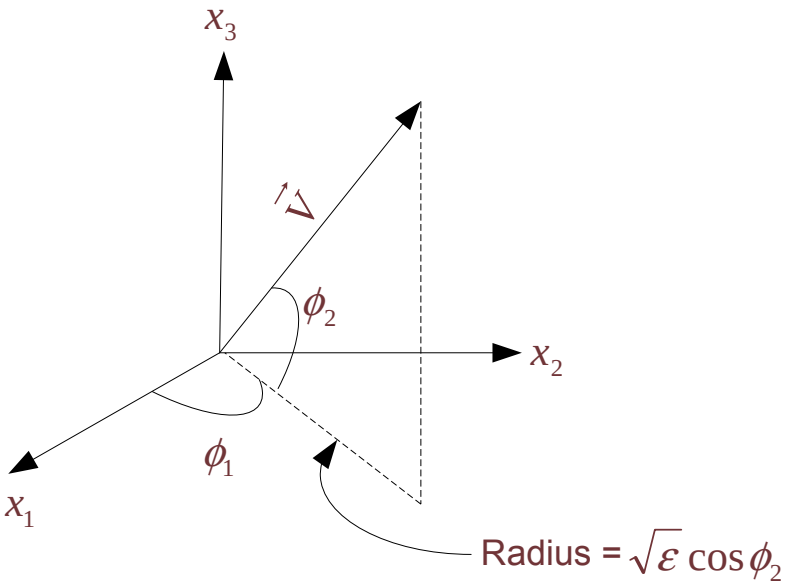

Fig. 51. The selection of the sets of points using polar coordinates for the non-stationary case

The weighting factor for the sphere in 3D then becomes:

$$Weighting\ factor\ 3D = \epsilon|\cos\phi_2| \tag{4.44}$$

Now, one needs to expand the previous approach to a sphere in 4D. Therefore, one will need to project down into the (x_1, x_2, x_3) space, where the vector $\overrightarrow{V}$ is at the angle in 4D of ϕ_3 with respect to the (x_1, x_2, x_3) space. Thus the "radius" in (x_1, x_2, x_3) is no longer $\sqrt{\epsilon}$ but $\sqrt{\epsilon}\cos\phi_3$. One hence obtain:

$$\begin{aligned} x_1 &= (\sqrt{\epsilon}\cos\phi_3)\cos\phi_2\cos\phi_1 = \sqrt{\epsilon}\cos\phi_3\cos\phi_2\cos\phi_1 \\ x_2 &= \sqrt{\epsilon}\cos\phi_3\cos\phi_2\sin\phi_1 \\ x_3 &= \sqrt{\epsilon}\cos\phi_3\sin\phi_2 \\ x_4 &= \sqrt{\epsilon}\sin\phi_3 \end{aligned} \tag{4.45}$$

where $0 \leq \phi_1 < 2\pi$, $-\pi/2 \leq \phi_i \leq \pi/2$, for i = 2,3. Now, the surface area element is equal to a surface area element of a sphere in 3D with radius $\sqrt{\epsilon}\cos\phi_3$ times $\sqrt{\epsilon}d\phi_3$. Hence:

$$\begin{aligned} Surface\ area\ element &= (\sqrt{\epsilon}d\phi_3)(\sqrt{\epsilon}\cos\phi_3 d\phi_2)(\sqrt{\epsilon}\cos\phi_3\cos\phi_2 d\phi_1) \\ &= \varepsilon^{3/2}\cos^2\phi_3\cos\phi_2 d\phi_1 d\phi_2 d\phi_3 \end{aligned} \tag{4.46}$$

The weighting factor for the 4D sphere then becomes:

$$Weighting\ factor\ 4D = \epsilon^{3/2}\cos^2\phi_3|\cos\phi_2| \tag{4.47}$$

Now, with inductive reasoning, for a sphere in 5D we have:

$$\begin{aligned} x_1 &= \sqrt{\epsilon}\cos\phi_4\cos\phi_3\cos\phi_2\cos\phi_1 \\ x_2 &= \sqrt{\epsilon}\cos\phi_4\cos\phi_3\cos\phi_2\sin\phi_1 \\ x_3 &= \sqrt{\epsilon}\cos\phi_4\cos\phi_3\sin\phi_2 \\ x_4 &= \sqrt{\epsilon}\cos\phi_4\sin\phi_3 \\ x_5 &= \sqrt{\epsilon}\sin\phi_4 \end{aligned} \tag{4.48}$$

where $0 \leq \phi_1 \leq 2\pi$ and $-\pi/2 \leq \phi_i \leq \pi/2$, i = 2,3,4. The surface area element is then equal to:

$$\begin{aligned} Surface\ area\ element &= \sqrt{\epsilon}d\phi_4(surface\ area\ element\ of \\ &\qquad 4D\ sphere\ with\ radius\ \sqrt{\epsilon}\cos\phi_4) \\ &= \sqrt{\epsilon}d\phi_4(\sqrt{\epsilon}\cos\phi_4)^3\cos^2\phi_3|\cos\phi_2|d\phi_1 d\phi_2 d\phi_3 \end{aligned} \tag{4.49}$$

The weighting factor of a sphere in 5D is then equal to:

$$Weighting\ factor\ 5D = \epsilon^2|\cos^3\phi_4\cos^2\phi_3\cos\phi_2| \tag{4.50}$$

The same reasoning is applied to a sphere in 6D. We then obtain:

$$\begin{aligned} x_1 &= \sqrt{\epsilon}\cos\phi_5\cos\phi_4\cos\phi_3\cos\phi_2\cos\phi_1 \\ x_2 &= \sqrt{\epsilon}\cos\phi_5\cos\phi_4\cos\phi_3\cos\phi_2\sin\phi_1 \\ x_3 &= \sqrt{\epsilon}\cos\phi_5\cos\phi_4\cos\phi_3\sin\phi_2 \\ x_4 &= \sqrt{\epsilon}\cos\phi_5\cos\phi_4\sin\phi_3 \\ x_5 &= \sqrt{\epsilon}\cos\phi_5\sin\phi_4 \\ x_6 &= \sqrt{\epsilon}\sin\phi_5 \end{aligned} \tag{4.51}$$

where $0 \leq \phi_1 \leq 2\pi$ and $-\pi/2 \leq \phi_i \leq \pi/2$, i = 2,3,4,5. The weighting factor of a sphere in 6D is then equal to:

$$Weighting\ factor\ 6D = \epsilon^{5/2} |\cos^4 \phi_5 \cos^3 \phi_4 \cos^2 \phi_3 \cos \phi_2| \tag{4.52}$$

The procedure to pick the points is as follows: First partition ϕ_1 from 0 to 2π and partition the other angles ϕ_2 to ϕ_5 from $-\pi/2$ to $\pi/2$ yielding points $(\phi_1, \phi_2, \phi_3, \phi_4, \phi_5)$. Then, one should use the following transformation: $a = x_1 + a_0$, $b = x_2 + b_0$, $c = x_3 + c_0$,$d = x_4 + d_0$, $e = x_5 + e_0$ and $f = x_6 + f_0$. One will then use these to evaluate the partial derivatives $\partial d/\partial a$, etc... and finally compute the weighting factor from $(\phi_2, \phi_3, \phi_4, \phi_5)$ associated with the points.

E. Directional Derivatives

The computations of the directional derivative $(D_{\vec{X}} h) = \sqrt{\nabla h G^{-1} \nabla h^T}$ depends directly on the approach used, but does not depend on the choice of the coordinates (provided the underlying manifold remains fixed). However, this follows from the classical interpretation of directional derivative as a limit of ∇h over the arc length, where the answer computes to be independent of the curve chosen as long as the tangent to the curve at P is fixed.

When one uses the stationary approach, the gradient of three variables embedded into 2D becomes (the parameter surface is defined as c=c(a,b)):

$$\frac{\partial}{\partial a} = (1, 0, \frac{\partial f}{\partial a}) \tag{4.53}$$

$$\frac{\partial}{\partial d} = (0, 1, \frac{\partial f}{\partial b}) \tag{4.54}$$

Inheriting the inner product from $\mathcal{R}^3$

$$g_{11} = 1 + (\frac{\partial c}{\partial a})^2 \tag{4.55}$$

$$g_{22} = 1 + (\frac{\partial c}{\partial b})^2 \tag{4.56}$$

$$g_{12} = g_{21} = \frac{\partial c}{\partial a}.\frac{\partial c}{\partial b} \tag{4.57}$$

We then remark that this choice of coordinates is for convenience, and the same result will be obtained for an alternative choice. Thus, the result is without bias. If the performance function is $P = h(a, b, c) \equiv h(a, b)$, we then obtain:

$$\left(D_{\vec{X}}h\right)\Big|_{Extreme} = \sqrt{\nabla h \underline{G}^{-1} \nabla h^T} \tag{4.58}$$

$$\nabla h = \left(\frac{\partial h}{\partial a} \; \frac{\partial h}{\partial b}\right) \tag{4.59}$$

and finally:

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix}$$

On the other side, the gradient of six variables embedded into 5D becomes (the parameter surface is defined as f(a,b,c,d,e)):

$$\frac{\partial}{\partial a} = (1, 0, 0, 0, 0, \frac{\partial f}{\partial a}) \tag{4.60}$$

$$\frac{\partial}{\partial b} = (0, 1, 0, 0, 0, \frac{\partial f}{\partial b}) \tag{4.61}$$

$$\frac{\partial}{\partial c} = (0, 0, 1, 0, 0, \frac{\partial f}{\partial c}) \tag{4.62}$$

$$\frac{\partial}{\partial d} = (0, 0, 0, 1, 0, \frac{\partial f}{\partial d}) \tag{4.63}$$

$$\frac{\partial}{\partial e} = (0, 0, 0, 0, 1, \frac{\partial f}{\partial e}) \tag{4.64}$$

Inheriting the inner product from $\mathcal{R}^6$

$$g_{11} = (\frac{\partial f}{\partial a}, \frac{\partial f}{\partial a}) = 1 + 0^2 + 0^2 + 0^2 + 0^2 + (\frac{\partial f}{\partial a})^2 \tag{4.65}$$

$$g_{12} = (\frac{\partial f}{\partial a}, \frac{\partial f}{\partial b}) = \frac{\partial f}{\partial a}\frac{\partial f}{\partial b} \tag{4.66}$$

$$g_{13} = (\frac{\partial f}{\partial a}, \frac{\partial f}{\partial c}) = \frac{\partial f}{\partial a}\frac{\partial f}{\partial c} \tag{4.67}$$

$$\vdots \tag{4.68}$$

$$g_{22} = (\frac{\partial f}{\partial b}, \frac{\partial f}{\partial b}) = 1 + (\frac{\partial f}{\partial a})^2 + 0^2 + 0^2 + 0^2 + 0^2 \tag{4.69}$$

where

$$G = \begin{pmatrix} g_{11} & \cdots & g_{15} \\ \vdots & \ddots & \vdots \\ g_{51} & \cdots & g_{55} \end{pmatrix}$$

Note:

$$\nabla h = (\frac{\partial \alpha}{\partial a}, \frac{\partial \alpha}{\partial b}, \ldots, \frac{\partial \alpha}{\partial e}) \tag{4.70}$$

F. Results

The results are obtained using two different signal vectors (a small one and a large one), and two different types of correlations (positive and negative). They are combined with two different values of ϵ (one close to ϵ_{max} and a smaller value of ϵ) allowing us to generate eight examples of gradient distributions for the stationary and non-stationary cases superimposed.

In each case, the results are obtained using a bin-size of $5 * 10^{-3}$. For each graph, we were forced to represent the normalized values of each sample densities because of the scaling effect on the results that ϵ_{5D} (non-stationary approach) has. Note that the distribution shapes are similar; we urge the reader to resist the temptation to draw comparative conclusions between non-stationary and stationary at this point.

Table VII. All the different cases scenario for both stationary and non-stationary

Case	a_0	d_0	f_0	s_1	s_2	s_3	ϵ_{2D}	ϵ_{5D}
1	1.5	3/4	1/2	1	1/4	1/8	1.05	0.60
2	1.5	3/4	1/2	1	1/4	1/8	0.15	0.0857
3	1.5	3/4	1/2	2	1.5	1.75	1.05	0.60
4	1.5	3/4	1/2	2	1.5	1.75	0.15	0.0857
5	1.5	-3/4	1/2	1	1/4	1/8	1.05	0.60
6	1.5	-3/4	1/2	1	1/4	1/8	0.15	0.0857
7	1.5	-3/4	1/2	2	1.5	1.75	1.05	0.60
8	1.5	-3/4	1/2	2	1.5	1.75	0.15	0.0857

The value of ϵ_{max} represents the maximum value that ϵ can take and still generate *only* sets of positive definite points. For the stationary case, we have $\epsilon_{2Dmax} = 1.092$. The corresponding value of ϵ_{5Dmax} is equal to 0.6238. The two values used for the simulations are $\epsilon_{2Dupper} = 1.05$ and $\epsilon_{2Dlower} = 0.15$. The corresponding values of ϵ_{5D} are respectively 0.6 and 0.0857.

For each sample density, its mean and variance are calculated. This is expected to allow us to state on the measure of robustness. All the results are presented in figures 52, 53, 54, 55, 56, 57, 58 , 59 and for a very small value of ϵ in figure 60. One of the requirements for all those sample densities is the need for positive definiteness for all sets of points. Each corresponding G matrix formed with those points has the particularity of being real and symmetrical forming an Hermitian

Table VIII. Means and variances for each case scenario for both the stationary and non-stationary approach

	Stationary		Non-Stationary	
Case	Mean	Variance	Mean	Variance
case 1	$2.16 * 10^{-2}$	$4.6 * 10^{-4}$	$3.21 * 10^{-2}$	$1.03 * 10^{-3}$
case 2	$2.51 * 10^{-2}$	$6.3 * 10^{-4}$	$3.83 * 10^{-2}$	$1.46 * 10^{-3}$
case 3	$2.12 * 10^{-2}$	$4.5 * 10^{-4}$	$3.61 * 10^{-2}$	$1.31 * 10^{-3}$
case 4	$2.73 * 10^{-2}$	$7.5 * 10^{-4}$	$4.31 * 10^{-2}$	$1.85 * 10^{-3}$
case 5	$1.75 * 10^{-2}$	$3.1 * 10^{-4}$	$2.74 * 10^{-2}$	$0.75 * 10^{-3}$
case 6	$2.05 * 10^{-2}$	$4.2 * 10^{-4}$	$3.21 * 10^{-2}$	$1.03 * 10^{-3}$
case 7	$1.37 * 10^{-2}$	$1.9 * 10^{-4}$	$2.36 * 10^{-2}$	$0.56 * 10^{-3}$
case 8	$1.73 * 10^{-2}$	$2.9 * 10^{-4}$	$2.71 * 10^{-2}$	$0.73 * 10^{-3}$

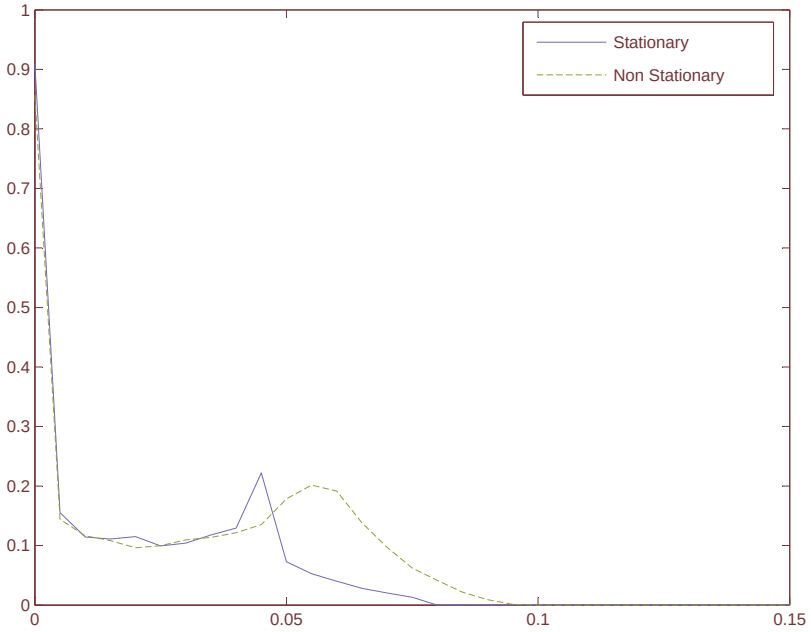

Fig. 52. Sample densities for the 3 sample example: case 1

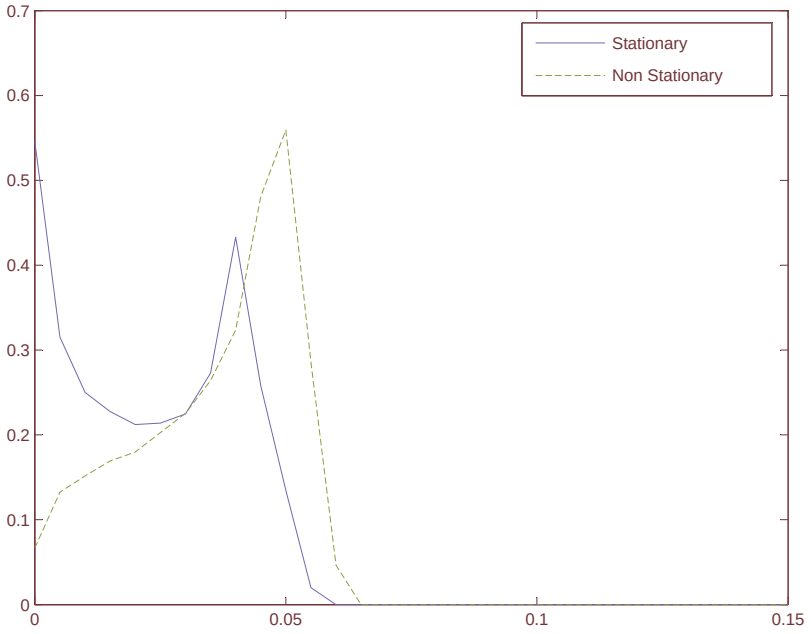

Fig. 53. Sample densities for the 3 sample example: case 2

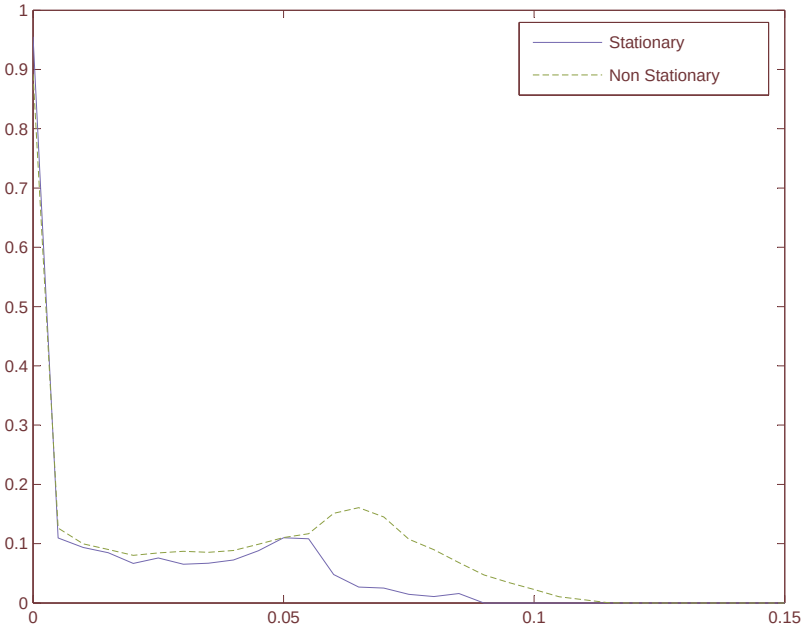

Fig. 54. Sample densities for the 3 sample example: case 3

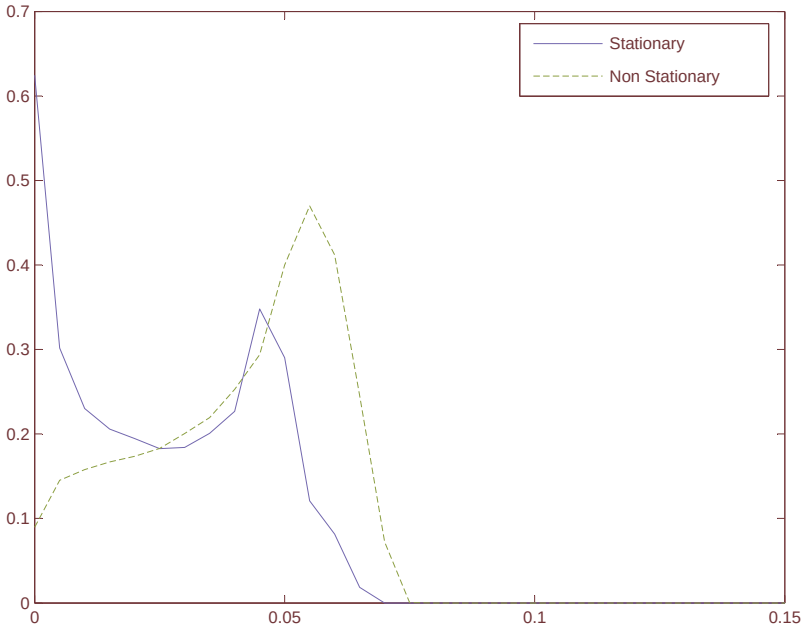

Fig. 55. Sample densities for the 3 sample example: case 4

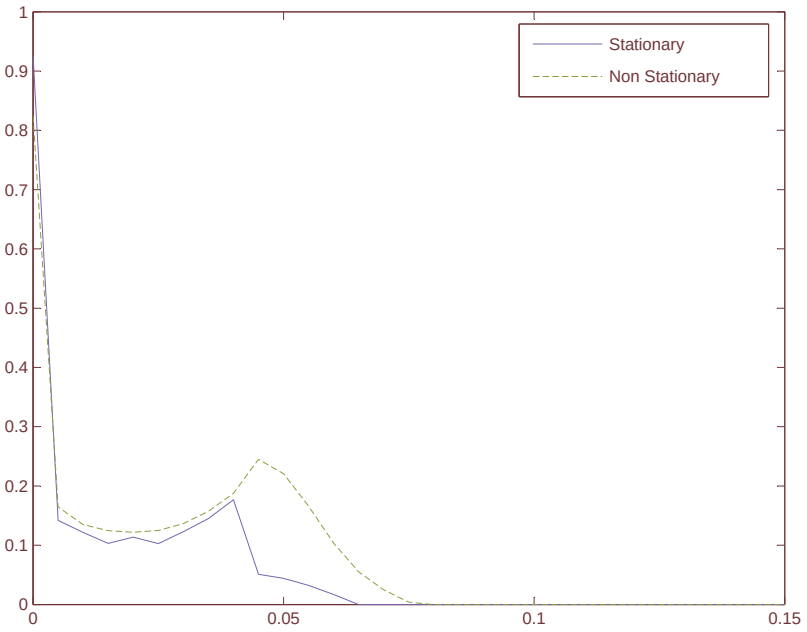

Fig. 56. Sample densities for the 3 sample example: case 5

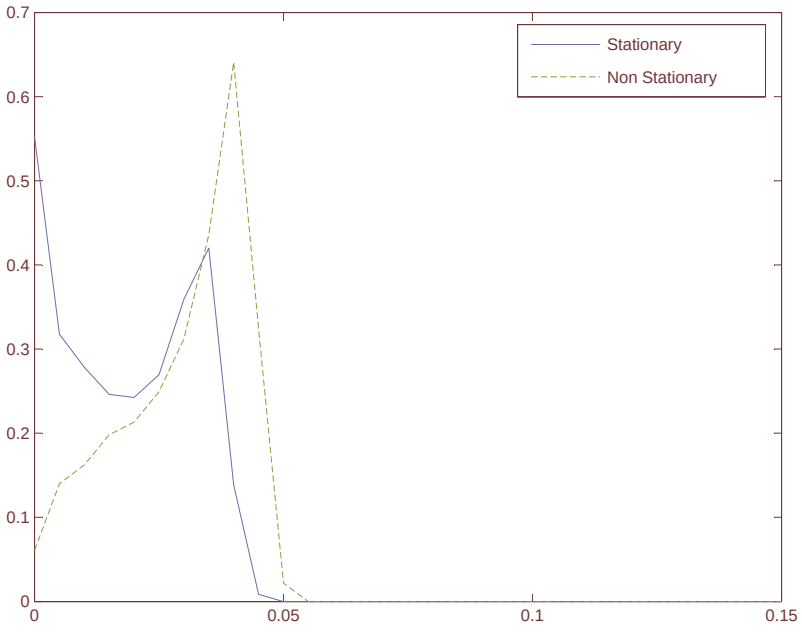

Fig. 57. Sample densities for the 3 sample example: case 6

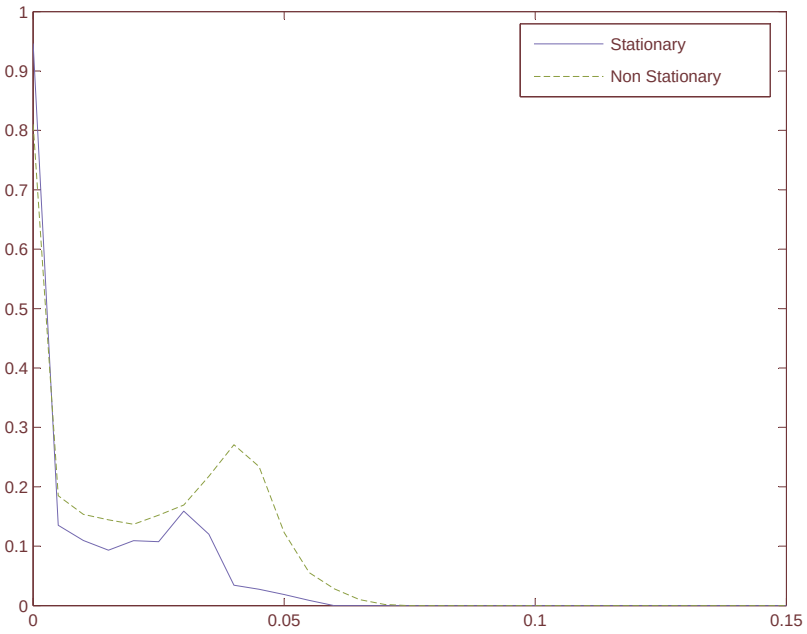

Fig. 58. Sample densities for the 3 sample example: case 7

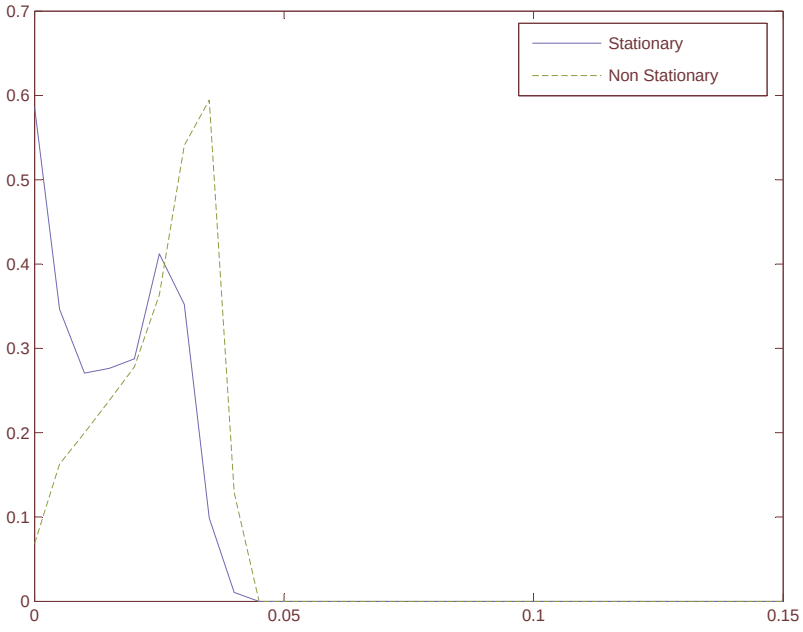

Fig. 59. Sample densities for the 3 sample example: case 8

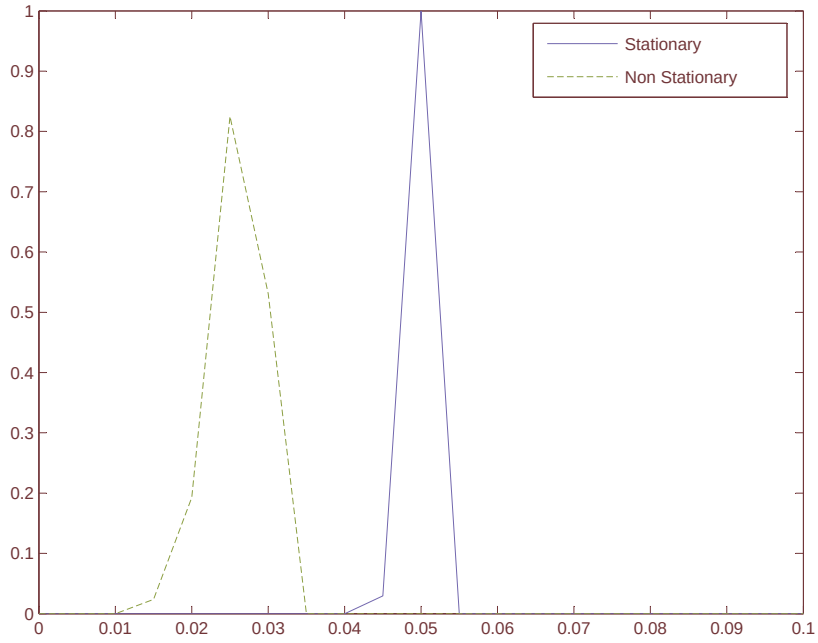

Fig. 60. Representation of the effect of a very small value of ϵ ($\epsilon = 10^{-3}$) on the sample density

matrix. The Hermitian matrix has special properties that could be used by one to verify that the sets of points are positive definite. One would only need to compute all the eigenvalues, knowing that if all the eigenvalues are positive, then the matrix is positive definite. Note that any not-positive definite sets of points are automatically ignored by the simulation.

1. Median, Mode And Confidence Bounds

While the distribution shapes such as Fig. 58 and 59 can be intriguing, it is not readily apparent how best to utilize them so as to address the fundamental question, "Is the stationary case more robust than the non-stationary?" This situation is complicated by the differing dimensionality that leads to the creation of the distributions. While there are admittedly many ways one might use the distributions to evoke an answer, we recommend one in particular. Since what we mean by robustness is related to

stability of "performance" (which is, for these examples, false alarm probability α), it might prove useful to the user to provide a bound on the change in performance as the covariance moves about the parameter surface. By making this surface very close to the nominal by controlling ϵ to be small, such a bound could be arbitrarily reduced. The comparison between stationary and non-stationary would thus translate into a comparison of the relative size of ϵ required to constrain the change in α (i.e $\Delta\alpha$) to be no more than a certain amount (i.e. 10% of the design goal of α=0.05). Since for both the non-stationary and stationary case the corresponding parameter surface is just a model of a $\sqrt{\epsilon}$-ball in a metric space of matrices, the dimensionality issue is alleviated. To compute such a bound one simply employes a ninety percent confidence bound on the slope (calculated from the appropriate distribution) and then multiplies by the maximum distance one can travel between any two points on the parameter surface, which is a 2-D ellipsoid for the stationary case and a 5-D sphere for the non-stationary case. Since these distances (which even for the ellipsoid case, can be easily computed – see following section) involve ϵ, the result will lead to a maximal epsilon compatible with a specified $\Delta\alpha$ (in per cent) at ninety percent confidence. The introduction of the notions of median, mode and confidence bound will also help in the making of the decision on the robustness of the detection scheme.

In statistics, the *mode* is the value that has the largest number of observations, namely the most frequent value or values. The mode is not necessarily unique, unlike the arithmetic mean [44]. In probability theory and statistics, the median is a number that separates the highest half of a sample, a population, or a probability distribution from the lowest half. More precisely half of the population will have values less than or equal to the median and half of the population will have values equal to or greater than the median [44].

A confidence bound is, for example, if X is a 90 percent upper one-sided bound,

Table IX. Means, variances, medians, standard deviation, mode and 90 and 75 percentile for each case scenario for the stationary case

Case	Mean	Variance	Median	Mode	90%	75%
case 1	$2.16 * 10^{-2}$	$4.66 * 10^{-4}$	0.01	0.0025	0.045	0.04
case 2	$2.51 * 10^{-2}$	$6.28 * 10^{-4}$	0.025	0.0025	0.045	0.04
case 3	$2.13 * 10^{-2}$	$4.53 * 10^{-4}$	0.005	0.0025	0.055	0.04
case 4	$2.74 * 10^{-2}$	$7.50 * 10^{-4}$	0.025	0.0025	0.05	0.045
case 5	$1.75 * 10^{-2}$	$3.06 * 10^{-4}$	0.005	0.0025	0.04	0.03
case 6	$2.05 * 10^{-2}$	$4.12 * 10^{-4}$	0.02	0.0025	0.035	0.03
case 7	$1.37 * 10^{-2}$	$1.89 * 10^{-4}$	0.005	0.0025	0.035	0.025
case 8	$1.73 * 10^{-2}$	$2.99 * 10^{-4}$	0.015	0.0025	0.03	0.025

this would imply that ninety percent of the population is less than X. If X is a ninety percent lower one-sided bound, this would indicate that ninety percent of the population is greater than X. For the following results, we used a ninety percent confidence bound that is an upper one sided bound (ninety percent of the area under the curve that is represented by the sample density is on the left of that limit) [45].

The median, mode, and ninety percent confidence bound are recalculated for all the previous simulations. A summary of the results are presented in tables IX and X.

Table X. Means, variances, medians, standard deviation, mode and 90 and 75 percentile for each case scenario for the non-stationary case

Case	Mean	Variance	Median	Mode	90%	75%
case 1	$3.21 * 10^{-2}$	$1.03 * 10^{-3}$	0.03	0.0025	0.065	0.055
case 2	$3.83 * 10^{-2}$	$1.46 * 10^{-3}$	0.04	0.0525	0.055	0.05
case 3	$3.62 * 10^{-2}$	$1.31 * 10^{-3}$	0.03	0.0025	0.075	0.06
case 4	$4.31 * 10^{-2}$	$1.85 * 10^{-3}$	0.045	0.0575	0.060	0.055
case 5	$2.74 * 10^{-2}$	$7.529 * 10^{-4}$	0.025	0.0025	0.055	0.045
case 6	$3.21 * 10^{-2}$	$1.03 * 10^{-3}$	0.035	0.0425	0.045	0.040
case 7	$2.36 * 10^{-2}$	$5.584 * 10^{-4}$	0.02	0.0025	0.045	0.040
case 8	$2.71 * 10^{-2}$	$7.3174 * 10^{-4}$	0.025	0.0375	0.035	0.035

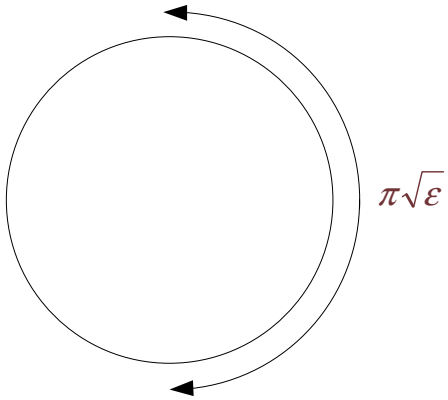

Fig. 61. Maximum distance between two points for the stationary case

2. Total Amount of Change in α

The total amount of change in α is the measure that will make a scheme robust or not. For both approaches, the stationary and non-stationary scheme, it is the maximum distance between two perturbations multiplied by the ninety percent confidence bound. It is the ability of one to surely state, with ninety percent confidence, that the scheme is robust or not. In order to do so, the total amount of change in α needs to be less than ten percent. Therefore we should have:

$$\Delta\alpha \leq 0.005 \tag{4.71}$$

a. Maximum Distance for the Stationary Case

The maximum distance between two perturbations (sphere in 3D) for the stationary approach is equal to $\pi\sqrt{\epsilon}$ and is represented in figure 61. Hence the total amount of change in α is then equal to:

$$\Delta\alpha = \pi\sqrt{\epsilon}(90\%) \tag{4.72}$$

b. Maximum Distance for the Non-Stationary Case

The maximum distance between two perturbations (ellipsoid of non circular radius) is a little more difficult to compute. We first need to compute the maximum distance between two points in a ellipse of equation $f^2 + 2d^2 = \epsilon$ (see figure 62).

$$Max\ distance = \int_{-\sqrt{\frac{\epsilon}{2}}}^{\sqrt{\frac{\epsilon}{2}}} \sqrt{1 + (\frac{\partial f}{\partial d})^2} dd \tag{4.73}$$

This integral is called an *elliptic integral* and can be numerically computed [46]. In integral calculus, elliptic integrals originally arose in connection with the problem of giving the arc length of an ellipse and were first studied by Fagnano and Leonhard Euler. In the modern definition, an elliptic integral is any function f which can be expressed in the form:

$$f(x) = \int_c^x R(t, P(t)) dt \tag{4.74}$$

where R is a rational function of its two arguments, P is the square root of a polynomial of degree three or four (a cubic or quartic) with no repeated roots, and c is a constant. In general, elliptic integrals cannot be expressed in terms of elementary functions. Exceptions to this are when P has repeated roots, or when R(x,y) contains no odd powers of y. However, with appropriate reduction formula, every elliptic integral can be brought into a form that involves integrals over rational functions, and the three canonical forms (i.e. the elliptic integrals of the first, second and third kind).

The partial derivative of f with respect to d is equal to:

$$\frac{\partial f}{\partial d} = \frac{-2d}{\sqrt{\epsilon - 2d^2}} \tag{4.75}$$

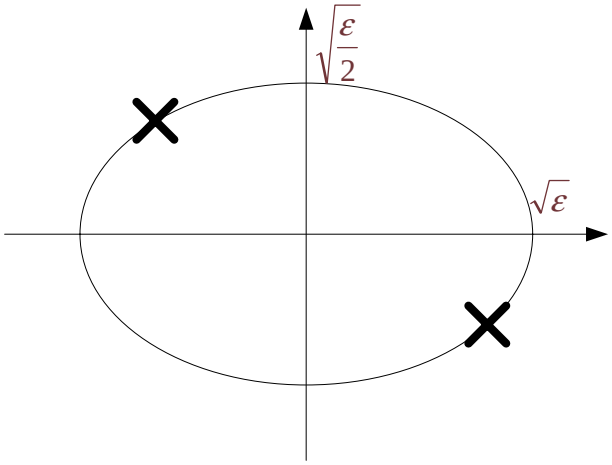

Fig. 62. Maximum distance between two points for the non-stationary case

Hence, the total amount of change in α is equal to:

$$\begin{aligned}\Delta\alpha &= 2\int_0^{\sqrt{\frac{\epsilon}{2}}}\sqrt{1+(\frac{\partial f}{\partial d})^2}dd(90\%) \\ &= 2\int_0^{\sqrt{\frac{\epsilon}{2}}}\sqrt{1+(\frac{-2d}{\sqrt{\epsilon-2d^2}})^2}dd(90\%) \leq 0.005 \end{aligned} \tag{4.76}$$

One might want to verify his or her results using an approximation (using the geometric mean and a circle instead of an ellipsoid for the shape). The approximation can be found to be equal to $0.84\pi\sqrt{\epsilon}$.

3. The Effect of α, Signal Vector and Correlation

From the previous results, it is easy to realize that the smaller the value of ϵ is, the less amount of change in α that will be available. This rises a question: What would be the maximum value of ϵ that could guaranty an amount of change in α smaller than ten percent? In other words: How much confidence can we have in α and still be sure that the detection scheme will be robust?

Table XI. Maximum value of α that would guarantee a maximum total amount of change of less than 10%

ϵ_{2D}	ϵ_{5D}
$1.00 * 10^{-3}$	$0.95 * 10^{-3}$

It is obvious that the robustness problem boils down to the knowledge of α. The amount of confidence put into the selection of α will very much affects the final decision. The upper limits for α that would guarantee a maximum of ten percent in the total amount of change in α is summarized in table XI.

The next important issue is the effect of the signal vector and the type of correlation used (positive or negative) on the results. The results are summarized in tables XII and XIII and are represented in figures 63 and 64. From all the simulations run (figures 65 to 70), it appears that the combination of positive correlation and strong signal vector generates the highest $\Delta\alpha$. Also, the combination of a negative correlation and a weak signal generates the weakest $\Delta\alpha$.

Note that if one selects a slightly higher value of ϵ_{5D} the results will show three of the four values of $\Delta\alpha$ bigger than 0.005. In order to have all four values of $\Delta\alpha$ bigger than 0.005, one needs to set ϵ_{5D} equals to a minimum of $3.80 * 10^{-3}$ and to set ϵ_{2D} equals to a minimum of $1.27 * 10^{-4}$.

Note that if we compare table XIV to table XV for each corresponding case, the ϵ value for table XIV is always greater than or equal to that for table XV. Thus, less deviation from the nominal is allowed for the non-stationary case and it can therefore be judged to be less robust (although the differences are small). While the presence of non-stationary data compromises robustness, the amount of compromise is not

Table XII. The effect of signal and correlation on the value of $\Delta\alpha$ for the non-stationary case

case	ϵ_{5D}	$a_0 = b_0 = c_0$	$d_0 = e_0$	f_0	s_1	s_2	s_3	90%	$\Delta\alpha$
1	$1.35 * 10^{-3}$	1.5	$\frac{3}{4}$	$\frac{1}{2}$	2	1.5	1.75	0.050	$4.95 * 10^{-3}$
2	$1.35 * 10^{-3}$	1.5	-$\frac{3}{4}$	$\frac{1}{2}$	2	1.5	1.75	0.030	$2.97 * 10^{-3}$
3	$1.35 * 10^{-3}$	1.5	$\frac{3}{4}$	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{8}$	0.045	$4.455 * 10^{-3}$
4	$1.35 * 10^{-3}$	1.5	-$\frac{3}{4}$	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{8}$	0.040	$3.96 * 10^{-3}$

Table XIII. The effect of signal and correlation on the value of $\Delta\alpha$ for the stationary case

case	ϵ_{2D}	$a_0 = b_0 = c_0$	$d_0 = e_0$	f_0	s_1	s_2	s_3	90%	$\Delta\alpha$
1	$1.00 * 10^{-3}$	1.5	$\frac{3}{4}$	$\frac{1}{2}$	2	1.5	1.75	0.0525	$4.96 * 10^{-3}$
2	$1.00 * 10^{-3}$	1.5	-$\frac{3}{4}$	$\frac{1}{2}$	2	1.5	1.75	0.0325	$2.98 * 10^{-3}$
3	$1.00 * 10^{-3}$	1.5	$\frac{3}{4}$	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{8}$	0.0425	$3.97 * 10^{-3}$
4	$1.00 * 10^{-3}$	1.5	-$\frac{3}{4}$	$\frac{1}{2}$	1	$\frac{1}{4}$	$\frac{1}{8}$	0.0355	$3.47 * 10^{-3}$

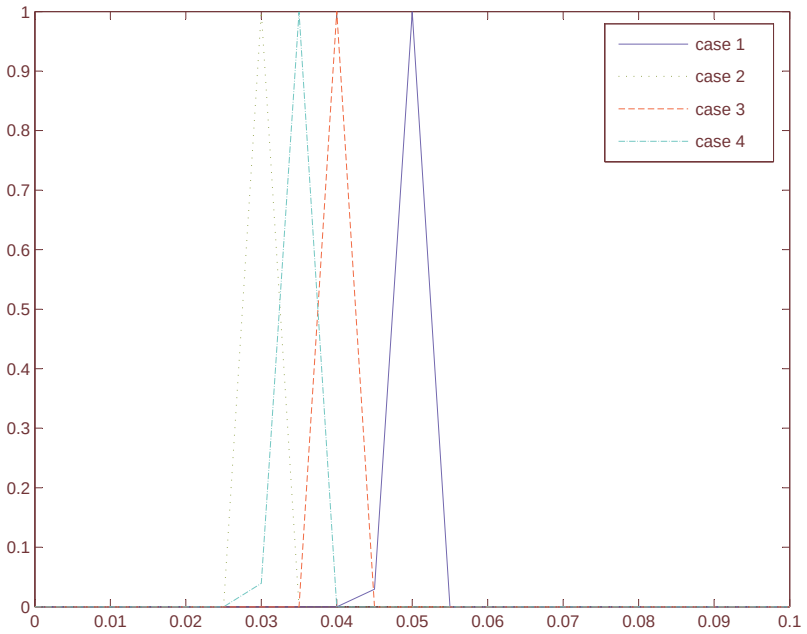

Fig. 63. Representation of the effect of the signal vector and the type of correlation for the stationary case

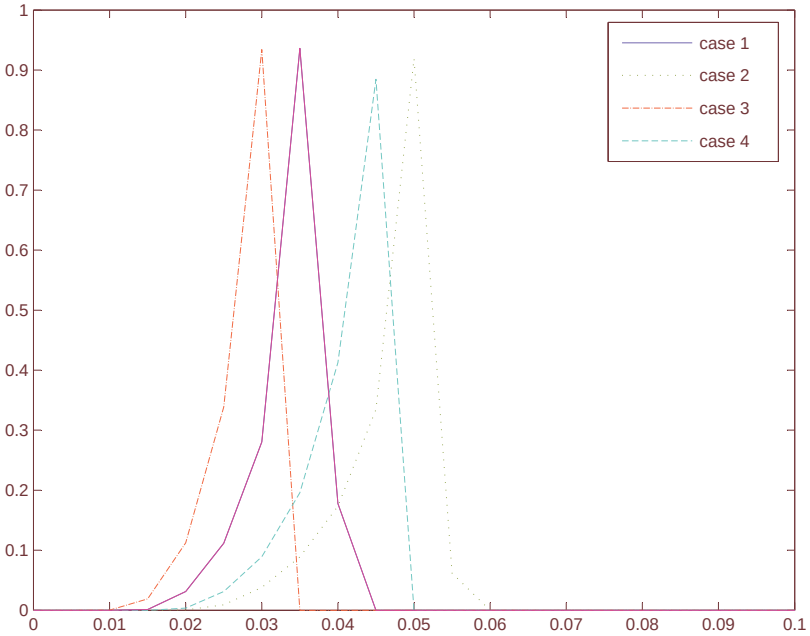

Fig. 64. Representation of the effect of the signal vector and the type of correlation for the non-stationary case

Table XIV. The values of ϵ_{2D} for the stationary case

case	$\Delta\alpha < 5\%$	$\Delta\alpha < 10\%$	$\Delta\alpha < 25\%$	$\Delta\alpha < 50\%$
1	$2.5 * 10^{-4}$	$1.0 * 10^{-3}$	$6.3 * 10^{-3}$	$2.25 * 10^{-2}$
2	$7.05 * 10^{-4}$	$2.85 * 10^{-3}$	$1.75 * 10^{-2}$	$6.85 * 10^{-2}$
3	$4.0 * 10^{-4}$	$1.5 * 10^{-3}$	$9.9 * 10^{-3}$	$3.9 * 10^{-2}$
4	$5.2 * 10^{-4}$	$2.2 * 10^{-3}$	$1.3 * 10^{-2}$	$5.2 * 10^{-2}$

Table XV. The values of ϵ_{5D} for the non-stationary case

case	$\Delta\alpha < 5\%$	$\Delta\alpha < 10\%$	$\Delta\alpha < 25\%$	$\Delta\alpha < 50\%$
1	$2.5 * 10^{-4}$	$8.5 * 10^{-4}$	$6.3 * 10^{-3}$	$2.05 * 10^{-2}$
2	$7.0 * 10^{-4}$	$2.8 * 10^{-3}$	$1.75 * 10^{-2}$	$4.98 * 10^{-2}$
3	$3.2 * 10^{-4}$	$1.25 * 10^{-3}$	$7.8 * 10^{-3}$	$2.55 * 10^{-2}$
4	$4.0 * 10^{-4}$	$1.6 * 10^{-3}$	$1.00 * 10^{-2}$	$4.0 * 10^{-2}$

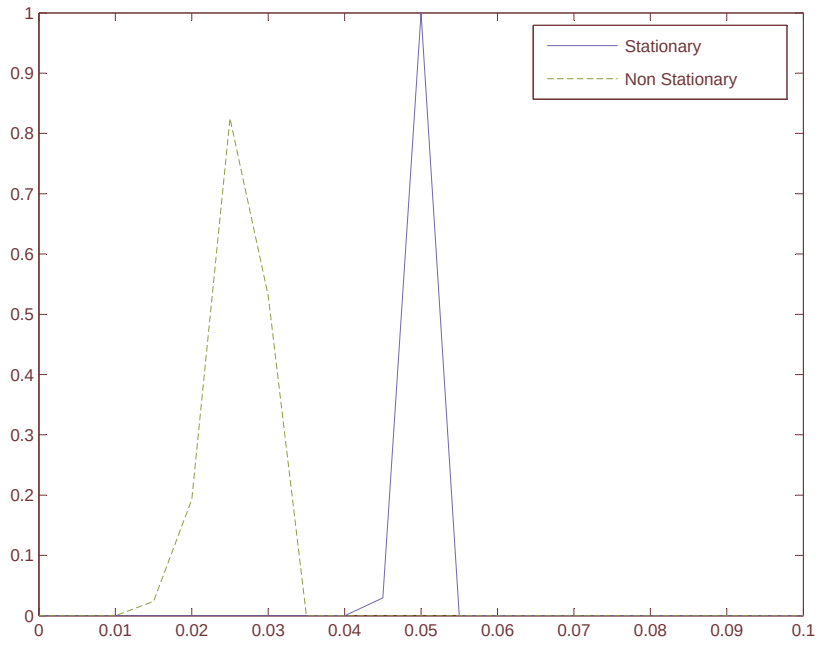

Fig. 65. Effect of ϵ on the location of the sample densities: case 1

large and may be considered acceptable - in view of the convenience in practice the assumption of stationarity offers.

G. Extension to N-Samples: Larger Sample Sizes

The aforementioned work has not only presented a versatile method for investigating robustness for a variety of applications, but has also addressed the specific question of the non-stationarity in the detection context. As might have been expected, the admission of non-stationarity data has been seen to compromise robustness, but as perhaps not expected, the degree of compromise is quite small in all cases considered. Another obvious question of interest concerns what happens to robustness as sample sizes increase. In many domains of statistical analysis, larger sample sizes make life easier and more convenient; one has laws of large numbers and central limit results. But one could also argue that the larger sample sizes imply larger covariance matrices,

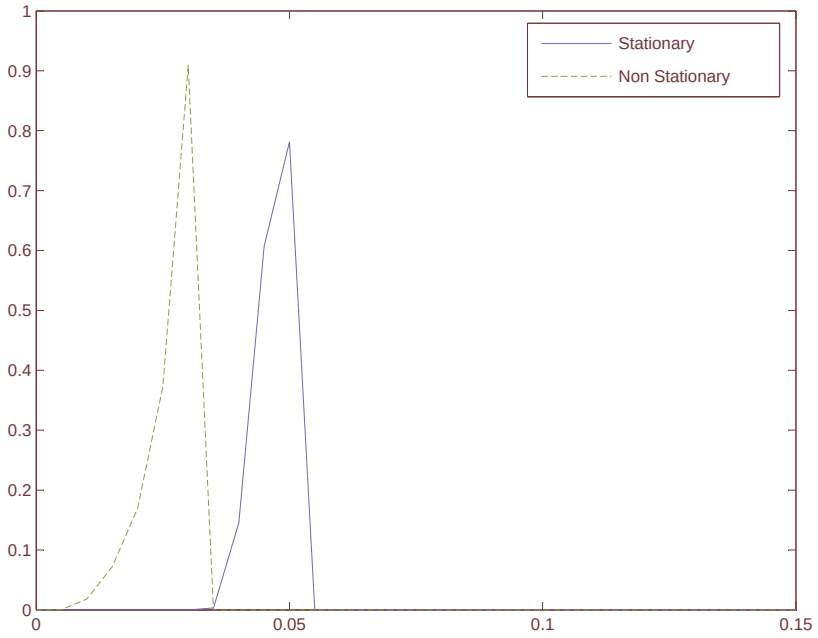

Fig. 66. Effect of ϵ on the location of the sample densities: case 2

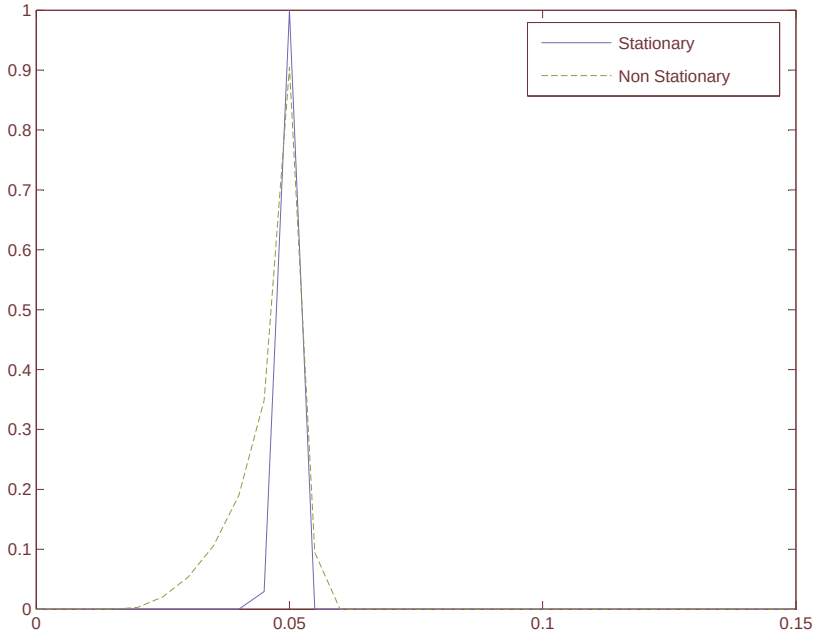

Fig. 67. Effect of ϵ on the location of the sample densities: case 3

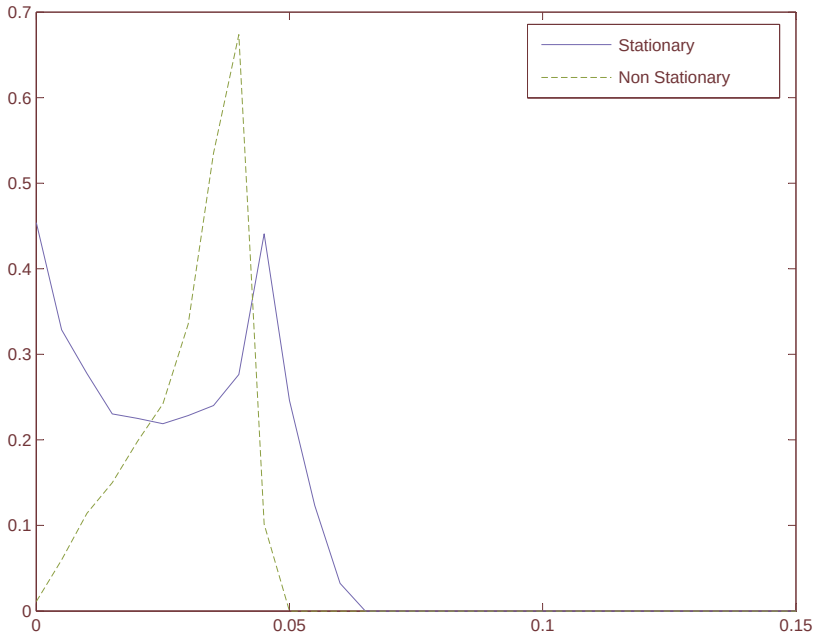

Fig. 68. Effect of ϵ on the location of the sample densities: case 4

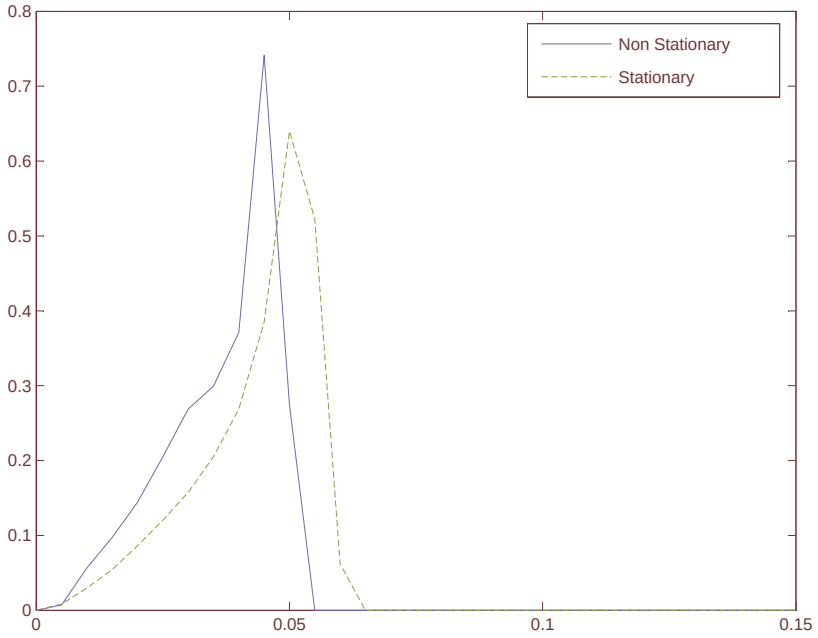

Fig. 69. Effect of ϵ on the location of the sample densities: case 5

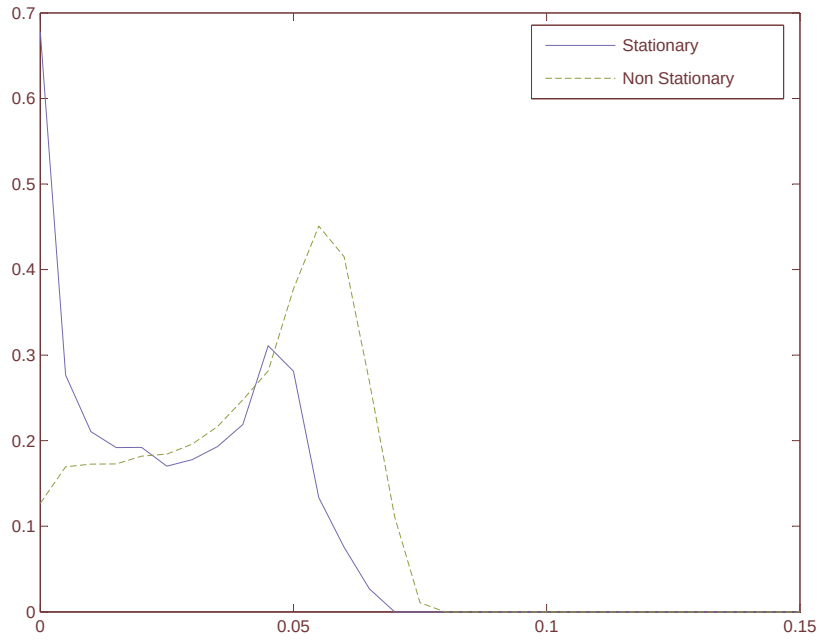

Fig. 70. Effect of ϵ on the location of the sample densities: case 6

with more opportunities for things to go wrong. In order to investigate the effect of sample size in robustness, we have considered an example which is not fully general so as to be tractable with many samples.

Consider the detection of signals in multi-variable Gaussian noise with an affine detector, with test statistic $\sum_{i=1}^{n} y_i$ compared to a threshold. The noise is assumed to be zero mean with covariance matrix of form:

$$C_{(a,c)} = \begin{pmatrix} a & c & & \cdots & & 0 \\ c & a & & & & \\ & & a & c & & \\ \vdots & & c & a & & \vdots \\ & & & & \ddots & \\ & & & & a & c \\ 0 & & & \cdots & c & a \end{pmatrix}$$

i.e., all zero off the diagonal blocks, and where the number of samples n are even. This type of matrix has been popular with various researchers and has been used quite recently in genomic work. Analogous work can be found in [47]. Note that this implies $\sigma_\Lambda^2 = E\{\sum_{ij} N_i N_j\} = na + nc$; and the parameter surface is given by:

$$\left\| C_{(a,c)} - C_{(a_0,c_0)} \right\| = n(a-a_0)^2 + n(c-c_0)^2 = \epsilon_n \tag{4.77}$$

i.e., $(a-a_0)^2 + (c-c_0)^2 = \epsilon_n/n$. This is a one dimensional manifold, in part, a circle of radius $\sqrt{\epsilon_n/n}$ centered on (a_0, c_0). Using "a" as the single coordinate, we have: $\sqrt{\nabla h G^{-1} \nabla h^T} = \sqrt{(\partial\alpha/\partial a) g_{11}^{-1} (\partial\alpha/\partial a)} = |\partial\alpha/\partial a| (g_{11})^{-1/2}$ where:

$$g_{11} = \langle \frac{\partial}{\partial a}, \frac{\partial}{\partial a} \rangle = 1 + (\frac{\partial c}{\partial a})^2 \tag{4.78}$$

In addition, for $\partial\alpha/\partial a$ we use equation 3.57 with $(\partial\sigma_\Lambda^2)/\partial a = n + n * \partial c/\partial a = n(1 + \partial c/\partial a)$ and since c is related to a via the circle, we have:

$$\begin{aligned} \frac{\partial c}{\partial a} &= \frac{\partial}{\partial a}(c_0 + \sqrt{\frac{\epsilon_n - n(a-a_0)^2}{n}}) \\ &= \frac{\partial}{\partial a}(c_0 + \sqrt{\epsilon_n/n - (a-a_0)^2}) \\ &= \frac{1}{2}(\epsilon_n/n - (a-a_0)^2)^{-1/2}(-2)(a-a_0) \\ &= \frac{\mp(a-a_0)}{\sqrt{\epsilon_n/n - (a-a_0)^2)}} \end{aligned} \tag{4.79}$$

To compute gradient distributions, point selection is readily obtained on the circle through regular subdivision of the polar angle, with unity weighting factor. Numerous examples were obtained, with all having he same shape as indicated in figure 71, with (a_0, c_0)=(1,1/2), ϵ=1/8 and threshold chosen for (nominally) $\alpha = 5{*}10^{-3}$. To compare the distribution shapes as sample size n varies, we matched the ϵ_n so that the "size" of the parameter manifold stays constant, i.e. ϵ_n/n is held constant (see Fig. 72). For convenience, this is done by setting $\epsilon_n = n\epsilon$, where ϵ is chosen small enough to

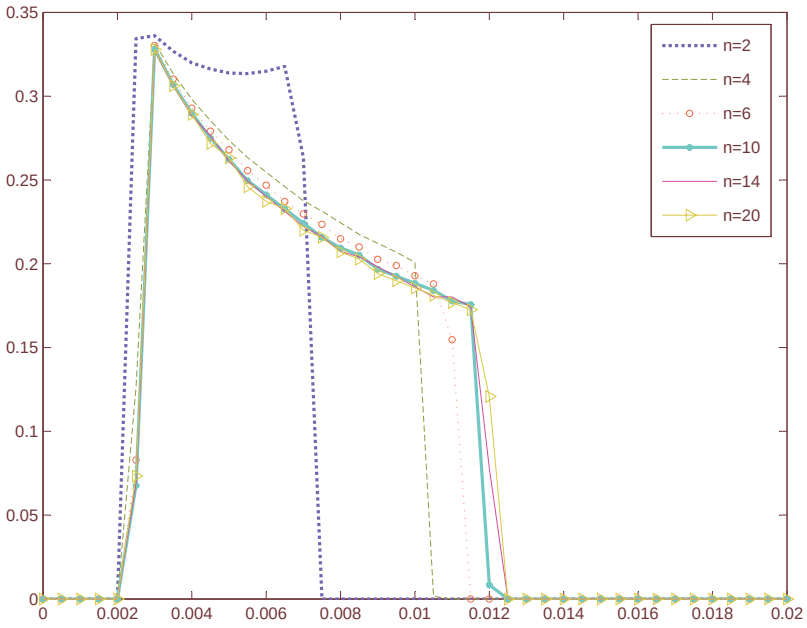

Fig. 71. Effect of an increase in sample size

guarantee that all points on the manifold satisfy the positive definite requirement. As one can see, gradient tends to increase with increasing sample size; however the effect is not dramatic for $n \geq 14$. We thus can conclude that larger sample sizes do indeed compromise robustness, but this compromise asymptotes around n=14.

H. Conclusion

The chapter illustrates how, for example, gradient can be regarded as a random variable and an empirical distribution generated by means of a density histogram, allowing conclusions regarding robustness to be drawn from statistical metrics such as median an confidence bounds. In order for one to be able to surely state on the robustness of a scheme requires the computation of $\Delta\alpha$. This $\Delta\alpha$ is very dependent on the value of ϵ and is therefore directly related to the confidence one has in the covariance matrix. For example, the more confidence one has in the covariance matrix,

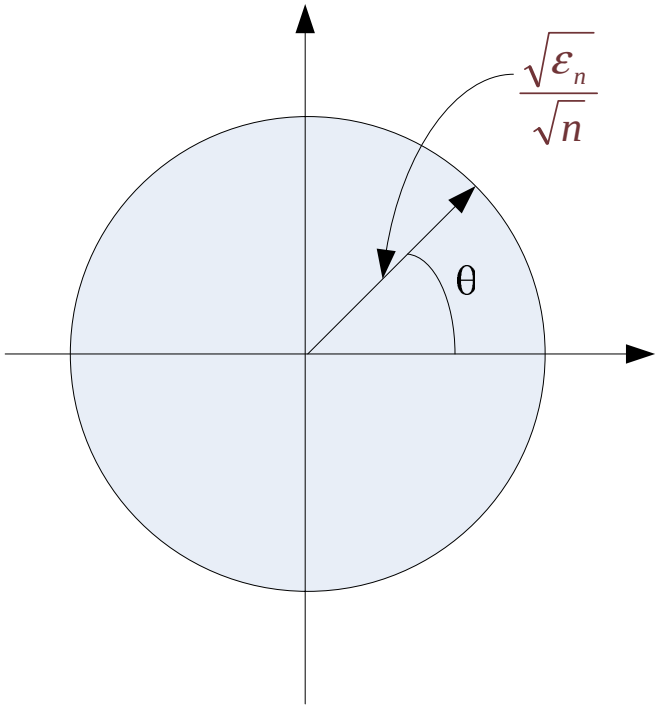

Fig. 72. Matching ϵ_n

the closer from the nominal values one will work, and the more robust the scheme might be.

This chapter also proves to go against all expected results. One legitimately could have been expecting the simulations would prove that the stationary scheme to be much more robust (due to fact that there is more constraint on the stationary approach than the non-stationary). The results show that if one compares stationary to non-stationary noise, robustness is reduced by admitting non-stationarity. The effect, however is not dramatic, and so little may be lost by assuming the convenient stationarity. This chapter shows the dependency of the measure of robustness on the signal vector, the sign of the correlation and ϵ. In addition, it shows that robustness decreases with larger sample sizes, but that saturation in this decrease occurs for sample sizes greater than 14.

CHAPTER V

A QUANTITATIVE ROBUSTNESS COMPARISON FOR SIGNAL DETECTION: NON-GAUSSIAN TAIL EFFECTS VERSUS RESIDUAL DEPENDENCY

In this chapter we investigate the effect of tail uncertainty on detector robustness through the use of a non-Euclidean manifold. This manifold is carefully chosen so as to simultaneously admit computation of detector robustness regarding tail effects while employing a model consistent with [48] which computes robustness regarding residual dependency. The goals of this paper thus include:

- Develop a non-Euclidean model which addresses tail effects and is consistent with the models of [48] for residual dependency.
- Compute results by example for robustness reflecting tail effects in a form which can be compared to the method of [48].
- Draw conclusions regarding the relative impact of tail effects vis a vis residual dependency on detector performance.

We are now poised to investigate the comparison of tail effects of a noise density on detector robustness with the impact of residual dependency as considered in [48]. We begin by modeling non-Gaussian tails with the generalized Gaussian.

A. Extension to Generalized Gaussian

The generalized Gaussian distribution (GGD) is another distribution that is used to characterize the statistics of signals. The main advantage of this distribution is the possible tuning of one of its parameters. The generalized Gaussian distribution (see

[49, 50, 51]) is defined as follows:

$$p(x; v, r) = \frac{r}{2v\Gamma(1/r)} e^{-(|x|/v)^r} \tag{5.1}$$

where $\Gamma(.)$ is the Gamma function, i.e. $\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} dt$, for $x > 0$.

Here v models the width of the PDF peak (variance), while r is inversely proportional with the decreasing rate of the peak. Sometimes, v is referred as the scale parameter while r is called the shape parameter. The GGD model contains the Gaussian and Laplacian PDF's as special cases, using r=2 and r=1, respectively.

Working out all the coefficients within equation 5.1, one can obtain the following equation for the PDF of the random variable:

$$f_X(x) = \frac{r\Gamma(3/r)^{1/2}}{2u^{1/2}\Gamma(1/r)^{3/2}} e^{-\frac{|x|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}} \tag{5.2}$$

where u can be expressed in terms of v and r and controls the variance for a fixed r value.

B. Detector and Parameter Surface

In order for one to be able to compare the results obtained through simulations of the generalized Gaussian case and the previous approach, one will need to use the same type of detector. The previous approach focused on residual dependency for jointly Gaussian noise, and therefore in this chapter we will investigate tail perturbations on *nominally* Gaussian noise by means of the generalized Gaussian model. Accordingly, for each detector we employ the matched filter within decision region as indicated in figure 73 for two samples, where the zero mean noise has variance a and b respectively, s_i represent the signal, and T the detector threshold.

Hence, the detector used is a linear detector (similar to a matched filter detector,

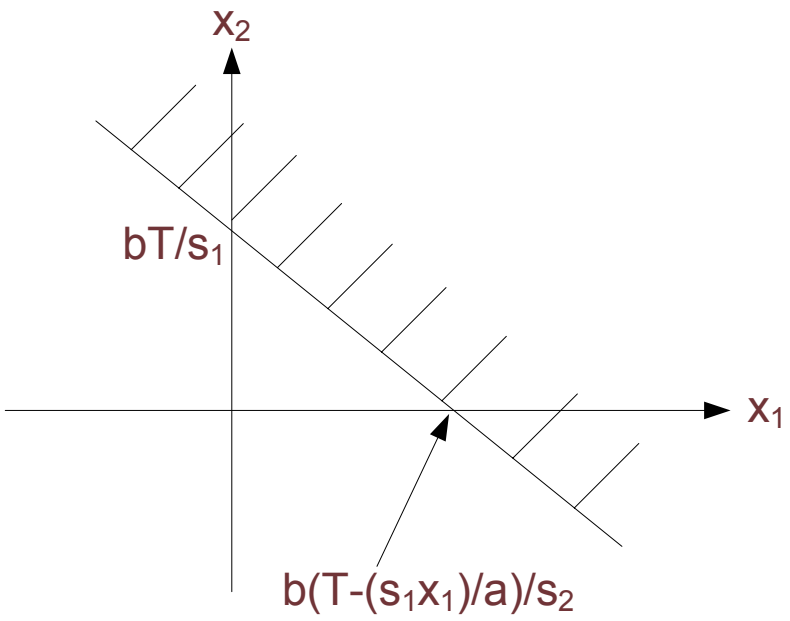

Fig. 73. Linear detector used for the generalized Gaussian case

see figure 73) with zero mean and variance σ_λ^2.The main difference with the two detectors is that the correlation coefficient is now equal to zero ($\rho = 0$), which is also equivalent to have $\rho = c/\sqrt{ab} = 0$, i.e. c=0.

While the detector is the same for both methods, the parameter surfaces for the previous approach and for the new one cannot be made the same because the previous approach admits dependency while forbidding the non-Gaussian, whereas the new tail-effect analysis features the reverse. What is important is that the two regions should be modeled in a comparable manner. For [48] with two samples, the parameter manifold is two-dimensional and employs coordinates a and b for the surface corresponding to:

$$\left\| \begin{pmatrix} a & c \\ c & b \end{pmatrix} - \begin{pmatrix} a_0 & c_0 \\ c_0 & b_0 \end{pmatrix} \right\| = \sqrt{\epsilon} \tag{5.3}$$

where the "0" subscript denotes nominal values and where the norm is Frobenius, leading to the surface

$$(a - a_0)^2 + (b - b_0)^2 + 2(c - c_0)^2 = \epsilon \tag{5.4}$$

Since this paper seeks to isolate the role of tail effects by assuming independence, when applying the previous method the above manifold should be defined with $c_0 = 0$, i.e., nominally independent, yielding:

$$(a - a_0)^2 + (b - b_0)^2 + 2c = \epsilon \tag{5.5}$$

To model tail effects assuming independence with generalized Gaussian noise, we employ the surface which is the Cartesian cross product of the reals (i.e. $r \in \mathcal{R}$) with the one-dimensional manifold:

$$\left\| \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} - \begin{pmatrix} a_0 & 0 \\ 0 & b_0 \end{pmatrix} \right\| = \sqrt{\epsilon}, \tag{5.6}$$

a circle $C(a_0, b_0; \sqrt{\epsilon})$ centered at (a_0, b_0) with radius $\sqrt{\epsilon}$. The resultant two-dimensional surface is the cylinder:

$$C(a_0, b_0; \sqrt{\epsilon}) \times \{r : r \in \mathcal{R}\} \tag{5.7}$$

which can admit coordinates a and r. Nominal values a_0 and b_0 would be chosen the same as for (5.5) but for proper comparison we take $r_0 = 2$ to reflect nominal Gaussianity.

Now that the parameter surfaces are specified, we wish to compute gradient for the detector false alarm rate α (the computation for detection probability β would be analogous). In [23] we show that for the "unbiased" method, gradient (in the

maximum direction) is given by:

$$(D_{\overrightarrow{X}}h)\Big|_{Extreme} = \sqrt{\nabla h G^{-1} \nabla h^T} \tag{5.8}$$

where h is the performance function (e.g. α or β),

$$\nabla h = (\frac{\partial h}{\partial x_1} \quad \frac{\partial h}{\partial x_2}) \tag{5.9}$$

where x_i are coordinates, and

$$G = \begin{pmatrix} g_{11} & g_{12} \\ g_{21} & g_{22} \end{pmatrix} \tag{5.10}$$

where the g_{ij} are the values of the Riemannian metric, which can be inherited for our purposes from the embedding of the surface in Euclidean space. For (5.7) this results in:

$$\begin{aligned} \frac{\partial}{\partial x_1} &= \frac{\partial}{\partial a} = (1, \frac{\partial b}{\partial a}, 0) \\ \frac{\partial}{\partial x_2} &= \frac{\partial}{\partial r} = (0, 0, 1) \end{aligned} \tag{5.11}$$

and hence:

$$\begin{aligned} g_{11} &= \langle \frac{\partial}{\partial a}, \frac{\partial}{\partial a} \rangle = 1 + (\frac{\partial b}{\partial a})^2 \\ g_{22} &= \langle \frac{\partial}{\partial r}, \frac{\partial}{\partial r} \rangle = 1 \\ g_{21} &= g_{12} = \langle \frac{\partial}{\partial a}, \frac{\partial}{\partial r} \rangle = 0 \end{aligned} \tag{5.12}$$

Using false alarm rate α as h, it is clear that the long calculation will be $\partial\alpha/\partial a$ and $\partial\alpha/\partial r$, i.e., ∇h.

C. Derivations: Partial with Respect to α and r

For this approach, the extended Gaussian approach, we will need to make few derivations. For example:

$$\frac{\partial \alpha}{\partial a} = \frac{\partial}{\partial a} \int_{-\infty}^{\infty} \int_{b(\frac{T-\frac{s_1 x_1}{a}}{s_2})}^{\infty} f_{X_1}(x_1) f_{X2}(x_2) dx_2 dx_1 \tag{5.13}$$

where:

$$f_{X_1}(x_1) = \frac{r\Gamma(3/r)^{1/2}}{2a^{1/2}\Gamma(1/r)^{3/2}} e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}$$

and

$$f_{X_2}(x_2) = \frac{r\Gamma(3/r)^{1/2}}{2b^{1/2}\Gamma(1/r)^{3/2}} e^{-\frac{|x_2|^r}{b^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}$$

Those derivations are computable through the use the Leibnitz integral rule, also known as the Leibnitz formula for the differentiation of definite integrals. The Leibnitz formula is used mainly when the boundaries of the integral are variables, and it is defined as follows [52, 53]:

$$\frac{d}{dt} \int_{a(t)}^{b(t)} f(t,x)\, dx = \int_{a(t)}^{b(t)} \frac{\partial f(t,x)}{\partial t}\, dx + f(t,b(t)) \frac{db(t)}{dt} - f(t,a(t)) \frac{da(t)}{dt} \tag{5.14}$$

Using $t(x_1) = b/s_2 * (T - s_1/a * x_1)$ as a simplification, one can easily obtain:

$$\begin{aligned}
\frac{\partial \alpha}{\partial a} &= \frac{\partial}{\partial a} \int_{-\infty}^{\infty} \int_{t(x_1)}^{\infty} f_{X_1}(x_1) f_{X2}(x_2) dx_2 dx_1 \\
&= \int_{-\infty}^{\infty} \frac{\partial}{\partial a} \int_{t(x_1)}^{\infty} f_{X_1}(x_1) f_{X2}(x_2) dx_2 dx_1 \\
&= \int_{-\infty}^{\infty} \frac{\partial}{\partial a} (f_{X_1}(x_1) \int_{t(x_1)}^{\infty} f_{X_2}(x_2) dx_2) dx_1 \\
&= \int_{-\infty}^{\infty} \Big[\frac{\partial f_{X_1}(x_1)}{\partial a} \int_{t(x_1)}^{\infty} f_{X2}(x_2) dx_2 + f_{X_1} \frac{\partial}{\partial a} \int_{t(x_1)}^{\infty} f_{X_2}(x_2) dx_2 \Big] dx_1 \\
&= \int_{-\infty}^{\infty} \Big[\frac{\partial f_{X_1}(x_1)}{\partial a} \int_{t(x_1)}^{\infty} f_{X_2}(x_2) dx_2 + f_{X_1}(x_1) \\
&* \quad \Big(- f_{X_2}(t(x_1)) \frac{\partial t(x_1)}{\partial a} + \int_{t(x_1)}^{\infty} \frac{\partial}{\partial a} f_{X_2}(x_2) dx_2 \Big) \Big] dx_1
\end{aligned} \tag{5.15}$$

Also, one can obtain:

$$\begin{aligned}
\frac{\partial}{\partial a} t(x_1) &= \frac{\partial}{\partial a} (\frac{b}{s_2} (T - \frac{s_1}{a} x_1)) \\
&= \frac{\frac{\partial b}{\partial a}}{s_2} (T - \frac{s_1}{a} x_1) + \frac{b s_1 x_1}{s_2 a^2}
\end{aligned} \tag{5.16}$$

Note:

$$\begin{aligned}
\frac{\partial b}{\partial a} &= \frac{\partial}{\partial a} (b \pm \sqrt{\epsilon - (a - a_0)^2}) \\
&= \mp (\epsilon - (a - a_0)^2)^{-(1/2)} (a - a_0)
\end{aligned} \tag{5.17}$$

Using Chain Rules, this time one will have:

$$\begin{aligned}\frac{\partial f_{X_2}(x_2)}{\partial a} &= \frac{\partial f_{X_1}(x_2,b)}{\partial a}.\frac{\partial b}{\partial a} \\ &= \frac{\partial}{\partial a}\frac{r\Gamma(3/r)^{1/2}}{2b^{1/2}\Gamma(1/r)^{3/2}}e^{-\frac{|x_2|^r}{b^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}.\frac{\partial b}{\partial a}\end{aligned} \tag{5.18}$$

Below are the details of the derivations:

$$\begin{aligned}\frac{\partial}{\partial a}(\frac{r\Gamma(3/r)^{1/2}}{2a^{1/2}\Gamma(1/r)^{3/2}}) &= \frac{r\Gamma(3/r)^{1/2}}{2\Gamma(1/r)^{3/2}}\frac{\partial}{\partial a}(\frac{1}{a^{1/2}}) \\ &= \frac{-r\Gamma(3/r)^{1/2}}{4\Gamma(1/r)^{3/2}a^{3/2}} = A\end{aligned} \tag{5.19}$$

$$\begin{aligned}\frac{\partial}{\partial a}(e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}) &= \frac{1}{2}\,\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}\frac{r}{a}\,e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}} \\ &= B\end{aligned} \tag{5.20}$$

Therefore, one finally obtains:

$$\frac{\partial f_{X_1}(x_1)}{\partial a} = A\,e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}} + B\frac{r\Gamma(3/r)^{1/2}}{2a^{1/2}\Gamma(1/r)^{3/2}} \tag{5.21}$$

The next logical step is to compute $\partial\alpha/\partial r$. This will need to be done in multiple steps. As a reminder we also have for the Gamma function:

$$\Gamma(x) = \int_0^\infty t^{x-1}e^{-t}dt \tag{5.22}$$

Therefore, we obtain, for the derivative of the Gamma function:

$$\Gamma'(x) = \int_0^\infty t^{x-1}e^{-t}ln(t)dt \tag{5.23}$$

For our equation we will need to compute:

$$\begin{aligned}\frac{\partial\Gamma(3/r)}{\partial r} &= \frac{\partial}{\partial r}\int_0^\infty t^{3/r-1}e^{-t}dt \\ &= \frac{-3}{r^2}\int_0^\infty t^{3/r-1}ln(t)e^{-t}dt \\ &= C \end{aligned} \tag{5.24}$$

Also,

$$\begin{aligned}\frac{\partial\Gamma(1/r)}{\partial r} &= \frac{\partial}{\partial r}\int_0^\infty t^{1/r-1}e^{-t}dt \\ &= \frac{-1}{r^2}\int_0^\infty t^{1/r-1}ln(t)e^{-t}dt \\ &= D \end{aligned} \tag{5.25}$$

Therefore, one will obtain for:

$$\begin{aligned}\frac{\partial}{\partial r}(\frac{r\Gamma(3/r)^{1/2}}{2a^{1/2}\Gamma(1/r)^{3/2}}) &= \frac{1}{2a^{1/2}}\frac{\partial}{\partial r}(\frac{r\Gamma(3/r)^{1/2}}{\Gamma(1/r)^{3/2}}) \\ &= \frac{1}{2a^{1/2}}\Big[\frac{1}{\Gamma(1/r)^{3/2}}(\Gamma(3/r)^{1/2}+\frac{r}{2}\ C\ \Gamma(3/r)^{-1/2}) \\ &+ \quad r\Gamma(3/r)^{\frac{1}{2}}(-\frac{3}{2}\Gamma(1/r)^{-5/2}\ D)\Big] = K \end{aligned} \tag{5.26}$$

and for $\partial/\partial r(e^{-|x_1|^r/(a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2})})$ one will have to use the following partial:

$$\frac{\partial}{\partial r}\Gamma(1/r)^{-r/2} = \frac{1}{2r}\Gamma(1/r)^{-r/2-1}\Gamma'(1/r) - \frac{1}{2}\Gamma(1/r)^{-r/2}ln(\Gamma(1/r)) = E \tag{5.27}$$

using integration per parts ($\int udv = uv - \int vdu$), and having $dv = ln(\Gamma(1/r))$, $u = -r/2$ (hence $v = 1/\Gamma(1/r) * \Gamma'(1/r) * -1/r$ and $du = -1/2dr$), one can obtain the previous equation.

Also, one can obtain:

$$\frac{\partial}{\partial r}\Gamma(3/r)^{r/2} = \frac{-3}{2r}\Gamma(3/r)^{r/2-1}\Gamma'(3/r) + \frac{1}{2}\Gamma(3/r)^{r/2} ln(\Gamma(3/r)) = F \tag{5.28}$$

again using integration per parts, and having $dv = ln(\Gamma(3/r))$, $u = -\frac{r}{2}$ (hence $v = \frac{1}{\Gamma(3/r)}\Gamma'(3/r)\frac{-3}{r}$ and $du = \frac{1}{2}dr$), one can obtain the previous equation.

We also need to compute:

$$\frac{\partial}{\partial r}(|x|^r) = |x|^r ln(|x|) = G \tag{5.29}$$

and finally:

$$\frac{\partial}{\partial r}(a^{-r/2}) = \frac{-1}{2}a^{-r/2} ln(a) = H \tag{5.30}$$

Knowing that $(e^U)' = U'e^U$, one will obtain for the final derivation for the exponential part:

$$\begin{aligned}
\frac{\partial}{\partial r}\frac{-|x|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}} &= \frac{\partial}{\partial r} - |x|^r a^{-r/2}\Gamma(1/r)^{-r/2}\Gamma(3/r)^{r/2} \\
&= -|x|^r a^{-r/2}\Gamma(1/r)^{-r/2}\ F\ - |x|^r a^{-r/2}\ E\ \Gamma(3/r)^{r/2} \\
&- \ |x|^r\ H\ \Gamma(1/r)^{-r/2}\Gamma(3/r)^{r/2} \\
&- \quad G\ a^{-r/2}\Gamma(1/r)^{-r/2}\Gamma(3/r)^{r/2} = I
\end{aligned} \tag{5.31}$$

Hence, one can easily compute:

$$\begin{aligned}
\frac{\partial}{\partial r} f_{X_1}(x_1) &= \frac{1}{2a^{1/2}}\frac{\partial}{\partial r}\Big(\frac{r\Gamma(3/r)^{1/2}}{\Gamma(1/r)^{3/2}} e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}\Big) \\
&= e^{-\frac{|x_1|^r}{a^{r/2}\Gamma(1/r)^{r/2}\Gamma(3/r)^{-r/2}}}\Big[K + \ I\ \frac{r\Gamma(3/r)^{1/2}}{\Gamma(1/r)^{3/2}}\frac{1}{2a^{1/2}}\Big] = J
\end{aligned} \tag{5.32}$$

Similar results will be obtained for $(\partial/\partial r * f_{X_2}(x_2) = L)$ by replacing all the a's by a

corresponding b. In a similar way, one will need to the following derivation:

$$\begin{aligned}
\frac{\partial \alpha}{\partial r} &= \frac{\partial}{\partial r} \int_{\frac{aT}{s_1}}^{\infty} \int_{b(\frac{T-\frac{s_1 x_1}{a}}{s_2})}^{\infty} f_{X_1}(x_1) f_{X2}(x_2) dx_2 dx_1 \\
&= \int_{\frac{aT}{s_1}}^{\infty} \int_{b(\frac{T-\frac{s_1 x_1}{a}}{s_2})}^{\infty} \Big[f_{X_1}(x_1) \frac{\partial f_{X_2}(x_2)}{\partial r} + f_{X_2}(x_2) \frac{\partial f_{X_1}(x_1)}{\partial r} \Big] dx_2 dx_1 \\
&= \int_{\frac{aT}{s_1}}^{\infty} \int_{b(\frac{T-\frac{s_1 x_1}{a}}{s_2})}^{\infty} \Big[f_{X_1}(x_1)\ L\ + f_{X_2}(x_2)\ J\ \Big] dx_2 dx_1 \qquad (5.33)
\end{aligned}$$

Computations were conducted over the sub-manifold of (5.7) given by the slice r=2 so as to admit comparison with the dependent Gaussian model corresponding to [48] (see (5.5)) for which nominal $c_0 = 0$, consistent with the independence assumption for extended Gaussian. Four cases were considered with nominal and signal values indicated in table XVI. Results are discussed in the next section.

Table XVI. All the different cases scenario for the extended Gaussian approach

case	a_0	b_0	c_0	s_1	s_2
1	1/2	1/4	0	1.5	2
2	1/2	1/4	0	1/2	1/4
3	1	1/2	0	1/2	1/4
4	1	1/2	0	2	1

D. Conclusion

We have described how to compute false alarm (or alternatively detection probability) gradient over two possible parameter manifolds. One assumes independence but

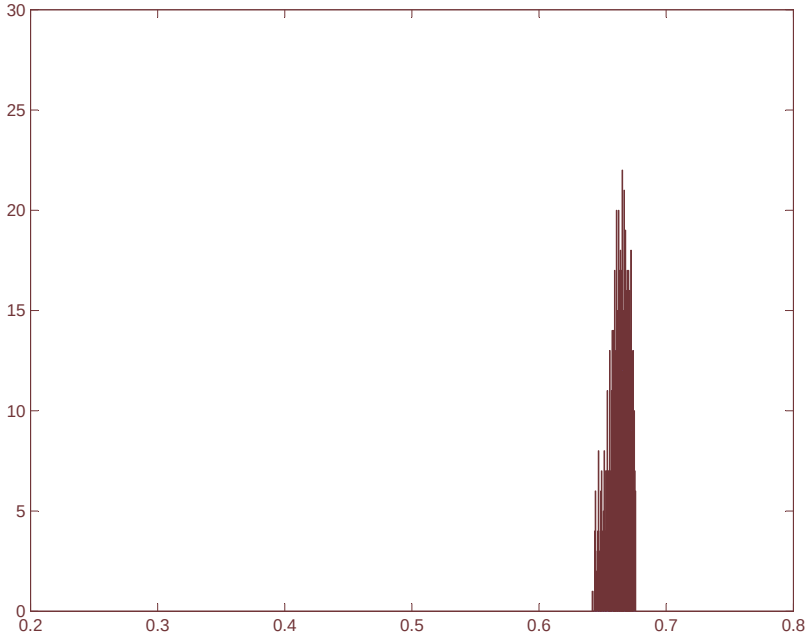

Fig. 74. Extended Gaussian approach (r = 2): case 1

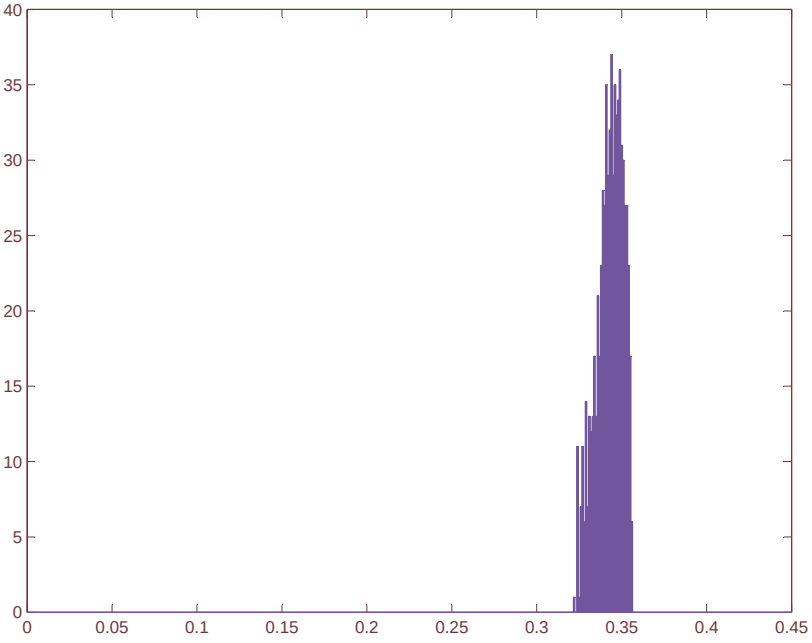

Fig. 75. Old method: case 1

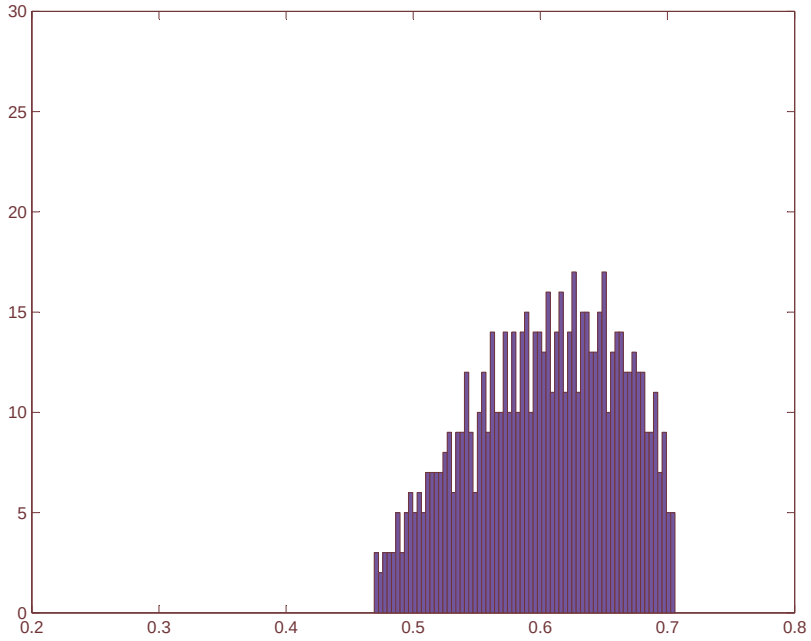

Fig. 76. Extended Gaussian approach (r = 2): case 2

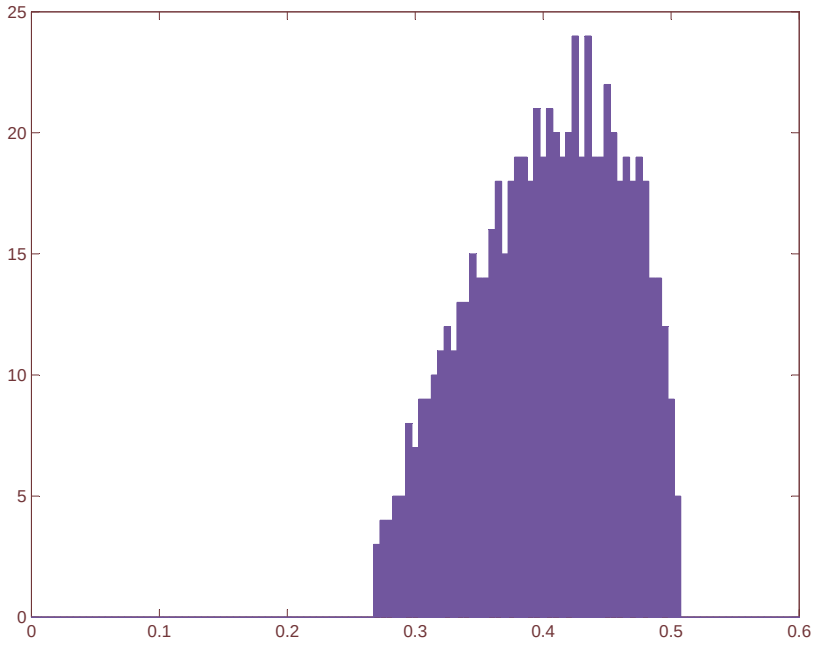

Fig. 77. Old method: case 2

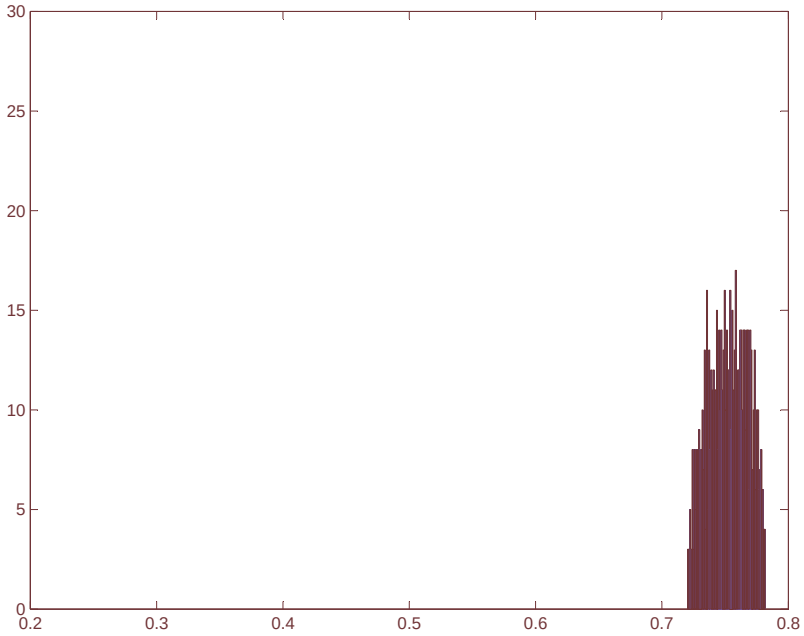

Fig. 78. Extended Gaussian approach (r = 2): case 3

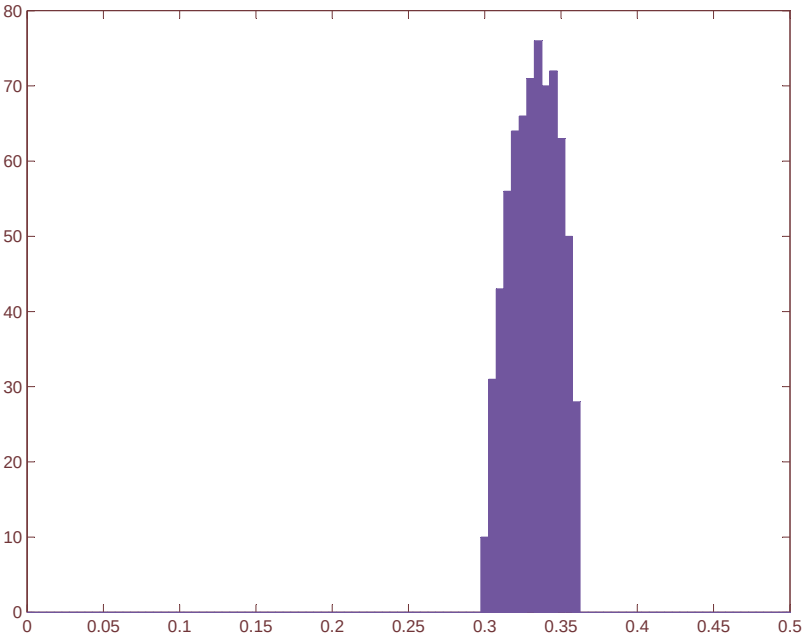

Fig. 79. Old method: case 3

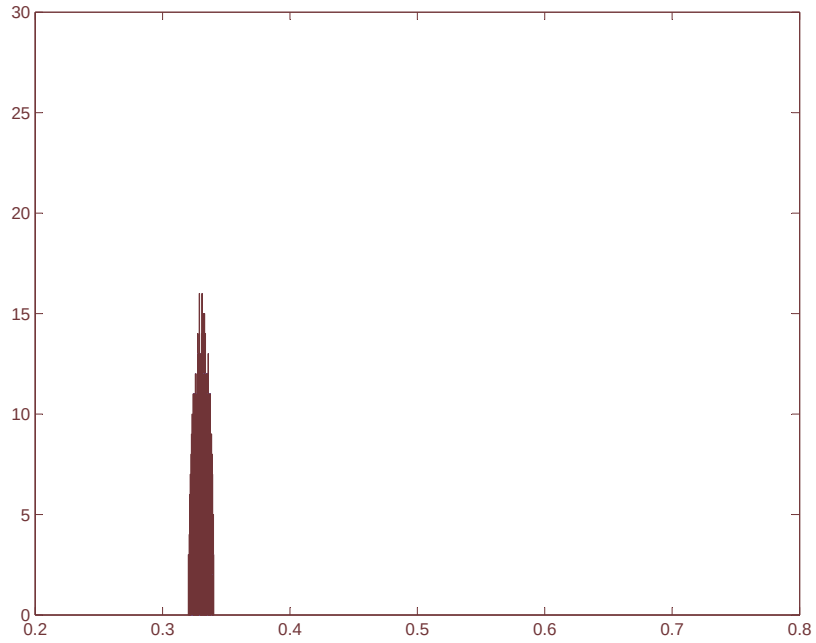

Fig. 80. Extended Gaussian approach (r = 2): case 4

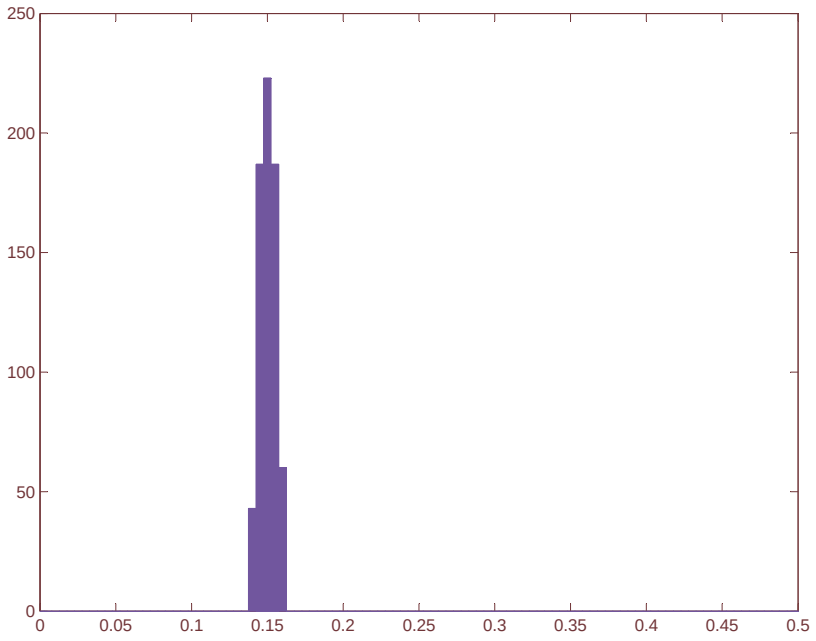

Fig. 81. Old method: case 4

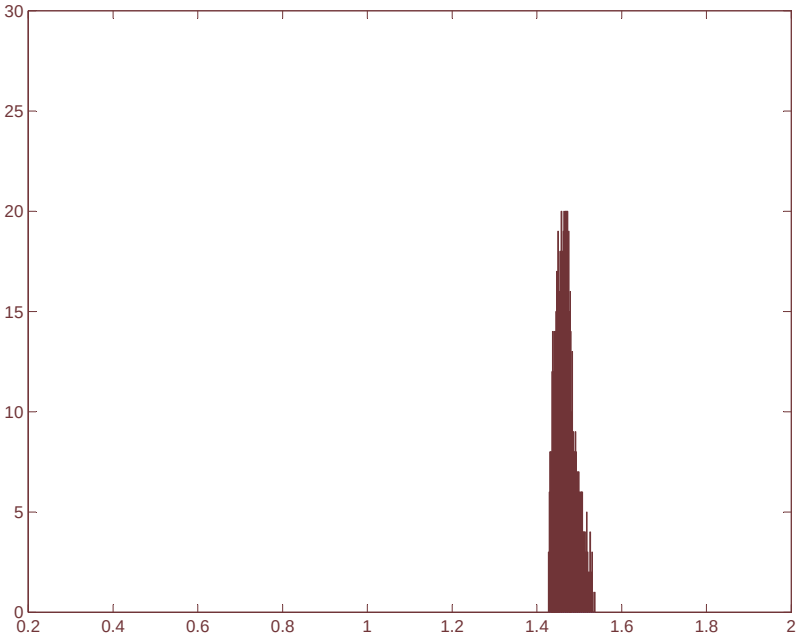

Fig. 82. Extended Gaussian approach (r = 1): case 1

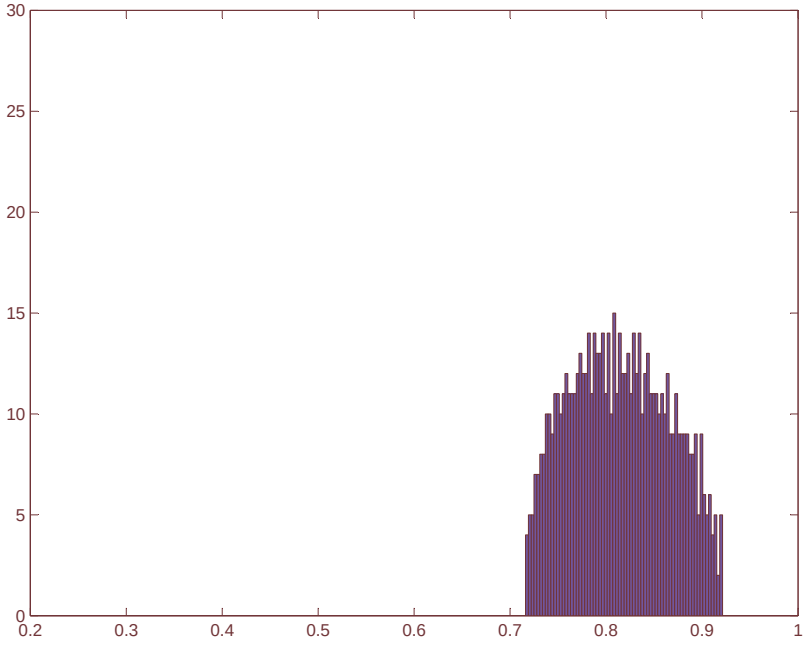

Fig. 83. Extended Gaussian approach (r = 1): case 2

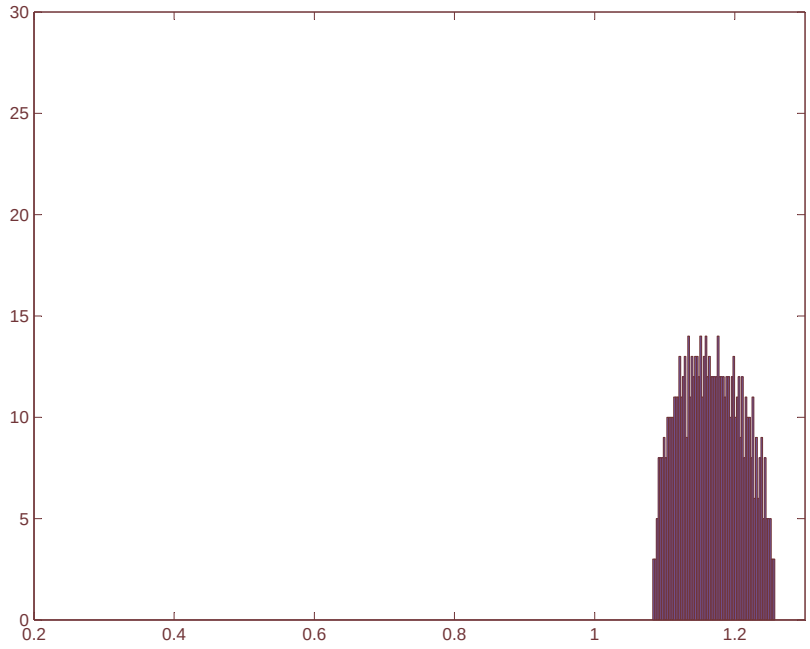

Fig. 84. Extended Gaussian approach (r = 1): case 3

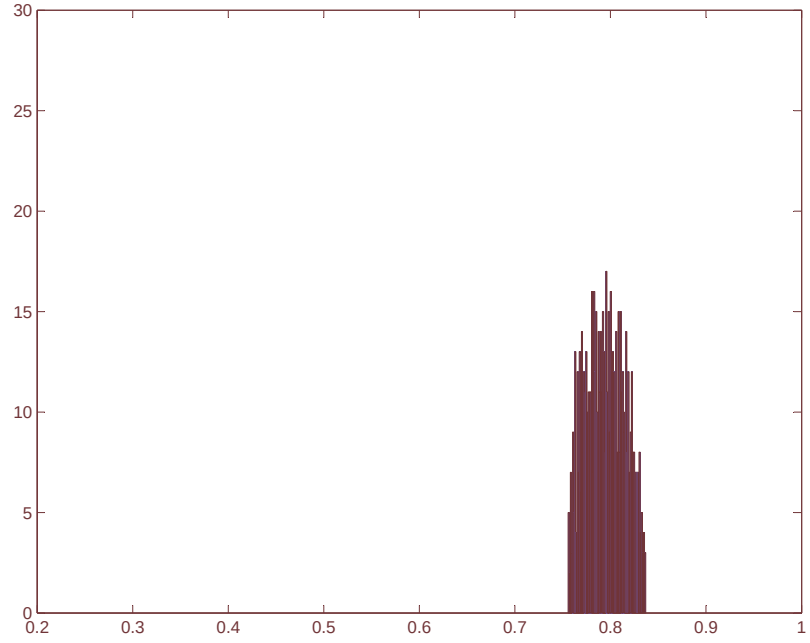

Fig. 85. Extended Gaussian approach (r = 1): case 4

Table XVII. Non Gaussian independent vs. dependent Gaussian

	$\Delta\alpha < 10\%$	
case	New method (r=2)	Old method
1	$\epsilon = 7.70 * 10^{-6}$	$\epsilon = 3.54 * 10^{-3}$
2	$\epsilon = 2.71 * 10^{-5}$	$\epsilon = 6.79 * 10^{-3}$
3	$\epsilon = 5.90 * 10^{-6}$	$\epsilon = 5.01 * 10^{-3}$
4	$\epsilon = 6.25 * 10^{-6}$	$\epsilon = 5.49 * 10^{-3}$

allows for non-Gaussian tail effects perturbed from nominally Gaussian by using an extended Gaussian model with r nominally equal to 2. The other, using a model from [48], admits dependency under Gaussian assumptions with nominal independence ($c_0 = 0$). For the cases of table XVI and for many points over the two parameter models, gradient was computed and sample densities obtained (figures 74-85). From these, comparisons can be made based on the $\epsilon - analysis$ described in the previous chapter. Recall that smaller epsilon reflects less robustness. Table XVII summarizes the results for residual dependency and non-Gaussian tail effects ("old method") and for residual tail effects and no dependency ("new method").

Note that for the new method, appropriate ϵ values are several orders of magnitude smaller, which is very revealing in how much more serious non-Gaussian tail effects degrade robustness compared to residual dependency under a Gaussian assumption. The new method also can of course transcend any kind of Gaussian assumption, even for the nominal. For example, with nominally Laplace noise ($r_0 = 1$) new method entries are shown in table XVIII.

Table XVIII. Results for the new method

	$\Delta\alpha < 10\%$
case	New method (r_0=1)
1	$\epsilon = 1.82 * 10^{-6}$
2	$\epsilon = 7.36 * 10^{-6}$
3	$\epsilon = 2.39 * 10^{-6}$
4	$\epsilon = 5.33 * 10^{-6}$

These ϵ values are even smaller indicating the difficulty that a linear detector experiences when the tail center of gravity moves from Gaussian to Laplace. Some example sample densities used to generate these results are illustrated through figures 74-85.

There remains the question of larger sample sizes. In our past work this has always been shown to lead to decreasing robustness as sample sizes increase. For example, in [48] an example is provided where gradient nearly doubles as n goes from 2 to 20. For the work of this dissertation it is expected that larger samples sizes will exacerbate the already serious impact of non-Gaussian tail effects.

CHAPTER VI

CONCLUSIONS: PUTTING IT ALL TOGETHER

A. Dissertation Summary

This dissertation is dedicated to the study of robustness measures for signal detection in non-stationary noise using differential geometric tools in conjunction with empirical distribution analysis. We first showed that gradient can be viewed as a random variable and therefore used to generate sample densities allowing one to conclude on robustness. We used the differential geometric methodology (through the use of non-Euclidean manifold) to the detection of time varying deterministic signals in imperfectly known dependent non-stationary Gaussian noise. The dissertation moves on to prove that robustness is barely reduced by admitting non-stationarity and also decreases with large sample sizes. The robustness converges in this decrease for sample sizes larger than 14.

The dissertation then investigated the effect of tail uncertainty on detector robustness and more precisely, investigated the comparison of tail effects of a noise density on detector robustness with the impact of residual dependency by modeling non-Gaussian tails with the generalized Gaussian formula. By comparing the two methods we showed how much more serious non-Gaussian tail effects degrade robustness compared to residual dependency under a Gaussian assumption. This approach also allowed us to transcend any kind of Gaussian assumption, even for the nominal (for example with nominally Laplace noise).

REFERENCES

[1] S. Kassam and H. Poor, "Robust techniques for signal processing: A survey," *Proc. IEEE*, vol. 73, pp. 433–480, 1985.

[2] P. Huber, "Robust statistical procedures," in *CBMS NSF Regional Conference Series in Applied Mathematics*, pp. 963–973, 1968.

[3] P. Huber, *Robust Statistics*, vol. 1. Hoboken NJ: Wiley, Inc., 1981.

[4] M. Thompson, D. Halverson, and G. Wise, "Robust detection in nominally laplace noise," *IEEE Transactions on Communications*, vol. 42, no. 3, pp. 1651–1660, February/March/April 1994.

[5] A. El-Sawy and V. V. Linde, "Robust detection of known signals," *IEEE Trans. Info. Theory*, vol. IT-23, pp. 722–727, November 1977.

[6] W. Liu, D. Halverson, S. Akkihal, and M. Thompson, "Local and nonlocal robustness measures with applications to distributed sensor systems," *IEEE Transactions on Aerospace and Electriconic System*, vol. 38, no. 2, pp. 675–681, 2002.

[7] W. Liu and D. Halverson, "Robustness of the sign detector in dependent noise," *J. Franklin Inst.*, vol. 336, no. 7, pp. 1155–1174, 1996.

[8] M. Bakich and D. Halverson, "Non-Euclidian robustness measures for communications and signal processing," *in Proc. of the Eleventh Annual International Conference on Signal Processing Applications and Technology*, pp. 567–601, October 16-19, 2000.

[9] V. V. Varma and D. Halverson, "Applications of unbiased perturbations towards providing robustness with pragmatic geometric methods," in *To appear in: Computational Statistics and Data Analysis*.

[10] D. Halverson, "Robust estimation and signal detection with dependent non-stationary data," *Circuits Systems and Signal Processing*, vol. 14, pp. 465–472, 1995.

[11] C. Tsai and D. Halverson, "Average robustness in signal detection and estimation," *J. Franklin Inst.*, vol. 333 B, pp. 127–139, 1996.

[12] F. Kellison and D. Halverson, "Applications of curvature toward the measurement of robustness for data processors," *Computational Statistics and Data Analysis*, vol. 11, no. 37, pp. 343–362, 2001.

[13] M. Thompson, D. Halverson, and Tsai, "Robust estimation of signal parameters with non stationary and/or dependent data," *IEEE Transactions on Information Theory*, vol. 39, pp. 617–623, 1993.

[14] M. Szewczul, D. Halverson, and M. Thompson, "Modelling and measuring detector performance/robustness in nominally laplace noise," *International J. of Modelling and Simulation*, vol. 24, no. 1, pp. 20–25, 2004.

[15] S. A. Kassam and H. V. Poor, "Robust techniques for signal processing: A survey," *Proc. IEEE*, vol. 73, pp. 433–481, March 1985.

[16] H. V. Poor, "Signal detection in the presence of weakly dependent noise - part ii: Robust detection," *IEEE Trans. Inform. Theory*, vol. 28, pp. 744–752, September 1982.

[17] G. V. Moustakides and J. B. Thomas, "Robust detection of signals in dependent noise," *IEEE Trans. Inform. Theory*, vol. 33, pp. 11–15, January 1987.

[18] E. C. Martin and H. V. Poor, "On the asymtotic efficiencies of robust detectors," *IEEE Trans. Inform. Theory*, vol. 38, pp. 50–60, January 1992.

[19] O. Kenny and L. White, "Robust detection of signal classes," *IEEE Trans. Inform. Theory*, vol. 34, no. 1, pp. 3131–3133, 1995.

[20] D. J. Warren and J. B. Thomas, "Asymptotically robust detection and estimation for very heavy-tailed noise," *IEEE Trans. Inform. Theory*, vol. 37, pp. 475–481, May 1991.

[21] M. Thompson and D. Halverson, "A differential geometric approach toward robust signal detection," *J. Franklin Inst.*, vol. 328, no. 4, pp. 379–401, 1991.

[22] M. Thompson and D. Halverson, "Geometric measures of robustness in signal processing." in *Advances in Communications and Signal Processing* (edited by W.A. Porter and S.C. Kakin), New York: Springer, 1989. pp. 345-348.

[23] M. Bakich, *Non Euclidean Robustness Measures for Communications and Signal Processing*. Ph.D. dissertation, Texas A&M University, 2002.

[24] V. Varma, "Applications of non-riemanian and differential information geometry in communications and signal processing algorithms with stationary and non-stationary data," M.S., Texas A&M University, 2004.

[25] K. Gauss, *Theory of Motion of Heavenly Bodies*. New York, Prentice Hall, 1963.

[26] S. Kay, *Fundamentals of Statistical Signal Processing: Estimation Theory*. Englewood Cliffs, NJ: Prentice Hall Signal Processing Series, 1998.

[27] S. Kay, *Fundamentals of Statistical Signal Processing: Detection Theory.* Englewood Cliffs, NJ: Prentice Hall Signal Processing Series, 1998.

[28] E. Lehman, *Testing Statistical Hypotheses.* New York, Springer, 1959.

[29] R. McDonough and A. Whalen, *Detection of Signals in Noise.* New York, Academic Press, 1995.

[30] H. V. Trees, *Detection, Estimation, and Modulation Theory*, vol. I-III. New York, Wiley-Interscience, 1968-1971.

[31] S. M. Kendall and A. Stuart, *The Advanced Theory of Statistics*, vol. 2. New York, Hafner Publishing Company, 1979.

[32] G. Box and G. Tiao, *Bayesian Interference in Statistical Analysis.* Reading, MA: Addison-Wesley, 1973.

[33] J. Rissanen, "Modeling by shortest data description," *Automatica*, vol. 14, pp. 465–471, 1978.

[34] E. Lehmann, *Testing Statistical Hypothesis.* New York, Wiley-Interscience, 1959.

[35] H. V. Trees, *Detection, Estimation, and Modulation Theory, Part I.* New York, Paperback, 1968.

[36] E. Lehman, *Testing Statistical Hypotheses.* New York, Hardcover, 1970.

[37] S. Chern and R. Osserman, Lebnitz Integrals, *Proceeding of Symposia in Pure Mathematics: Differential Geometry*, vol. XXVII. Austin, TX: Advanced Mathematic Symposium, 1975.

[38] S. Waner, Lecture Notes "Introduction to differential geometry and general relativity." Hofstra University, Online available.

http://people.hofstra.edu/faculty/StefanWaner/diffgeom/tc.html, April 19th 2005.

[39] I. Chavel, *Riemannian Geometry: A Modern Introduction*. Cambridge: Cambridge University Press 1994.

[40] V. Guillemin and A. Pollack, *Differential Topology*. Englewood Cliffs, NJ: Prentice-Hall, 1974.

[41] R. Millman and G. Parker, *Elements of Differential Geometry*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

[42] H. Reckziegel, *Mathematical Models from the Collections of Universities and Museums*. Munich, Germany: Braunschweig, 1986.

[43] I. Singer and J. Thorpe, *Lecture Notes on Elementary Topology and Geometry*. New York: Springer-Verlag, 1996.

[44] J. Kenney and E. Keeping, *Relation between Mean, Median, and Mode*, vol. 1, 3rd ed. Princeton, NJ: Mathematics of Statistics, 1962.

[45] J. Kenney and E. Keeping, *Confidence Interval Charts*, vol. 1. Princeton, NJ: Mathematics of Statistics, 1962.

[46] M. Abramowitz and I. A. Stegun, chap 17: Leibnitz Integrals, pp. 234-278 *Handbook of Mathematical Functions*. New York: Dover Publications, 1964.

[47] A. Jain and W. Waller, "On the optimal number of features in the classification of multivariate Gaussian data," *Pattern Recognition*, vol. 10, pp. 365–374, 1978.

[48] G. Raux and D. Halverson, "An empirical distribution approach to the application of differential geometric robustness analysis," *IEEE Trans. Inform. Theory*, submitted.

[49] S. Mallat, "A theory for multiresolution signal decomposition: the wavelet representation," *IEEE Trans. Inform. Theory*, vol. 11, pp. 674–693, July 1989.

[50] G. V. Wouwer, P. Scheunders, and D. V. Dyck, "Statistical texture characterization from discrete wavelet representations," *IEEE Trans. Signal Proc.*, vol. 8, pp. 592–598, April 1999.

[51] P. Moulin and J. Liu, "Analysis of multiresolution image denoising schemes using generalized gaussian and complexity priors," *IEEE Trans. Inform. Theory*, vol. 45, pp. 909–919, 1999.

[52] M. Abramowitz and I. Stegun, *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover, 1972.

[53] G. Borros and V. Moll, *Irresistible Integrals: Symbolics, Analysis and Experiments in the Evaluation of Integrals*. Cambridge: Cambridge University Press, 2004. 25-31.

APPENDIX A

LIST OF VARIABLES

L_p : error for the estimator

$\pm\delta$: variation in the entry of the covariance matrix

P : performance function

$f_X(x)$: PDF of the random variable x

x[.] : data set

θ : unknown parameter

$\widehat{\theta}$: estimator for detection theory

$p(x[n];\theta)$: probability density function of data x[n] parameterized by the unknown parameter θ

σ^2 : variance of the random variable

E_n : n-dimensional Euclidian space

y_i : coordinates in the n-dimensional Euclidian space

$\mathcal{H}_0$: binary hypothesis (null hypothesis)

$\mathcal{H}_1$: binary hypothesis (alternative hypothesis)

α : false alarm probability

β : detection probability

$\widehat{T}$: threshold value for the test statistic

Γ : log likelihood ratio

x^1 : parameter or local coordinate in differential geometry

T_m : set of all tangent vectors at m

$\overline{C}_i$: covariant vector field

$v^i w^j$: tensor product

δ^i_j : Kronecker delta tensor

g_{ij} : metric tensor

ds^2 : arc-length differential

$\widetilde{C}$: set containing all the local coordinates

C : covariance matrix

E[x] : mean of X

$\Lambda(y_1, y_2)$: Neyman-Pearson optimal detector

$\overrightarrow{X}$: tangent vector

J : Lagrange multiplier

$(D_{\overrightarrow{X}}h)\Big|_{Extreme}$: directional derivative

X, Y : random variable

K_n : curvature at a point

$\chi(M)$: Euler characteristic of M

$\Delta\alpha$: total amount of change in α

APPENDIX B

3 DIMENSION MODEL - MATLAB PROGRAM

```
% Guillaume Raux
% Spring 2004 - Fall 2005 - Summer 2005
% PhD Texas A&M University
clear; clc; close all;
% Nominal values
a0 = 1.5; b0 = 2; c0 = 1/2; s1 = 1.5; s2 = 0.5; epsi = .05; alpha =
0.05;

% Getting points @ the origine
% Ax^2+By^2=C and (a-a0)^2+(b-b0)^2+2(c-c0)^2=epsi^2 gives A=2, B=1 and
% C=epsi^2
partial = partial_points(2,1,epsi);

% Moving the manifold over a0, b0 and c0
points = final_points(partial,a0,b0,c0); Set_of_Points_not_posedef =
posedef(points)
% Computing the weighting factor
Weights = weighting2(points,epsi,2,1);

% Assigning weights to their corresponding points
% Weight_final = assigning(Weights,points);

AA = points(:,1); BB = points(:,2); CC = points(:,3);
% Computations for Sigma square positive
for i = 1 : length(AA)
Sigma_square_positive(i) =
AA(i)*(CC(i)*s2-s1*BB(i))^2+2*CC(i)*
(CC(i)*s2-s1*BB(i))*(CC(i)*s1-s2*AA(i))+(CC(i)*s1-s2*AA(i))^2*BB(i);

% Computations for partiel of c wrt a and b

partiel_c_a(i)=-(AA(i)-a0)/(2*(epsi-(AA(i)-a0)^2-(BB(i)-b0)^2))^.5;
partiel_c_b(i)= -(BB(i)-b0)/(2*(epsi-(AA(i)-a0)^2-(BB(i)-b0)^2))^.5;
% Computation for partiel of Sigma square wrt a and b
    term_1a = (CC(i)*s2-s1*BB(i))^2+2*AA(i)*partiel_c_a(i)*s2*(CC(i)*s2
    -s1*BB(i));
    term_2a = 2*partiel_c_a(i)*(CC(i)*s2-s1*BB(i))*(CC(i)*s1-s2*AA(i))+
    2*CC(i)*partiel_c_a(i)*s2*(CC(i)*s1-s2*AA(i))+2*CC(i)*(CC(i)*s2
    -s1*BB(i))*
    (partiel_c_a(i)*s1-s2);
    term_3a = 2*BB(i)*(partiel_c_a(i)*s1-s2)*(CC(i)*s1-s2*AA(i));
    Partiel_sigma_a(i) = term_1a + term_2a + term_3a;
    term_1b = 2*AA(i)*(CC(i)*s2-s1*BB(i))*(partiel_c_b(i)*s2-s1);
```

```
    term_2b = 2*partiel_c_b(i)*(CC(i)*s2-s1*BB(i))
    *(CC(i)*s1-s2*AA(i))+2*CC(i)*(partiel_c_b(i)*s2-s1)
    *(CC(i)*s1-s2*AA(i))
    +2*CC(i)*(CC(i)*s2-s1*BB(i))*partiel_c_b(i)*s1;
    term_3b = (CC(i)*s1-s2*AA(i))^2+2*BB(i)*partiel_c_b(i)
    *s1*(CC(i)*s1-s2*AA(i));
    Partiel_sigma_b(i) = term_1b + term_2b + term_3b;
end
% Computing the Threshold value for integral
nominal_variance =
    (c0*s2-b0*s1)^2*a0+2*c0*(c0*s2-b0*s1)*(c0*s1-a0*s2)
    +(c0*s1-a0*s2)^2*b0;
T = inverse_erf(alpha)*sqrt(nominal_variance);
% Computation of the Integrale
delta = 0.001; u = T:delta:1000;
% d_rond_a = Partiel_sigma_a(i);
% d_rond_b = Partiel_sigma_b(i);
for i = 1 : length(Partiel_sigma_a)
    d_rond_a = Partiel_sigma_a(i);
    d_rond_b = Partiel_sigma_b(i);
    koulchen = Sigma_square_positive(i);
    funky_1a = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_a*
    sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2a = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_a*
    sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_a(i) = funky_1a + funky_2a;
    funky_1b = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_b*
    sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2b = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_b*
    sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_b(i) = funky_1b + funky_2b;
end

% Biased
for i = 1 : length(funky_a)
    Slope_biased(i) = sqrt(funky_a(i)^2+funky_b(i)^2);
end
% Slope Unbiased case
for i = 1 : length(AA)
    g11 = 1+partiel_c_a(i)^2;
    g22 = 1+partiel_c_b(i)^2;
    g12 = partiel_c_a(i)*partiel_c_b(i);
    Slope_unbiased(i) = 1/(g11*g22-g12^2)*(g22*funky_a(i)^2-
    2*g12*funky_a(i)*funky_b(i)+g11*funky_b(i)^2);
end

% Assigning the according weight to the slope
results = histo(Slope_unbiased, Slope_biased,Weights); x =
0:0.005:2; plot(x,results(:,1),x,results(:,2)); legend('Unbiased
Case','Biased Case');
% Mean and Variance without the last bin
```

```
Mean_Variance =
meanvariance(results(1:end-1,1),results(1:end-1,2),x(1:end-1));
% Saving the results
save('Results_case_8');

function y = assigning(weights,points) A = []; for i = 1:37
    AA(180*(i-1)+1:180*i) = weights(i);
end AA = AA';
y = AA;

function y = final_points(partial,a0,b0,c0)

% Dummy Var
dummy = min(size(partial));
%dummy = max(size(partial));
% The function
A = partial(38:74,:); B = partial(75:end,:); C = partial(1:37,:);
for i = 1:37
    for j = 1:dummy
        A(i,j) = A(i,j)+a0;
        B(i,j) = B(i,j)+b0;
        C(i,j) = C(i,j)+c0;
    end
end surf(C,A,B);
% Change matrix into 3 column vector
u = 0; for i = 1:37
    for j = 1:dummy
        u = u + 1;
        AA(u) =  A(i,j);
        BB(u) =  B(i,j);
        CC(u) =  C(i,j);
    end
end

y = [AA; BB; CC]';

function y = histo2(U,B,Weights)
% Making a matrix of the result
U = U'; B = B'; AA = []; CC = []; for i = 1:37
    A =  sort(U(180*(i-1)+1:180*i));
    AA = [AA A]; % each column correspond to a weight
    C =  sort(B(180*(i-1)+1:180*i));
    CC = [CC C]; % each column correspond to a weight
end
% Matrix that we need to compare the values to
% x = 0:0.005:2; for bin size (401 steps)
Bin = []; for i = 1:2001
    Bin(i) = 0.005*(i-1);
end
% Finding the right spot in the matrix
Count = zeros(2001,37); for j = 1:37
```

```
    for i = 1:180
        for k = 1:2000
            if (AA(i,j)>Bin(k) & AA(i,j)<=Bin(k+1)),
                Count(k,j) = Count(k,j) + 1;
                k = 2000;
            end
            if(AA(i,j)>Bin(2001)),
                Count(2001,j) = Count(2001,j) + 1;
            end
        end
    end
end

Count2 = zeros(2001,37); for j = 1:37
    for i = 1:180
        for k = 1:2000
            if (CC(i,j)>Bin(k) & CC(i,j)<=Bin(k+1)),
                Count2(k,j) = Count2(k,j) + 1;
                k = 2000;
            end
            if(CC(i,j)>Bin(2001)),
                Count2(2001,j) = Count2(2001,j) + 1;
            end
        end
    end
end

% Getting the final histogram
for i = 1:37
    Count(:,i) = Weights(i).*Count(:,i);
    Count2(:,i) = Weights(i).*Count2(:,i);
end Dummy = cumsum(Count,2); Dummy2 = cumsum(Count2,2);
Final_Unbiased = Dummy(:,end); Final_Biased = Dummy2(:,end);

y = [Final_Unbiased Final_Biased];


function y = integral(threshold, sigmasquare)

delta = 0.001; u = threshold:delta:1000; funky =
sum(u.^2.*exp(-u.^2/(2*sigmasquare)).*delta); y = funky;

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
```

```
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end

y = i;

function y = meanvariance(U,B,x) U = U'; B = B'; for i = 1:length(x)
    xi(i) = ((i-1)+i)*0.005/2;
end Mean_Biased = sum(B.*xi)/sum(B) Mean_Unbiased =
sum(U.*xi)/sum(U)
% Var(X) = E[X^2]-E[X]^2 = 1/N*sum(yi-mean)^2
A = length(B); C = length(U); Var_Biased = 0; Var_Unbiased = 0; for
j = 1 : length(x)
    Var_Biased = Var_Biased + (B(i)-Mean_Biased)^2;
    Var_Unbiased = Var_Unbiased + (U(i)-Mean_Unbiased)^2;
end Var_Biased = 1/A*Var_Biased Var_Unbiased = 1/C*Var_Unbiased

y = [Mean_Biased Mean_Unbiased Var_Biased Var_Unbiased];

function y = partial_points(A,B,epsi) i = 0; for theta =
-pi/2:pi/36:pi/2
    i = i + 1;
    j = 0;
    for phi = 0:pi/9:2*pi
        j = j + 1;
        c(i,j) = sqrt(epsi/(A*sin(theta)^2+B*cos(theta)^2))*sin(theta);
        a(i,j) = sqrt(epsi/(A*sin(theta)^2+B*cos(theta)^2))*cos(theta)
                 *cos(phi);
        b(i,j) = sqrt(epsi/(A*sin(theta)^2+B*cos(theta)^2))*cos(theta)
                 *sin(phi);
    end
end y = [a; b; c];

function y = posedef(points)
% points is a 2-by-2 matrix with constant coefficients
% We need to represent all the points as matrix = [a c; c b]
count = 0; for i = 1:length(points)
    set = points(i,1)*points(i,2)-points(i,3)^2;
    if (set < 0),
        points(i,:) = 0;
        count = count + 1;
    end
end

y = count;
```

```
function y = variance(a,b,c,d,e,f,s1,s2,s3)

A = -2*s1*(b*c-d*f)-d*s2*(e-c)-2*s3*(d*f-b*e)-f*s2*(e-c); B =
-2*s2*(a*c-e^2)-d*s1*(e-c)-f*s1*(e-c)-f*s3*(e-a)-d*s3*(e-a); C =
-2*s3*(a*b-d*f)-2*s1*(d*f-b*e)-f*s2*(e-a)-d*s2*(e-a);

sigma_square = a*A^2+b*B^2+c*C^2+2*A*B*d+2*A*C*e+2*B*C*f; y =
sigma_square;

function y = weighting2(points,epsi,A,B) i = 0; for theta =
-pi/2:pi/36:pi/2
    i = i + 1;
    %gamma = theta-atan(A/B*tan(theta));
    %w(i) = abs(epsi*cos(theta)/(A*sin(theta)^2
            +B*cos(theta)^2)/cos(gamma));
    w2(i) = 2*epsi*sqrt(sin(theta)^2*cos(theta)^2
            +0.25*cos(theta)^4);
end

y = w2';
```

APPENDIX C

THE SADDLE SURFACE

```
% Guillaume Raux
% Spring 2005 - Fall 2004 - Summer 2005
% PhD Texas A&M University
clear; clc; close all;
% Nominal values
epsi = .05; alpha = 0.05; lambda = 0.5; L = 1; a = 1.5; b = 1; s1 =
1.5; s2 = 2; factor = 2;

%Computing the corresponding c0 tilda
c0 = cotilda(a,b,lambda);
% Finding points on the saddle c = lambda*a'*b'
partial = points2(L,a,b,epsi,factor); Points =
final_points(partial,lambda);

Set_of_Points_not_posedef = posedef(Points)
% Computing the weighting factor and add it to the Points matrix
Points_and_Weights = weighting(Points,lambda);

% I can only use the points that are positive definite
if (Set_of_Points_not_posedef > 0),
    Points_and_Weights = fixed(Points_and_Weights);
end

% The 4th column of Points contains the weighting factors
AA = Points_and_Weights(:,1); BB = Points_and_Weights(:,2); CC =
Points_and_Weights(:,3); Weighting_factors =
Points_and_Weights(:,4);

% Computations for Sigma square positive
for i = 1 : length(AA)
    Sigma_square(i) = AA(i)*(CC(i)*s2-s1*BB(i))^2+2*CC(i)*(CC(i)*s2
    -s1*BB(i))*(CC(i)*s1-s2*AA(i))+(CC(i)*s1-s2*AA(i))^2*BB(i);
% Computations for partiel of c wrt a and b
    partiel_c_a(i) = (2*AA(i))*lambda*AA(i)^2*(AA(i)^2*BB(i)^2)^(-.5);
    partiel_c_b(i) = (2*BB(i))*lambda*BB(i)^2*(AA(i)^2*BB(i)^2)^(-.5);
% Computation for partiel of Sigma square wrt a and b
    term_1a = (CC(i)*s2-s1*BB(i)^2)^2+2*AA(i)^2*partiel_c_a(i)*s2*(CC(i)
    *s2-s1*BB(i)^2);
    term_2a = 2*partiel_c_a(i)*(CC(i)*s2-s1*BB(i)^2)*(CC(i)*s1-s2*AA(i)^2)
    +2*CC(i)*partiel_c_a(i)*s2*(CC(i)*s1-s2*AA(i)^2)+2*CC(i)*(CC(i)*s2-s1
    *BB(i)^2)*(partiel_c_a(i)*s1-s2);
    term_3a = 2*BB(i)^2*(partiel_c_a(i)*s1-s2)*(CC(i)*s1-s2*AA(i)^2);
    Partiel_sigma_a(i) = 2*AA(i)*(term_1a + term_2a + term_3a);
```

```
    term_1b = 2*AA(i)^2*(CC(i)*s2-s1*BB(i)^2)*(partiel_c_b(i)*s2-s1);
    term_2b = 2*partiel_c_b(i)*(CC(i)*s2-s1*BB(i)^2)*(CC(i)*s1
              -s2*AA(i)^2)+2*CC(i)*(partiel_c_b(i)*s2-s1)*(CC(i)
              *s1-s2*AA(i)^2)+2*CC(i)*(CC(i)*s2-s1*BB(i)^2)
              *partiel_c_b(i)*s1;
    term_3b = (CC(i)*s1-s2*AA(i)^2)^2+2*BB(i)^2*partiel_c_b(i)*s1
              *(CC(i)*s1-s2*AA(i)^2);
    Partiel_sigma_b(i) = 2*BB(i)*(term_1b + term_2b + term_3b);
end
% Computing the Threshold value for integral
nominal_variance =
(c0*s2-b*s1)^2*a+2*c0*(c0*s2-b*s1)*(c0*s1-a*s2)+(c0*s1-a*s2)^2*b; T
= inverse_erf(alpha)*sqrt(nominal_variance);
% Computation of the Integrale
delta = 0.001; u = T:delta:1000;

for i = 1 : length(Partiel_sigma_a)
    d_rond_a = Partiel_sigma_a(i);
    d_rond_b = Partiel_sigma_b(i);
    koulchen = Sigma_square(i);
    funky_1a = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_a
    *sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2a = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_a
    *sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_a(i) = funky_1a + funky_2a;
    funky_1b = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_b
    *sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2b = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_b
    *sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_b(i) = funky_1b + funky_2b;
end

% Biased
for i = 1 : length(funky_a)
    Slope_biased(i) = sqrt(funky_a(i)^2+funky_b(i)^2);
end
% Slope Unbiased case
for i = 1 : length(AA)
    g11 = 1+partiel_c_a(i)^2;
    g22 = 1+partiel_c_b(i)^2;
    g12 = partiel_c_a(i)*partiel_c_b(i);
    Slope_unbiased(i) = 1/(g11*g22-g12^2)*
    (g22*funky_a(i)^2-2*g12*funky_a(i)*funky_b(i)+g11*funky_b(i)^2);
end

% Assigning the according weight to the slope
results = histo2(Slope_unbiased, Slope_biased,Weighting_factors); x
= 0:0.005:4; plot(x,results(:,1),x,results(:,2)); legend('Unbiased
Case','Biased Case');
% Mean and Variance without the last bin
Mean_Variance =
```

```
meanvariance(results(1:end-1,1),results(1:end-1,2),x(1:end-1));
% Saving the results
save('Results_case_5');

function y = assigning(weights,points) A = []; for i = 1:37
    AA(180*(i-1)+1:180*i) = weights(i);
end
AA = AA';
y = AA;

function y = cotilda(a,b,lambda)
c0 = lambda*a*b;
y = c0;

function y = final_points(partial,lambda) A = partial(:,1); B =
partial(:,2); for i = 1:length(partial)
    C(i) = lambda*A(i)*B(i);
end C = C'; y = [A B C];

function y = fixed(Points)

for i = 1:length(Points)
    set = Points(i,1)*Points(i,2)-Points(i,3)^2;
    if (set < 0),
        Points(i,:) = 0;
    end
end A = []; B = []; C = []; D = []; for j = 1 : length(Points)
    if (sum(xor(Points(j,:),[0 0 0 0]) ~= 0)),
        dummy_A = Points(j,1);
        dummy_B = Points(j,2);
        dummy_C = Points(j,3);
        dummy_D = Points(j,4);
        A = [A ; dummy_A];
        B = [B ; dummy_B];
        C = [C ; dummy_C];
        D = [D ; dummy_D];
    end
end
%Box = A + sqrt(-1)*B;
%plot(Box,'*');
y = [A B C D];

function y = histo2(U,B,Weights)
% Making a matrix of the result
U = U'; B = B'; A = [B Weights]; C = [U Weights];

% Sort 1st row of matrix and corresponding weights
AA = sortrows(A,1); CC = sortrows(C,1);

% Matrix that we need to compare the values to
% x = 0:0.005:4; for bin size (401 steps)
```

```
Bin = []; for i = 1:801
    Bin(i) = 0.005*(i-1);
end
% Finding the right spot in the matrix
Count = zeros(1,801); for j = 1:length(AA)
    for k = 1:800
            if (AA(j)>Bin(k) & AA(j)<=Bin(k+1)),
                Count(k) = Count(k) + AA(j,2);
                k = 800;
            end
            if(AA(j)>Bin(801)),
                Count(801) = Count(801) + AA(j,2);
            end
    end
end

Count2 = zeros(1,801); for j = 1:length(CC)
        for k = 1:800
            if (CC(j)>Bin(k) & CC(j)<=Bin(k+1)),
                Count2(k) = Count2(k) + CC(j,2);
                k = 800;
            end
            if(CC(j)>Bin(801)),
                Count2(801) = Count2(801) + CC(j,2);
            end
        end
end

% Getting the final histogram
Final_Unbiased = Count'; Final_Biased = Count2';

y = [Final_Unbiased Final_Biased];

function y = integral(threshold, sigmasquare)

delta = 0.001; u = threshold:delta:1000; funky =
sum(u.^2.*exp(-u.^2/(2*sigmasquare)).*delta); y = funky;

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
```

```
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
%error_function_equals_to = input
%corresponding_value_of_x = i
y = i;

function y = meanvariance(U,B,x) U = U'; B = B'; for i = 1:length(x)
    xi(i) = ((i-1)+i)*0.005/2;
end Mean_Biased = sum(B.*xi)/sum(B) Mean_Unbiased =
sum(U.*xi)/sum(U)
% Var(X) = E[X^2]-E[X]^2 = 1/N*sum(yi-mean)^2
A = length(B); C = length(U); Var_Biased = 0; Var_Unbiased = 0; for
j = 1 : length(x)
    Var_Biased = Var_Biased + (B(i)-Mean_Biased)^2;
    Var_Unbiased = Var_Unbiased + (U(i)-Mean_Unbiased)^2;
end Var_Biased = 1/A*Var_Biased Var_Unbiased = 1/C*Var_Unbiased

y = [Mean_Biased Mean_Unbiased Var_Biased Var_Unbiased];

function y = points2(L,a,b,epsi,factor) j = sqrt(-1); origine =
a+j*b;
% Design of the box and its intersections
box = []; index = 0; for x = 1 : (L/(epsi/factor))+1
    for y = 1 : (L/(epsi/factor))+1
        index = index + 1;
        box(x,y) = (a+(x-1)*(epsi/factor))+j*(b+(y-1)*(epsi/factor));
        real_part(index) = a+(x-1)*epsi/factor;
        imaginary_part(index) = b+(y-1)*epsi/factor;
    end
end
%plot(box,'*');
y = [real_part; imaginary_part]';

function y = posedef(points)
% points is a 2-by-2 matrix with constant coefficients
% We need to represent all the points as matrix = [a c; c b]
count = 0; for i = 1:length(points)
    set(i) = points(i,1)*points(i,2)-points(i,3)^2;
    if (set(i)<0),
        count = count + 1;
    end
end

y = count;

function y = variance(a,b,c,d,e,f,s1,s2,s3)

A = -2*s1*(b*c-d*f)-d*s2*(e-c)-2*s3*(d*f-b*e)-f*s2*(e-c); B =
```

```
-2*s2*(a*c-e^2)-d*s1*(e-c)-f*s1*(e-c)-f*s3*(e-a)-d*s3*(e-a); C =
-2*s3*(a*b-d*f)-2*s1*(d*f-b*e)-f*s2*(e-a)-d*s2*(e-a);

sigma_square = a*A^2+b*B^2+c*C^2+2*A*B*d+2*A*C*e+2*B*C*f; y =
sigma_square;

function y = weighting(Points,lambda)

for i = 1 : length(Points)
    w(i) = sqrt(1+lambda^2*Points(i,1)^2+lambda^2*Points(i,2)^2);
end Points = [Points w']; y = Points;
```

APPENDIX D

THE 6 DIMENSION MODEL

```
% Guillaume Raux
% Spring 2004 - Fall 2005 - Summer 2005
% PhD Texas A&M University
% Simulation for the stationary case in 6D
clear all; close all; clc;
% Nominal values
a0 = 1.5; d0 = -3/4 ; f0 = 1/2; s1 = 1; s2 = 1/4; s3 = 1/8; epsi =
1.5*10^(-3); alpha = 0.05; factor = 3; delta = 0.005; m = sqrt(2);
Infinity = 70;

% Getting the points
partial = partial_points2(a0, d0, epsi, factor);

% Getting the corresponding f points
final = final_points(partial, f0, epsi, a0, d0);

% Positive def
non_posi = posi_def(final);

% Weighting factor
weight = weighting_factor(final, a0, d0, epsi);

% Assigning weights to their corresponding points
%Final_with_weights = [final; weight];

% Nominal variance
Nominal = nominal_variance(a0, d0, f0, s1, s2, s3);

% Threshold value
T = inverse_erf(alpha)*sqrt(Nominal);

% Computations for Sigma square positive

AA = final(1,:); BB = final(2,:); CC = final(3,:); DD = weight;


A = 2*s1.*(AA.^2-BB.^2./2)+2*s2.*(BB.*CC./2-AA.*BB./m)+2*s3.
    *(BB.^2./2-AA.*CC./m);
B = 2*s1.*(BB.*CC./2-AA.*BB./m)+2*s2.*(AA.^2-CC.^2./2)+2*s3.
    *(BB.*CC./2-AA.*BB./m);
C = 2*s1.*(BB.^2./2-AA.*CC./m)+2*s2.*(BB.*CC./2-AA.*BB./m)+2*s3.
    *(AA.^2-BB.^2./2);
Sigma_square_positive =
```

```
AA.*(A.^2+B.^2+C.^2)+2.*A.*B.*BB./m+2.*A.*C.*CC./m+2.*B.*C.*BB./m;
    %%%%%%% Computation for partiel of f wrt to a and d

partiel_f_a = -3.*(AA-a0)./sqrt(epsi-3.*(AA-a0).^2-2.*(BB-d0).^2);
partiel_f_d = -2.*(BB-d0)./sqrt(epsi-3.*(AA-a0).^2-2.*(BB-d0).^2);

    %%%%%%% Computation for partiel of Sigma square wrt to a and d
term_1aa = 2.*(4.*AA.*s1-m.*BB.*s2-m.*CC.*s3).*A; term_1bb =
2.*(-m.*BB.*s1+4.*AA.*s2-m.*BB.*s3).*B; term_1cc =
2.*(-m.*CC.*s1-m.*BB.*s2+4.*AA.*s3).*C; term_1a =
A.^2+B.^2+C.^2+AA.*(term_1aa+term_1bb+term_1cc); term_2a =
m.*BB.*(B.*(4.*AA.*s1-m.*BB.*s2-m.*CC.*s3)
+A.*(-m.*BB.*s1+4.*AA.*s2-m.*BB.*s3));
term_3a=m.*CC.*(C.*(4.*AA.*s1-m.*BB.*s2-m.*CC.*s3)
+A.*(-m.*CC.*s1-m.*BB.*s2+4.*AA.*s3));
term_4a =
m.*BB.*(C.*(-m.*BB.*s1+4.*AA.*s2-m.*BB.*s3)
+B.*(-m.*CC.*s1-m.*BB.*s2+4.*AA.*s3));
Partiel_sigma_a = term_1a + term_2a + term_3a + term_4a;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
term_2aa = 2.*(-2.*BB.*s1+2.*s2.*(CC./2-AA./m)+2.*BB.*s3).*A;
term_2bb = 2.*(2.*s1.*(CC./2-AA./m)+2.*s3.*(CC./2-AA./m)).*B;
term_2cc = 2.*(2.*BB.*s1+2.*s2.*(CC./2-AA./m)
-2.*BB.*s3).*C; term_1d
= AA.*(term_2aa+term_2bb+term_2cc); term_2d =
m.*BB.*(B.*(-2.*BB.*s1+2.*s2.*(CC./2-AA./m)+2.*BB.*s3)
+A.*(2.*s1.*(CC./2-AA./m)+2.*s3.*(CC./2-AA./m)));
term_3d =
m.*A.*B+m.*CC.*(C.*(-2.*BB.*s1+2.*s2.*(CC./2-AA./m)+2.*BB.*s3)
+A.*(2.*BB.*s1+2.*s2.*(CC./2-AA./m)-2.*BB.*s3));
term_4d =
m.*BB.*(C.*(2.*s1.*(CC./2-AA./m)+2.*s3.*(CC./2-AA./m))
+B.*(2.*BB.*s1+2.*s2.*(CC./2-AA./m)-2.*BB.*s3));
term_5d = m.*B.*C; Partiel_sigma_d = term_1d + term_2d + term_3d +
term_4d + term_5d;


% Computation of the integral
u = T : delta : Infinity; for i = 1 : length(Partiel_sigma_a)
    d_rond_a = Partiel_sigma_a(i);
    d_rond_d = Partiel_sigma_d(i);
    koulchen = Sigma_square_positive(i);
    funky_1a = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_a
    *sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2a = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_a
    *sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_a(i) = funky_1a + funky_2a;
    funky_1d = -.5*(koulchen^(-3/2))/sqrt(2*pi)*d_rond_d
    *sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2d = 1/(2*sqrt(2*pi)*koulchen^(5/2))*d_rond_d
    *sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
```

```
    funky_d(i) = funky_1d + funky_2d;
end

% Getting red of the complex elements in funky_a and funky_d
if (sum(abs(imag(funky_a))) > 0),
    dummy=onlyreal2(funky_a,funky_d,weight,partiel_f_a,partiel_f_d);
    funky_a = dummy(1,:);
    funky_d = dummy(2,:);
    weight = dummy(3,:);
    partiel_f_a = dummy(4,:);
    partiel_f_d = dummy(5,:);
    final_number_of_points = length(weight)
end

% Slope Unbiased case
g11 = 3+partiel_f_a.^2; g22 = 2+partiel_f_d.^2; g12 =
partiel_f_a.*partiel_f_d; Slope_unbiased =
1./(g11.*g22-g12.^2).
*(g22.*funky_a.^2-2.*g12.*funky_a.*funky_d+g11.*funky_d.^2);

save('Results_XX','Slope_unbiased','weight'); clear all; load
Results_XX;
% Assigning the according weight to the slope
results = histo2(Slope_unbiased, weight); Normalized_results =
Norm(results); x = 0:0.005:2;
%plot(x,results);
%legend('Unbiased Case');

% Mean and Variance without the last bin
Mean_Variance = meanvariance(results(1:end-1),x(1:end-1));
% Median, Mode and standart deviation
Median = over50(results,x) Mode = mode(results, x) Over_90_pourcent
= over90(results,x) Over_75_pourcent = over75(results,x)
% Saving the results
save('Results_XX');

function y = weighting_factor(final, a0, d0, epsi) for i = 1 :
length(final)
    weight(i) = sqrt(((epsi+6*(final(1,i)-a0)^2+2*(final(2,i)-d0)^2))
    /(epsi-3*(final(1,i)-a0)^2-2*(final(2,i)-d0)^2));
end y = weight;

function y = final_points(partial, f0, epsi, a0, d0) for i = 1 :
length(partial)
    f(i) = f0+sqrt(epsi-3*(partial(1,i)-a0)^2-2*(partial(2,i)-d0)^2);
end y = [partial(1,:); partial(2,:); f];

function y = S_square_posi(epsi, a0, d0, f0, s1, s2, s3,
Final_with_weights)

m = sqrt(2); AA = Final_with_weights(1,:); BB =
```

```
Final_with_weights(2,:); CC = Final_with_weights(3,:); DD =
Final_with_weights(4,:);

for i = 1 : length(AA)
    A = 2*s1*(AA(i)^2-BB(i)^2/2)+2*s2*(BB(i)*CC(i)/2-AA(i)*BB(i)/m)
    +2*s3*(BB(i)^2/2-AA(i)*CC(i)/m);
    B = 2*s1*(BB(i)*CC(i)/2-AA(i)*BB(i)/m)+2*s2*(AA(i)^2-CC(i)^2/2)
    +2*s3*(BB(i)*CC(i)/2-AA(i)*BB(i)/m);
    C = 2*s1*(BB(i)^2/2-AA(i)*CC(i)/m)+2*s2*(BB(i)*CC(i)/2-AA(i)
              *BB(i)/m)+2*s3*(AA(i)^2-BB(i)^2/2);
    dummy(i) = AA(i)*(A^2+B^2+C^2)+2*A*B*BB(i)/m+2*A*C*CC(i)/m
               +2*B*C*BB(i)/m;
    %%%%%% Computation for partiel of f wrt to a and d

    partiel_f_a(i) = -3*(AA(i)-a0)/sqrt(epsi-3*(AA(i)-a0)^2
                     -2*(DD(i)-d0)^2);
    partiel_f_d(i) = -2*(DD(i)-d0)/sqrt(epsi-3*(AA(i)-a0)^2
                     -2*(DD(i)-d0)^2);

    %%%%%% Computation for partiel of Sigma square wrt to a and d
    term_1aa = 2*(4*AA(i)*s1-m*BB(i)*s2-m*CC(i)*s3)*A;
    term_1bb = 2*(-m*BB(i)*s1+4*AA(i)*s2-m*BB(i)*s3)*B;
    term_1cc = 2*(-m*CC(i)*s1-m*BB(i)*s2+4*AA(i)*s3)*C;
    term_1a = A^2+B^2+C^2+AA(i)*(term_1aa+term_1bb+term_1cc);
    term_2a = m*BB(i)*(B*(4*AA(i)*s1-m*BB(i)*s2-m*CC(i)*s3)
    +A*(-m*BB(i)*s1+4*AA(i)*s2-m*BB(i)*s3));
    term_3a = m*CC(i)*(C*(4*AA(i)*s1-m*BB(i)*s2-m*CC(i)*s3)
    +A*(-m*CC(i)*s1-m*BB(i)*s2+4*AA(i)*s3));
    term_4a = m*BB(i)*(C*(-m*BB(i)*s1+4*AA(i)*s2-m*BB(i)*s3)
    +B*(-m*CC(i)*s1-m*BB(i)*s2+4*AA(i)*s3));
    Partiel_sigma_a(i) = term_1a + term_2a + term_3a + term_4a;
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    term_1aa = 2*(-2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)+2*BB(i)*s3)*A;
    term_1bb = 2*(2*s1*(CC(i)/2-AA(i)/m)+2*s3*(CC(i)/2-AA(i)/m))*B;
    term_1cc = 2*(2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)-2*BB(i)*s3)*C;
    term_1d = AA(i)*(term_1aa+term_1bb+term_1cc);
    term_2d = m*BB(i)*(B*(-2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)
    +2*BB(i)*s3)+A*(2*s1*(CC(i)/2-AA(i)/m)+2*s3*(CC(i)/2-AA(i)/m)));
    term_3d = m*A*B+m*CC(i)*(C*(-2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)
    +2*BB(i)*s3)+A*(2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)-2*BB(i)*s3));
    term_4d = m*BB(i)*(C*(2*s1*(CC(i)/2-AA(i)/m)+2*s3*(CC(i)/2-AA(i)/m))
    +B*(2*BB(i)*s1+2*s2*(CC(i)/2-AA(i)/m)-2*BB(i)*s3));
    term_5d = m*B*C;
    Partiel_sigma_d(i) = term_1d + term_2d + term_3d
                         + term_4d + term_5d;
end y = [Partiel_sigma_a ; Partiel_sigma_d];

function y = posi_def(final) a = final(1,:); d = final(2,:); f =
final(3,:);

m = sqrt(2); count = 0; for i = 1 : length(a)
```

```
    G = [a(i) d(i)/m f(i)/m ;d(i)/m a(i) d(i)/m ;f(i)/m d(i)/m a(i)];
    eigenvalues = eig(G);
    for j = 1 : length(eigenvalues)
        if ( eigenvalues(j) < 0),
            count = count + 1;
        end
    end
end y = count;

function y = partial_points2(a0, d0, epsi, factor)

% 3(a-a0)^2+2(d-d0)^2+(f-f0)^2=epsi
% Gives an ellipse of equation 3A^2+2D^2=epsi after projection

j = sqrt(-1); offset = epsi/(10*factor);
% Design of the box and its intersections
box = []; index = 0; abc = factor*10*6; for x = 1 : abc
    for y = 1 : abc
        index = index + 1;
        box(x,y) = (a0-3*epsi+(x-1)*offset)+j
                    *(d0-3*epsi+(y-1)*offset);
        real_part(index) = (a0-3*epsi+(x-1)*offset);
        imaginary_part(index) = (d0-3*epsi+(y-1)*offset);
    end
end
%plot(box,'*');
aaa = size(box); number_of_points_on_grid = aaa(1)*aaa(2)
% Needs to respect the equation of the ellipse
A = real_part'; D = imaginary_part';

Set = []; for i = 1 : length(A)
    result = 3*(A(i)-a0)^2+2*(D(i)-d0)^2;
    if (result - epsi > 0) % fail the test
        A(i) = 0;
        D(i) = 0;
    end
end
% Taking care of only the none zero elements
coord_D = []; coord_A = []; for i = 1 : length(D)
    if (A(i) ~= 0 ),
        coordinates = D(i);
        coordinate = A(i);
        coord_D = [coord_D ;coordinates];
        coord_A = [coord_A ;coordinate];
    end
end D = coord_D; A = coord_A; inside_ellipse = length(A)
y = [A D]';

function y = over90(results,x) a = sum(results); b = 0.9 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
```

```
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_90 = x(index); y = Over_90;

function y = over75(results,x) a = sum(results); b = 0.75 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_75 = x(index); y = Over_75;

function y = over50(results,x) a = sum(results); b = 0.5 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_50 = x(index); y = Over_50;

function y = onlyreal2(funky_a, funky_d, weight, partiel_f_a,
partiel_f_d)

aaa = zeros(1,length(funky_a)); bbb = zeros(1,length(funky_d));

ccc = find(imag(funky_a)>0);

for j = 1 : length(ccc)
    funky_a(ccc(j)) = 0;
    funky_d(ccc(j)) = 0;
    weight(ccc(j)) = 0;
    partiel_f_a(ccc(j)) = 0;
    partiel_f_d(ccc(j)) = 0;
end

% Keep only the non zero elements
dummy1 = []; dummy2 = []; dummy3 = []; dummy4 = []; dummy5 = []; for
i = 1 : length(funky_a)
    if (funky_a(i) ~= 0 ),
        coordinates = funky_a(i);
        coordinate = funky_d(i);
        coordinat = weight(i);
        coordina = partiel_f_a(i);
        coordin = partiel_f_d(i);
        dummy1 = [dummy1 ;coordinates];
        dummy2 = [dummy2 ;coordinate];
        dummy3 = [dummy3 ;coordinat];
        dummy4 = [dummy4 ; coordina];
        dummy5 = [dummy5 ; coordin];
    end
end y = [dummy1 dummy2 dummy3 dummy4 dummy5]';

function y = Norm(results) a = norm(results); b = results / a;

y = b;
```

```
function y = nominal_variance(a0, d0, f0, s1, s2, s3) m = sqrt(2); A
= 2*s1*(a0^2-d0^2/2)+2*s2*(d0*f0/2-a0*d0/m)+2*s3*(d0^2/2-a0*f0/m); B
= 2*s1*(d0*f0/2-a0*d0/m)+2*s2*(a0^2-f0^2/2)+2*s3*(d0*f0/2-a0*d0/m);
C = 2*s1*(d0^2/2-a0*f0/m)+2*s2*(d0*f0/2-a0*d0/m)+2*s3*(a0^2-d0^2/2);

dummy = a0*(A^2+B^2+C^2)+2*A*B*d0/m+2*A*C*f0/m+2*B*C*d0/m;

y = dummy;

function y = mode(results,x)
% In statistics, the mode is the value that has the largest number of
% observations, namely the most frequent value or values.
% The mode is not necessarily unique, unlike the arithmetic mean.

[a, index] = max(results); dummy = (x(index)+x(index+1))/2; y =
dummy;

function y = meanvariance(U,x)
%U = U';
for i = 1:length(x)
    xi(i) = ((i-1)+i)*0.005/2;
end Mean_Unbiased = sum(U.*xi)/sum(U)
% Var(X) = E[X^2]-E[X]^2 = 1/N*sum(yi-mean)^2
C = length(U); Var_Unbiased = 0; for j = 1 : length(x)
    Var_Unbiased = Var_Unbiased + (U(i)-Mean_Unbiased)^2;
end Var_Unbiased = 1/C*Var_Unbiased y = [Mean_Unbiased
Var_Unbiased];

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
%error_function_equals_to = input
%corresponding_value_of_x = i
y = i;

function y = integr(T, delta, epsi, a0, d0, f0, s1, s2, s3,
```

```
Final_with_weights, Partiel_sigma_a_and_d) u = u = T:delta:1000; m =
sqrt(2); AA = Final_with_weights(1,:); BB = Final_with_weights(2,:);
CC = Final_with_weights(3,:); DD = Final_with_weights(4,:);


function y = histo2(U, weight)

A = U;
% Matrix that we need to compare the values to
% x = 0:0.005:2; for bin size (401 steps)
Bin = []; for i = 1:401
    Bin(i) = 0.005*(i-1);
end

% Finding the right spot in the matrix
Count = zeros(length(A),401);

for i = 1 : length(A)
    for k = 1:400
        if (A(i)>Bin(k) & A(i)<=Bin(k+1)),
            Count(i,k) = weight(i);
            k = 400;
        end
        if(A(i)>Bin(401)),
            Count(i,401) = weight(i);
        end
    end
end dummy =  cumsum(Count); Count = dummy(end,:); y = Count;
```

APPENDIX E

LARGE SAMPLE EXAMPLE

```
clear all; clc;

epsi_n = 1/8; a0 = 1; c0 = -1/2; alpha = 0.005; n = 10; delta =
0.005; factor = 10000;
% Find a's and corresponding c's
a = -epsi_n/n:epsi_n/(factor):epsi_n/n-epsi_n/(factor); c =
sqrt(epsi_n/n - a.^2)/sqrt(n-1); Infinity = 70;

% Shift them using the nominal values
a = a0 + a; c = c0 + c;

% Weighting factor
partial_c_a = -(a - a0)./sqrt((n-1).*(epsi_n/n-(a-a0).^2)); WF =
sqrt(1 + partial_c_a.^2);

% Nominal variances
sigma_lambda_square_nomi = n*a0 + (n^2-n)*c0; Nominal =
sigma_lambda_square_nomi;
% Sigma Square
Sigma_square = n.*a + (n^2-n).*c;

% Threshold value
T = inverse_erf(alpha)*sqrt(Nominal);
% Computation of the integral
Partial_sigma_a = n - (n^2-n).*(a -
a0)./sqrt((n-1)*(epsi_n/n-(a-a0).^2)); u = T : delta : Infinity; for
i = 1 : length(Partial_sigma_a)
    funky_a1(i) = -.5*Sigma_square(i)^(-3/2)*
    sum(exp(-u.^2/(2*Sigma_square(i))).*delta);
    funky_a2(i) = 1/(2*sqrt(2*pi)*Sigma_square(i)^(5/2))
    *Partial_sigma_a(i)*sum(u.^2.*exp(-u.^2/(2*Sigma_square(i))).*delta);
end partial_alpha_a = 1/(2*pi).*Partial_sigma_a.*(funky_a1 +
funky_a2);

% Slopes
g11 = 1 + (partial_c_a).^2; Slopes =
abs(partial_alpha_a).*g11.^(-1/2); save('Results_3a','Slopes','WF');
clear all; load Results_3a;
% Assigning the according weight to the slope
results = histo2(Slopes, WF); Normalized_results = Norm(results); x
= 0:0.0005:2; plot(x,Normalized_results); axis([0 .07 0 1]);
legend('Slopes for n = 10');
```

```
% Mean and Variance without the last bin
Mean_Variance = meanvariance(results(1:end-1),x(1:end-1));
% Median, Mode and standart deviation
Median = over50(results,x) Mode = mode(results, x) Over_90_pourcent
= over90(results,x) Over_75_pourcent = over75(results,x)
% Saving the results
save('Results_3a');

function y = over90(results,x) a = sum(results); b = 0.9 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_90 = x(index); y = Over_90;

function y = over75(results,x) a = sum(results); b = 0.75 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_75 = x(index); y = Over_75;

function y = over50(results,x) a = sum(results); b = 0.5 * a; Sum =
[]; dummy = 0; for i = 1:length(results)
    dummy = dummy + results(i);
    Sum = [Sum  dummy];
end index = min(find(Sum > b)); Over_50 = x(index); y = Over_50;

function y = Norm(results) a = norm(results); b = results / a;

y = b;

function y = mode(results,x)
% In statistics, the mode is the value that has the largest number of
% observations, namely the most frequent value or values.
% The mode is not necessarily unique, unlike the arithmetic mean.

[a, index] = max(results); dummy = (x(index)+x(index+1))/2; y =
dummy;

function y = meanvariance(U,x)
%U = U';
for i = 1:length(x)
    xi(i) = ((i-1)+i)*0.005/2;
end Mean_Unbiased = sum(U.*xi)/sum(U)
% Var(X) = E[X^2]-E[X]^2 = 1/N*sum(yi-mean)^2
C = length(U); Var_Unbiased = 0; for j = 1 : length(x)
    Var_Unbiased = Var_Unbiased + (U(i)-Mean_Unbiased)^2;
end Var_Unbiased = 1/C*Var_Unbiased y = [Mean_Unbiased
Var_Unbiased];

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
```

```
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
%error_function_equals_to = input
%corresponding_value_of_x = i
y = i;

function y = histo2(U, weight)

A = U;
% Matrix that we need to compare the values to
% x = 0:0.0005:2; for bin size (401 steps)
Bin = []; for i = 1:4001
    Bin(i) = 0.0005*(i-1);
end

% Finding the right spot in the matrix
Count = zeros(length(A),4001);

for i = 1 : length(A)
    for k = 1:4000
        if (A(i)>Bin(k) & A(i)<=Bin(k+1)),
            Count(i,k) = weight(i);
            k = 4000;
        end
        if(A(i)>Bin(4001)),
            Count(i,4001) = weight(i);
        end
    end
end dummy =  cumsum(Count); Count = dummy(end,:); y = Count;
```

APPENDIX F

EXTENSION TO GENERALIZED GAUSSIAN

```
% Old simulation with new parameters surface
% Guillaume Raux
% Spring 2004 - Fall 2005
% PhD Texas A&M University

clear all; clc;
% Nominal values
a0 = 1/2; b0 = 1/4; c0 = 0; s1 = 1/2; s2 = 1/4; epsi = 0.05; alpha =
0.05; delta = 0.001;


% Selection of Points
Points = circle(a0 ,b0 ,c0, epsi); Number_of_points =
min(size(Points))*max(size(Points))/2; A = Points(:,1); B =
Points(:,2); C = zeros(1,length(A))';
% Computation of Sigma square nominal
Sigma_square_nom = a0*b0^2*s1^2 + a0^2*b0*s2^2;

% Computations for Sigma square positive
Partial_b_a = -(A-a0).*(epsi-(A-a0).^2).^(1/2); Partial_b2_a =
-2*b0*Partial_b_a - 2*(A-a0); Partial_sigma_a = (B.^2 +
A.*Partial_b2_a)*s1^2 + (2.*A.^2.*B + A.^2.*Partial_b_a)*s2^2;
Sigma_square = A.*B.^2*s1^2 + A.^2.*B*s2^2;

% Computation of the threshold
T = inverse_erf(alpha)*sqrt(Sigma_square_nom);

% Computation of the Integrale
u = T : delta : 200; for i = 1 : length(Partial_sigma_a)
    d_rond_a = Partial_sigma_a(i);
    koulchen = Sigma_square(i);
    funky_1a = -.5*(koulchen^(-3/2))/sqrt(2*pi)
    *d_rond_a*sum(exp(-u.^2/(2*koulchen)).*delta);
    funky_2a = 1/(2*sqrt(2*pi)*koulchen^(5/2))
    *d_rond_a*sum(u.^2.*exp(-u.^2/(2*koulchen)).*delta);
    funky_a(i) = funky_1a + funky_2a;
end funky_a = funky_a';
% Slope Unbiased case
Slope_unbiased = sqrt(funky_a./(1 + Partial_sigma_a.^2));

% Assigning the according weight to the slope
xxx = -.05:0.001:0.5; figure; hist(Slope_unbiased,xxx); axis([0 0.5
0 500]);
```

```
% Saving the results
save('Results_case_4');

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
%error_function_equals_to = input
%corresponding_value_of_x = i
y = i;

function y = circle(a0, b0, c0, epsi)

% All the points need to be inside a circle of radius epsi
j = sqrt(-1); origine = a0+j*b0;
% Design of the box and its intersections
box = []; index = 0; factor = 3; for x = 1 : (11*factor)
    for y = 1 : (11*factor)
        index = index + 1;
        box(x,y) = ((a0-epsi)+(x-1)*epsi/(5*factor))
        +j*((b0-epsi)+(y-1)*epsi/(5*factor));
        real_part(index) = (a0-epsi)+(x-1)*epsi/(5*factor);
        imaginary_part(index) = (b0-epsi)+(y-1)*epsi/(5*factor);
    end
end
%plot(box,'*');

% Taking only the points that are within the circle of radius epsi
A = real_part'; B = imaginary_part';

Set = []; BB = []; AA = []; for i = 1 : length(B)
    result = (b0-B(i))^2+(a0-A(i))^2;
    if (result - epsi^2 > 0) % fail the test
        B(i) = 0;
        A(i) = 0;
    else
        B(i) = B(i);
```

```
        A(i) = A(i);
    end
end

% Taking care of only the none zero elements
coord_B = []; coord_A = []; for i = 1 : length(B)
    if (B(i) ~= 0 ),
        coordinates = B(i);
        coordinate = A(i);
        coord_B = [coord_B ;coordinates];
        coord_A = [coord_A ;coordinate];
    end
end B = coord_B; A = coord_A;


y = [A B];

% Extended Gaussian Case
% Guillaume Raux
% Spring 2004 - Fall 2005
% PhD Texas A&M University

clear all; clc;
% Nominal values
a0 = 1/2; b0 = 1/4; c0 = 0; s1 = 1/2; s2 = 1/4; epsi = 0.05; alpha =
0.05; delta = 0.001; r = 2;


% Selection of Points
Points = circle(a0 ,b0 ,c0, epsi); Number_of_points =
min(size(Points))*max(size(Points))/2; A = Points(:,1); B =
Points(:,2); C = zeros(1,length(A))';
% Computation of Sigma square nominal
Sigma_square_nom = a0*b0^2*s1^2 + a0^2*b0*s2^2;

% Computations for Sigma square positive
Partial_b_a = -(A-a0).*(epsi-(A-a0).^2).^(1/2); Partial_b2_a =
-2*b0*Partial_b_a - 2*(A-a0); Partial_sigma_a = (B.^2 +
A.*Partial_b2_a)*s1^2 + (2.*A.^2.*B + A.^2.*Partial_b_a)*s2^2;
Sigma_square = A.*B.^2*s1^2 + A.^2.*B*s2^2;

% Computation of the threshold
T = inverse_erf(alpha)*sqrt(Sigma_square_nom);

% Computation of the integrales
Gam1 = gamma(3/r); Gam2 = gamma(1/r); for i = 1 : length(A)
    term1(i) = -T/s1*r*Gam1^.5/(2*A(i)^.5*Gam2^(3/2))
    *exp(-(abs(A(i)*T/s1))^r/(A(i)^(r/2)*Gam2^(r/2)*Gam1^(-r/2)));
    dummy = B(i);
    dummy2 = A(i);
    term2(i) = quadl(@(y) r*Gam1^(.5)/(2*dummy^.5*Gam2^(3/2))
```

```
    .*exp(-y.^r./(dummy^(r/2)*Gam2^(r/2)*Gam1^(-r/2))), 0, 100);
    term3(i) = @(x) -r*Gam1^(.5)/(4*Gam2^(3/2)*dummy2)
    .*exp(-x.^r./(dummy2^(r/2)*Gam2^(r/2)*Gam1^(-r/2)))
    + r/(2*dummy2).*x.^r./(dummy2^(r/2)*Gam2^(r/2)*Gam1^(-r/2))
    .*exp(-x.^r./(dummy2^(r/2)*Gam2^(r/2)*Gam1^(-r/2)));
end Partial_b_a = -(A-a0).*(epsi-(A-a0).^2).^(1/2); for i = 1 :
length(A)
    % Matrix G and Joestar
    g11 = 1 + Partial_b_a(i)^2;
    g21 = 0;
    g12 = 0;
    g22 = 1;
    G = [g11 g12; g21 g22];
    Joestar = [funky_a(i) funky_r(i)];
    % Slope Unbiased case
    Slope_unbiased(i) = sqrt(Joestar*inv(G)*Joestar');
end
% Assigning the according weight to the slope
xxx = -.05:0.001:0.5; figure; hist(Slope_unbiased,xxx);
%axis([0 0.5 0 50]);
% Saving the results
save('Results_case_1');

function y = inverse_erf(alpha) delta = 0.00001; sigmasquare = 1;
err = 1; i = 0;
%alpha = 0.1;
input = 0.5 - alpha; while (err > 0.005)
    i = i + .05;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
% In case I went too far
i = i - 0.05; err = 1;
% Same procedure with smaller increment and smaller err
while (err > 0.0001)
    i = i + 0.001;
    u = 0:delta:i;
    funky = 1/sqrt(2*pi).*sum(exp(-u.^2/(2*sigmasquare)).*delta);
    err = abs(funky-input);
end
%error_function_equals_to = input
%corresponding_value_of_x = i
y = i;

function y = Integr(lower ,higher, delta) for x = lower : delta :
100
    index = index + 1;
    aa(index) = quadl(@(y) cst2*exp(-abs(y).^r/cst22),x,higher);
    bb(index) = cst1*exp(-abs(x).^r/cst11);
end
```

```
Final = sum(bb.*aa)*delta; y = Final;

function y = circle(a0, b0, c0, epsi)

% All the points need to be inside a circle of radius epsi
j = sqrt(-1); origine = a0+j*b0;
% Design of the box and its intersections
box = []; index = 0; factor = 3; for x = 1 : (11*factor)
    for y = 1 : (11*factor)
        index = index + 1;
        box(x,y) = ((a0-epsi)+(x-1)*epsi/(5*factor))
        +j*((b0-epsi)+(y-1)*epsi/(5*factor));
        real_part(index) = (a0-epsi)+(x-1)*epsi/(5*factor);
        imaginary_part(index) = (b0-epsi)+(y-1)*epsi/(5*factor);
    end
end
%plot(box,'*');

% Taking only the points that are within the circle of radius epsi
A = real_part'; B = imaginary_part';

Set = []; BB = []; AA = []; for i = 1 : length(B)
    result = (b0-B(i))^2+(a0-A(i))^2;
    if (result - epsi^2 > 0) % fail the test
        B(i) = 0;
        A(i) = 0;
    else
        B(i) = B(i);
        A(i) = A(i);
    end
end

% Taking care of only the none zero elements
coord_B = []; coord_A = []; for i = 1 : length(B)
    if (B(i) ~= 0 ),
        coordinates = B(i);
        coordinate = A(i);
        coord_B = [coord_B ;coordinates];
        coord_A = [coord_A ;coordinate];
    end
end B = coord_B; A = coord_A;


y = [A B];
```

APPENDIX G

INTEGRAL FOR THE GENERALIZED GAUSSIAN CASE USING FORTRAN 90

```
PROGRAM integrale

   IMPLICIT NONE

!--------------PARAMETRES DU PROBLEME------------------------------
REAL(KIND=8), DIMENSION(:), ALLOCATABLE  :: a_0, b_0
REAL(KIND=8)::s1, s2, T, r0
   INTEGER  :: i, ix1, ix2
   INTEGER  :: nb_pts    !nb de couples
   REAL(KIND=8)  :: inf       !borne infini
!--------------PARAMETRE POUR L'INTEGRATION DES DERIVEES
PARTIELLES------
   REAL(KIND=8) :: b_inf1, b_sup1  !pour IX1_1,
   IX1_2, IX1_3
   REAL(KIND=8) :: b_inf2, b_sup2  !pour IX2_1,
   IX2_2
   REAL(KIND=8) :: b_inf3, b_sup3  !pour IX2_3

   INTEGER  :: N1, N2
   !nb de pts de discretisation
   REAL(KIND=8) :: h1, h2
   !pas d'espace = (b_sup-b_inf)/N
   REAL(KIND=8) :: Itotal, It1, It2
   REAL(KIND=8) :: IX1_1, IX1_2, IX1_3
   REAL(KIND=8) :: IX2_1, IX2_2, IX2_3
   REAL(KIND=8) :: X1, X2
   LOGICAL      :: once1 = .true., once2 = .true.,
   once = .true.
   REAL(KIND=8) :: user_time, dummy, tt1, tps
   REAL(KIND=8)                                  :: norme
!--------------PARAMETER POUR L'INTEGRATION DE C ET
D----------------
   REAL(KIND=8) :: b_infc, b_supc, t0
   INTEGER      :: N
   REAL(KIND=8) :: h
   REAL(KIND=8) :: C0, D0, E0, F0
   REAL(KIND=8) :: IX2R_1, IX2R_2
   REAL(KIND=8) :: IX1R_1, IX1R_2
   REAL(KIND=8) :: ItotalR
!--------------FIN DES
DECLARATIONS----------------------------------
!=====================================================================
```

```
!---------------LECTURE
PARAMETRES----------------------------------
  write(*,*) 'Lecture data ...'
  OPEN(UNIT = 10, FILE = 'data',FORM ='formatted',STATUS ='unknown')
  READ(10,*) s1, s2, T, nb_pts, inf, r0
!---------------LECTURE DATA
INTEGRATION---------------------------
  READ(10,*) N1, N2
  READ(10,*) b_infc, b_supc, N
  CLOSE(10)
!---------------LECTURE NUAGE DE
POINTS------------------------------
 write(*,*) 'Lecture nuage de points ...'
 ALLOCATE(a_0(nb_pts), b_0(nb_pts))
 OPEN(UNIT = 11, FILE = 'points',FORM ='formatted',STATUS ='unknown')
 DO i=1, nb_pts
    READ(11,*) a_0(i), b_0(i)
 ENDDO
 CLOSE(11)
!---------------OUVERTURE FICHIERS
SORTIE------------------------------
 OPEN(UNIT = 12,FILE='sortie_A',FORM ='formatted',STATUS='unknown')
 OPEN(UNIT = 17, FILE='sortie_R',FORM ='formatted',STATUS='unknown')
 OPEN(UNIT = 13, FILE='fX1_fX2',FORM='formatted',STATUS='unknown')
 OPEN(UNIT = 14, FILE='dfX1_dFX2',FORM='formatted',STATUS='unknown')
 OPEN(UNIT = 15, FILE='gamma',FORM='formatted',STATUS='unknown')
 OPEN(UNIT = 16, FILE='CD',FORM='formatted',STATUS='unknown')
!---------------------------------------------------------------------
!=====================================================================

   !trace des fonctions sur (a,b)
   write(*,*) 'Trace des fonctions et derivees ...'
   DO i=1, N1
      h1 = (inf)/N1
      X1 = h1*i
      write(13,103) X1, f_X1(a_0(1), b_0(1), r0, X1), f_X2(a_0(1),
      b_0(1), r0, X1)
      write(14,103) X1, df_X1(a_0(1), b_0(1), r0, X1), df_X2(a_0(1),
      b_0(1), r0, X1)
      write(15,103) X1, gamma(X1)
   ENDDO
   DO i=1, N
      h1 = (b_supc-b_infc)/N
      X1 = h1*i
      write(16,103) X1, X1**(3.d0/r0-1.d0)*log(X1)*exp(-X1), &
           X1**(1.d0/r0-1.d0)*log(X1)*exp(-X1)
   ENDDO
   CLOSE(13)
   CLOSE(14)
   CLOSE(15)
   CLOSE(16)
```

```
   !STOP
!------------------Calcul des integrales C et D-----------------
   write(*,*) 'Calcul integrales C et D ...'
   DO ix1=1, N

      h = (b_supc-b_infc)/N
      t0 = b_infc + (2.d0*ix1-1)*h/2.d0
      C0 = C0 + h*(-3.d0/r0**2)*t0**(3.d0/r0-1.d0)*log(t0)*exp(-t0)
      D0 = D0 + h*(-1.d0/r0**2)*t0**(1.d0/r0-1.d0)*log(t0)*exp(-t0)

   ENDDO
!-----------------------------------------------------------------
!------------------Calcul de E et F-----------------------------
    write(*,*) 'Calcul constante E et F '
   E0 = 1.d0/(2.d0*r0)*gamma(1.d0/r0)**(-r0/2.d0-1.d0)*D0 -&
        0.5d0*gamma(1.d0/r0)**(-r0/2.d0)*log(gamma(1.d0/r0))

   F0 = -3.d0/(2.d0*r0)*gamma(3.d0/r0)**(-r0/2.d0-1.d0)*C0 + &
        0.5d0*gamma(3.d0/r0)**(r0/2.d0)*log(gamma(3.d0/r0))
!-----------------------------------------------------------------

   write(*,*) 'Calcul integrales ...'

   DO i=1, nb_pts         !boucle sur les couples de points

      tt1 = user_time(dummy)

      write(*,*) 'point numero',i

      b_inf1 = a_0(i)*T/s1
      b_sup1 = inf

      IX1_1  = 0.d0
      IX1_2  = 0.d0
      IX1_3  = 0.d0
      IX2_3  = 0.d0
      IX1R_1 = 0.d0
      IX1R_2 = 0.d0

      DO ix1 = 1, N1          !boucle pour l'integration sur X1

         h1 = (b_sup1-b_inf1)/N1

         IF (once1) THEN
            once1 = .false.
            write(*,*) 'h1=',h1
         ENDIF

         X1 = b_inf1 + (2.d0*ix1-1)*h1/2.d0

         IX2_1  = 0.d0
```

```
IX2_2  = 0.d0
    IX2R_1 = 0.d0
    IX2R_2 = 0.d0

    DO ix2 = 1, N2       !boucle pour l'integration sur X2

       b_inf2 = b_0(i)*((T-(s1*X1)/a_0(i))/s2)
       b_sup2 = inf
       h2 = (b_sup2-b_inf2)/N2

       IF (once2) THEN
      once2 = .false.
          write(*,*) 'h2 = ',h2
       ENDIF

       X2 = b_inf2 + (2.d0*ix2-1)*h2/2.d0

       !D(alpha)/Da
       IX2_1 = IX2_1 + h2*f_X2 (a_0(i), b_0(i), r0, X2)
       IX2_2 = IX2_2 + h2*df_X2(a_0(i), b_0(i), r0, X2)

       !D(alpha)/Dr
       IX2R_1 = IX2R_1 + h2*        &
            J(b_0(i), r0, X2, C0, D0, E0, F0)
       IX2R_2 = IX2R_2 + h2*        &
            J(a_0(i), r0, X2, C0, D0, E0, F0)

   IF (ix1 == 1) THEN
      b_inf2 = 0.d0
          b_sup2 = inf
          h2 = (b_sup2-b_inf2)/N2
          X2 = b_inf2 + (2.d0*ix2-1)*h2/2.d0

          !D(alpha)/Da
      IX2_3 = IX2_3 - T/s1*f_X1(a_0(i),b_0(i),r0,a_0(i)*T/s1)*&
              h2*f_X2(a_0(i), b_0(i), r0, X2)
       ENDIF

       !D(alpha)/Da
       IX1_1 = IX1_1 + h1*df_X1(a_0(i), b_0(i), r0, X1)*IX2_1
       IX1_2 = IX1_2 + h1*f_X1 (a_0(i), b_0(i), r0, X1)*IX2_2
       IX1_3 = IX1_3 - h1*b_0(i)*X1/(s2*a_0(i)**2) * &
           f_X1(a_0(i), b_0(i), r0, X1)*             &
           f_X2(a_0(i), b_0(i), r0, ((T-(s1*X1)/a_0(i))/s2) )

       !D(alpha)/Dr
       IX1R_1 = IX1R_1 + h1*f_X1(a_0(i), b_0(i), r0, X1)*IX2R_1
       IX1R_2 = IX1R_2 + h1*f_X2(a_0(i), b_0(i), r0, X1)*IX2R_2

    ENDDO
```

```
         Itotal = IX1_1 + IX1_2 + IX1_3 + IX2_3
         ItotalR = IX1R_1 + IX1R_2

      ENDDO

      IF (once) THEN
         once = .false.
     tps = user_time(dummy) - tt1
         write(*,*) 'estimation temps total :', tps*nb_pts/60.d0
      ENDIF

      write(12,103) a_0(i), b_0(i), Itotal
      write(17,103) a_0(i), b_0(i), ItotalR


   ENDDO

   CLOSE(12)

103 FORMAT(50(e22.9,2x))

CONTAINS

FUNCTION f_test(x) RESULT(valeur)

   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: x
   REAL(KIND=8)               :: valeur

   valeur = x**5

END FUNCTION f_test

FUNCTION f_X1(a, b, r, x) RESULT(valeur)

   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: a, b, r, x
   REAL(KIND=8)               :: valeur

   valeur = (r*gamma(3.d0/r)**(0.5)) /         &
    (2.d0*a**(0.5d0)*gamma(1.d0/r)**(1.5d0)) * &
        exp(                                   &
        -abs(x)**(r) /                         &
        (a**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0)))

END FUNCTION f_X1


FUNCTION f_X2(a, b, r, x) RESULT(valeur)
```

```
   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: a, b, r, x
   REAL(KIND=8)               :: valeur

   valeur = (r*gamma(3.d0/r)**(0.5)) /         &
        (2.d0*b**(0.5)*gamma(1.d0/r)**(1.5)) * &
        exp(                                   &
        -abs(x)**(r) /                         &
        (b**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0)))

END FUNCTION f_X2

FUNCTION df_X1(a, b, r, x) RESULT(valeur)

   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: a, b, r, x
   REAL(KIND=8)               :: valeur
   REAL(KIND=8)               :: AA, BB

   AA = -r*gamma(3.d0/r)**(0.5) /  &
    (4.d0*gamma(1.d0/r)**(1.5)*a**(1.5))

   BB = 0.5d0 * abs(x)**r /                      &
        (a**(r/2.d0)*gamma(1.d0/r)**(r/2.d0)*  &
    gamma(3.d0/r)**(-r/2.d0)) * r/a *      &
    exp(                                   &
    -abs(x)**(r) /                         &
    (a**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0)))

   valeur = AA * exp(                          &
        -abs(x)**(r) /                         &
        (a**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0))) +           &
    BB * (r*gamma(3.d0/r)**(0.5d0)) /       &
    (2.d0*a**(0.5)*gamma(1.d0/r)**(1.5))


END FUNCTION df_X1

FUNCTION df_X2(a, b, r, x) RESULT(valeur)

   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: a, b, r, x
   REAL(KIND=8)               :: valeur
   REAL(KIND=8)               :: AA, BB

   AA = -r*gamma(3.d0/r)**(0.5) /  &
        (4.d0*gamma(1.d0/r)**(1.5)*b**(1.5))
```

```
   BB = 0.5d0 * abs(x)**r /                          &
        (b**(r/2.d0)*gamma(1.d0/r)**(r/2.d0)*  &
        gamma(3.d0/r)**(-r/2.d0)) * r/b *      &
        exp(                                   &
        -abs(x)**(r) /                         &
        (b**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0)))

   valeur = AA * exp(                              &
        -abs(x)**(r) /                             &
        (b**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0))) +           &
        BB * (r*gamma(3.d0/r)**(0.5d0)) /       &
        (2.d0*b**(0.5)*gamma(1.d0/r)**(1.5))


END FUNCTION df_X2

FUNCTION J(a, r, x, C, D, E, F) RESULT(valeur)

   IMPLICIT NONE
   REAL(KIND=8), INTENT(IN)   :: a, r, x
   REAL(KIND=8), INTENT(IN)   :: C, D, E, F
   REAL(KIND=8)               :: valeur
   REAL(KIND=8)               :: K, I, H


   H = -0.5d0*a**(-r/2.d0)*log(a)

   K = 1.d0/(2.d0*a**(0.5))*   (                  &
        gamma(1.d0/r)**(-1.5)* (gamma(3.d0/r)**(0.5) + &
        r/2.d0*C*gamma(3.d0/r)**(-0.5)) +      &
        r*gamma(3.d0/r)**(0.5)*-1.5*           &
        gamma(1.d0/r)**(-2.5)*D)

   I = -abs(x)**(r)*a**(-r/2.d0)*gamma(1.d0/r)**(-r/2.d0)*F &
        -abs(x)**(r)*a**(-r/2.d0)*E*gamma(3.d0/r)**(r/2.d0) &
        - abs(x)**(r)*H*gamma(1.d0/r)**(-r/2.d0)*gamma(3.d0/r)
        **(r/2.d0) &-G(r, x)*a**(-r/2.d0)*gamma(1.d0/r)**(-r/2.d0)
        *gamma(3.d0/r)**(r/2.d0)

   valeur = exp(-abs(x)**(r) /                    &
        (a**(r/2.d0)*gamma(1.d0/r)**(r/2.d0) * &
        gamma(3.d0/r)**(-r/2.d0)))*(           &
        K+I*r*gamma(3.d0/r)**(0.5)/(           &
        gamma(1.d0/r)**(1.5)*2.d0*a**(0.5)))

END FUNCTION J

FUNCTION G(r, x) RESULT(valeur)
```

```
   IMPLICIT NONE
   REAL(KIND=8),      INTENT(IN) :: r, x
   REAL(KIND=8)                  :: valeur

   valeur = abs(x)**(r)*log(abs(x))

END FUNCTION G


function gamma(x) result(dgamma)

      implicit none
      real(kind=8), intent(in) :: x
      real(kind=8)             :: dgamma
      integer                  :: n, k
      real(kind=8)             :: w, y

      real(kind=8), parameter ::   p0 = 0.99999999999999990d0
      real(kind=8), parameter ::   p1 = -0.42278433509846784d0
      real(kind=8), parameter ::   p2 = -0.23309373642178287 8d0
      real(kind=8), parameter ::   p3 = 0.19109110138763841 0d0
      real(kind=8), parameter ::   p4 = -0.02455249000564127 8d0
      real(kind=8), parameter ::   p5 = -0.01764524454785141 4d0
      real(kind=8), parameter ::   p6 = 0.00802327302785534 6d0
      real(kind=8), parameter ::   p7 = -0.00080432981925574 4d0
      real(kind=8), parameter ::   p8 = -0.00036083787664825 5d0
      real(kind=8), parameter ::   p9 = 0.00014559656861752 6d0
      real(kind=8), parameter ::   p10 = -0.00001754553939520 5d0
      real(kind=8), parameter ::   p11 = -0.00000259122526768 9d0
      real(kind=8), parameter ::   p12 = 0.00000133776738406 7d0
      real(kind=8), parameter ::   p13 = -0.00000019954286367 4d0

      n = nint(x - 2)
      w = x - (n + 2.d0)
      y = ((((((((((((p13 * w + p12) * w + p11) * w + p10) * &
         w + p9) * w + p8) * w + p7) * w + p6) * w + p5) *  &
         w + p4) * w + p3) * w + p2) * w + p1) * w + p0
      if (n .gt. 0) then
          w = x - 1
          do k = 2, n
              w = w * (x - k)
          end do
      else
          w = 1
          do k = 0, -n - 1
              y = y * (x + k)
          end do
      end if
      dgamma = w / y
end function gamma
```

```
FUNCTION user_time(dummy) RESULT(time)
   IMPLICIT NONE

      REAL(KIND=8) :: dummy, time

      INTEGER :: count, count_rate, count_max

      CALL SYSTEM_CLOCK(COUNT, COUNT_RATE, COUNT_MAX)
      time = (1.d0*count)/count_rate

END FUNCTION user_time


END PROGRAM integrale
```

VITA

Guillaume Raux was born in Dijon, France on January 3, 1976. He is the son of Michèle Blanche Raux and Michel Georges Raux, the brother of Emmanuel Yann and the uncle of Anne-Sophie and Camille, from France. He graduated from L'Universit*é* de Rennes 1 with a Licence d'Ingenierie Electrique in May 1998. He then attended West Virginia University where he played varsity tennis and obtained both a B.S. in 2001 and a M.S. in 2003, both in Electrical Engineering. After that, Guillaume attended Texas A&M University in 2003 for his Ph.D. He studied ways to quantify measures of robustness for signal detection under the supervision of Dr. Don Halverson. He obtained his diploma in December 2006. His research interests are in wireless communication systems, estimation and/or detection of signals, and spread spectrum.

Permanent address:

3400 Clayton Blvd Apt#6B

Shaker Heights, OH 44120

The typist for this dissertation was Guillaume Julien Raux.