ROBUST HUBER-BASED OPTIMIZATION FOR CIRCUIT DESIGN, MODELING AND DIAGNOSIS

ROBUST HUBER-BASED OPTIMIZATION FOR CIRCUIT DESIGN, MODELING AND DIAGNOSIS

H. Yu

SOS-93-18-T

October 1993

ROBUST HUBER-BASED OPTIMIZATION FOR CIRCUIT DESIGN, MODELING AND DIAGNOSIS

By

HUANYU YU, B. Eng.
(Tsinghua University)

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Engineering

McMaster University

October 1993

MASTER OF ENGINEERING (1993)
(Electrical and Computer Engineering)

McMASTER UNIVERSITY
Hamilton, Ontario

TITLE: **Robust Huber-Based Optimization for Circuit Design, Modeling and Diagnosis**

AUTHOR: Huanyu Yu
B. Eng.
(Tsinghua University)

SUPERVISOR: J. W. Bandler
Professor, Department of Electrical and Computer Engineering
B.Sc.(Eng.), Ph.D., D.Sc.(Eng.) (University of London)
D.I.C. (Imperial College)
P.Eng. (Province of Ontario)
C.Eng. F.I.E.E. (United Kingdom)
Fellow, I.E.E.E.
Fellow, Royal Society of Canada

NUMBER OF PAGES: xii, 149

ABSTRACT

This thesis addresses robust Huber-based optimization for circuit design, modeling and diagnosis.

Optimization problems using Huber functions are formulated. First- and second-order sensitivities of the Huber objective functions are investigated. The robustness of the Huber approach is explored. One-sided Huber-based optimization is introduced for engineering design problems.

Dedicated algorithms for Huber-based optimization are reviewed. The appropriateness and efficiency of the algorithms are demonstrated.

Robust statistical modeling is addressed. Statistical parameter extraction and postprocessing are used to obtain the sample of models and estimate parameter statistics. The advantage of the Huber statistical estimator is illustrated. Suitable threshold selection is discussed.

For the first time, the analog fault isolation problem is solved by Huber-based optimization. The effectiveness of the Huber approach is demonstrated by both theoretical formulation and a real example.

The Huber concept is applied to large-scale design problems. The one-sided Huber preparatory optimization is shown to be more effective than minimax method in terms of providing a good starting point for full-scale design optimization.

ACKNOWLEDGEMENTS

The author wishes to express his sincere appreciation to Dr. J.W. Bandler for his encouragement, expert guidance and supervision throughout the course of this work.

The author appreciates the opportunity given by Dr. Bandler to be involved in industrially relevant projects and academic research activities. Specially, the author has greatly benefitted from participation in the development and application of the OSA90/hope™ microwave CAD system developed by Optimization Systems Associates Inc., Dundas, Ontario, Canada.

The author offers his deep gratitude to Dr. R.M. Biernacki and Dr. S.H. Chen, of McMaster University and Optimization Systems Associates Inc., for their expert advice, through frequent and very beneficial discussions. Their excellent work and valuable suggestions have helped the author to develop, improve and test new ideas.

The author would like to express his appreciation to Dr. K. Madsen of the Institute for Numerical Analysis, The Technical University of Denmark, Lyngby, Denmark, for introducing Huber-based optimization to the author, and for helpful discussions while he was visiting McMaster University.

It is the author's pleasure to have been associated with Dr. Q. Cai of the Simulation Optimization Systems Research Laboratory, McMaster University, and Dr. S. Ye, now with ComDev Ltd., Cambridge, Ontario, Canada. Their useful suggestions and some of their original work have been very helpful in the author's research. The author acknowledges inspiring discussions with his colleague P.A. Grobelny.

The financial assistance provided by the Natural Sciences and Engineering Research Council of Canada through Grants OGP0007239, STR0042444, and STR0117819, the Department of Electrical and Computer Engineering through a Teaching Assistantship and Scholarship, and McMaster University through a Barkley's of Avonmore Bursary is gratefully acknowledged.

Finally, thanks are due to the author's family for encouragement, understanding, patience and continuous support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Optimization-oriented computer-aided circuit design (CAD) has been an active area of computer applications for more than two decades (Bandler and Chen (1988) [1]). The classic paper by Temes and Calahan in 1967 [2] was one of the earliest to formally advocate the use of iterative optimization in circuit design.

Since its inception, computer-aided design optimization has been experiencing significant advances made by engineers and scientists. Techniques for efficient large-scale network simulation and optimization have been developed due to the emergence of the VLSI era. Optimization methods have evolved from primitive and low-dimension-oriented algorithms to sophisticated and powerful ones.

Plainly speaking, the aim of circuit optimization is to select appropriate circuit (model) parameter values so that the circuit best satisfies the design requirements. Closely related is the modeling problem, where parameters are to be determined in accordance with measurements/experimental data.

Nominal design and modeling is the conventional approach adopted by many circuit designers. It concentrates on satisfying design requirements by the responses of a single circuit or circuit model of interest. Unfortunately, practical realization of a nominal design is inevitably subject to fluctuation and/or deviation due to manufacturing uncertainties, parameter tolerances and model inaccuracy, etc.

The focus of circuit design is shifting from improving individual circuit responses to estimation and enhancement of yield of production by taking various uncertainties into consideration. To be distinguished from the traditional nominal design, the new approach is called yield/cost-driven design (Bandler and Abdel-Malek (1977) [3], Abdel-Malek and Bandler (1980) [4,5]). The objective is the minimization of cost, the maximization of production yield, or a combination of both.

Circuit design problems can be considered as optimization problems at different levels of the hierarchical simulation model (Bandler, El-Kady, Kellermann and Zuberek [6] and Tromp [7] and [8]). Therefore, unified, integrated approaches have been developed to attempt the problems [1], either nominal or statistical.

The classic least-squares (ℓ_2) measure (see, for example, Morrison [9]) has been religiously accepted as the criterion for the matching between the simulated responses (and/or outcomes) and the specifications. The ℓ_1 approach has been strongly advocated to overcome large errors [10,11]. Another widely advertised method is minimax optimization (for example, Madsen, Schjaer-Jacobsen and Voldby (1975) [12]). These optimization objectives are generalized as ℓ_p objectives (Bandler and Chen [1]).

However, the use of the ℓ_2 and minimax optimization methods is considerably limited by the presence of gross errors originating from measurements, faulty elements and/or devices or process uncertainties, etc. The ℓ_2 solution is inherently affected by "bad data points" (Bandler, Chen, Biernacki, Madsen, Li Gao and Yu [13]). Minimax optimization emphasizes the biggest errors and, therefore, is least suitable for data fitting. The ℓ_1 method, though robust against large errors, is not suitable for modeling small variations in the data.

Huber (1964) [14] proposed a function which he considered is the most robust for statistics. Ekblom and Madsen [15] and Gao Li and Madsen [16] provided an algorithm, the Ekblom Madsen algorithm (EMA), and the corresponding computer implementation, respectively, for robust nonlinear optimization using the "Huber function". The work of Bandler, Chen, Biernacki, Madsen, Li Gao and Yu [13] first introduced the robust Huber concept to the engineering field.

This thesis addresses a novel approach to "robustizing" circuit optimization using Huber functions: both two-sided and one-sided [13-18]. Advantages of the Huber functions for optimization in the presence of faults, large and small measurement errors, bad starting points and statistical uncertainties are described. In its context, comparisons are made with optimization using traditional ℓ_1, ℓ_2 and minimax objective functions. The gradients and Hessians of the Huber objective functions are formulated and compared with those in the least-squares case. We review a dedicated, efficient algorithm for minimizing Huber objective functions and show, by comparison, that generic optimization methods are not adequate for Huber-based optimization. A wide range of significant applications is illustrated, including FET statistical modeling, multiplexer optimization, analog fault location and data fitting.

Throughout this thesis, unless otherwise indicated, the terms "Huber optimization" and "Huber solution" refer to the optimization using the dedicated EMA and the solutions obtained by this algorithm, respectively. When the problem is discussed on a more general basis, for example, when a problem is solved by formulating the Huber penalty terms into the objective function of a generic optimizer, it is referred to as "Huber-based optimization" or "Huber-based solution".

The robustness of the Huber-based optimization is exhibited in Chapter 2 through theoretical formulations. The gradients and Hessians of the Huber objective functions are derived. Comparison with the gradient and Hessian of the ℓ_2 objective is made to further the insight into the properties of the Huber formulation. The performance of Huber optimization is compared with ℓ_1 and ℓ_2 optimization through a data fitting example with gross errors. A "one-sided" Huber function is also formulated in this chapter for engineering design problems.

Chapter 3 gives a detailed review of the dedicated EMA. Two stages of the algorithm, namely a trust-region method and a quasi-Newton method, are described. Comparison is made between the trust-region method and conventional Newton method. Switching criteria between the two stages are addressed. The dedicated EMA is compared with generic methods in a number of numerical examples in Chapters 3 and 4.

Chapters 4 to 6 address three application areas of circuit design and modeling. We show, through examples, that the utilization of Huber-based optimization is far more advantageous than traditional optimization methods in these areas.

Robust statistical modeling is explored in Chapter 4. The applicability of the Huber-based optimization is exposed through the description of statistical parameter extraction and estimation. The robustness of the Huber statistical estimator is illustrated by a MESFET example. Appropriate selection of the threshold value is also addressed.

Analog fault diagnosis by Huber-based optimization is presented in Chapter 5. The Huber concept is integrated into the problem formulation. A resistive mesh network example is provided to illustrate the effectiveness of the Huber method in fault isolation, in comparison with the ℓ_1 method. The impact of different starting points for optimization is investigated.

Preprocessing of large-scale microwave multiplexer design optimization is addressed in Chapter 6. A comparison between "one-sided" Huber optimization and minimax optimization is made in terms of providing a good starting point for subsequent full-scale optimization. A 5-channel microwave multiplexer design problem is presented. The effectiveness and efficiency of the one-sided Huber optimization is demonstrated.

The numerical examples demonstrated in this thesis have been prepared using the CAE software systems OSA90/hope™ [19] and HarPE™ [20] developed by Optimization Systems Associates, Inc., Dundas, Ontario, Canada.

We conclude this thesis in Chapter 7, with some suggestions for further research.

In this thesis, the author contributed substantially to the following original developments:

1. A new approach of robustizing circuit design using Huber functions.
2. Exposition of the derivative information of the Huber functions in relation to their robustness.

3. Expansion of the one-sided Huber function for design optimization with upper and/or lower specifications.
4. Statistical device modeling using the Huber function as a robust estimator.
5. Integration of the Huber concept into the formulation of analog fault diagnosis.
6. Preparatory one-sided Huber optimization for large-scale design optimization.

Chapter 2

ROBUST HUBER-BASED OPTIMIZATION – THEORETICAL FORMULATION AND PROPERTIES

2.1 INTRODUCTION

For about three decades, robustness has been an important goal of many mathematicians and statisticians, knowing that real-world data is inevitably degraded by errors. As is well-known, however, the classical least squares method is unduly sensitive to gross errors. In one of the pioneering papers, Huber (1964) [14] proposed an alternative to replace the vulnerable least-squares estimator: the Huber function.

The high sensitivity of the ℓ_2 estimator was attributed by Dutter and Huber (1981) [21] and Shanno and Rocke (1986) [22] to the fact that each item enters the objective quadratically. A consequence of this is that large error functions have a major influence on the results.

The Huber function was devised as a slower growing function. It appears to be a hybrid of the ℓ_1 and ℓ_2 objective functions. As we will demonstrate later, the Huber-based optimization inherits mixed numerical properties from both methods, i.e., the robustness against gross errors as well as the discriminatory power to deal with small, statistical variations. In fact, as pointed out by Shanno and Rocke [22], the Huber function is one of the two most commonly adopted penalty functions for robust statistics. The other penalty function is the so-called biweight function due to Beaton and Tukey (1974) [23].

The recent paper by Bandler, Chen, Biernacki, Madsen, Gao and Yu (1993) [13] first introduced the robust Huber functions to circuit design engineers. Mathematical robustness has, thereby, entered into real and complicated applications.

This chapter is devoted to providing a mathematical understanding of the Huber-based optimization. Definitions of the Huber functions and Huber-based optimization are given following the general formulation for circuit optimization. Investigation of the gradients and Hessians of the Huber objective functions is made to expose the distinctive robust property. A data fitting problem is demonstrated, associating the abstract theory with reality. Emphasis is given to comparing the Huber solution with ℓ_1 and ℓ_2 solutions. One-sided Huber-based optimization is also formulated for design problems with upper and/or lower specifications.

2.2 THEORETICAL FORMULATION

2.2.1 General Formulation for Circuit Optimization

The following formulation is based on Bandler and Chen [1]. More detailed illustrations of specifications and error functions can also be found in Bandler [24] and Bandler and Rizk [25].

The performance of an electrical system is usually expressed by circuit outputs or responses, which are very often functions of independent variables such as time, frequency and temperature. Circuit output or response can be mathematically abstracted as

$$R = R(\boldsymbol{\phi}, \psi) \tag{2-1}$$

where $\boldsymbol{\phi} = [\phi_1 \ \phi_2 \ \dots \ \phi_n]^T$ is the parameter vector of the system and ψ is the independent variable. The superscript T in the above notation stands for vector or matrix transposition.

Circuit design is to select appropriate values for the parameters in $\boldsymbol{\phi}$ so that the circuit responses satisfy design specifications, each of which can be denoted by

$$S = S(\psi) \tag{2-2}$$

Fig. 2.1(a) depicts a scenario of one circuit response with upper and lower specifications. Fig. 2.2(a) illustrates a single specification case.

Error functions arise from the difference between the specifications and the responses. In the single specification case, an error function is defined as

$$e(\boldsymbol{\phi}, \psi) = w|R - S| \tag{2-3}$$

where w is a nonnegative weighting factor. In the case of having upper specifications S_u or lower specifications S_l, error functions are defined as

$$e_u(\boldsymbol{\phi}, \psi) = w_u(R - S_u) \tag{2-4}$$

or

$$e_l(\boldsymbol{\phi}, \psi) = w_l(S_l - R) \tag{2-5}$$

respectively, where w_u and w_l are nonnegative weighting factors. Fig. 2.1(b,c) and Fig. 2.2(b) depict the concept of error functions. It is clearly observed that positive

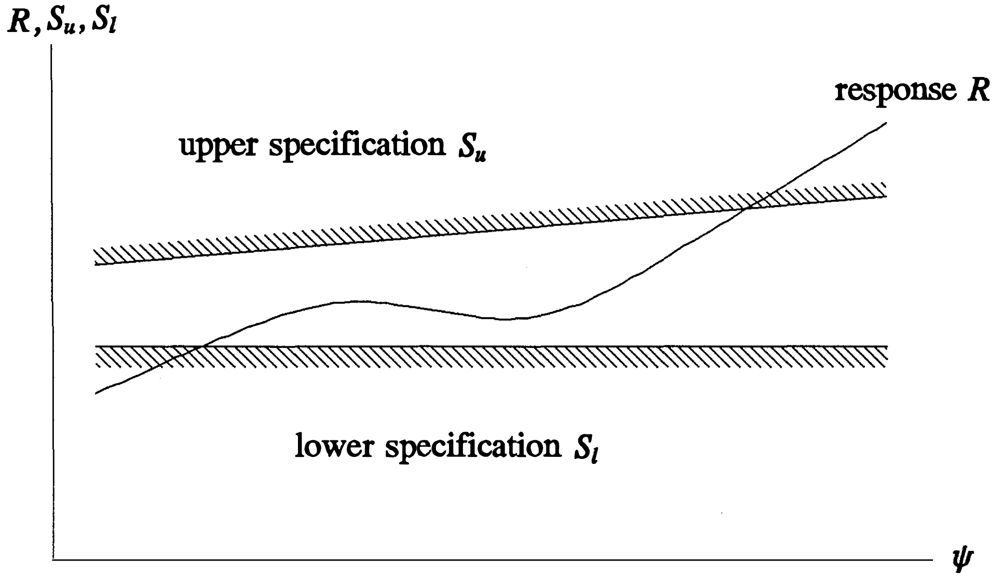

Fig. 2.1(a) Circuit response with upper and lower specifications.

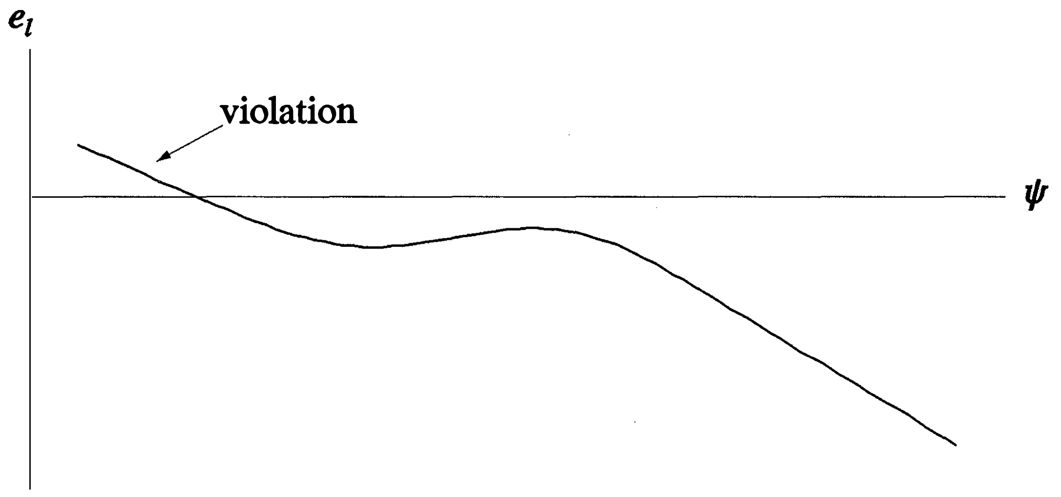

Fig. 2.1(b) Error function with respect to the lower specification.

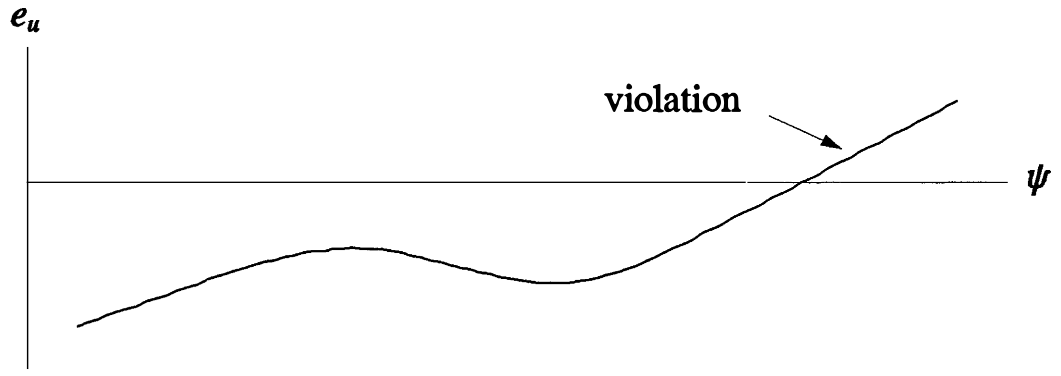

Fig. 2.1(c) Error function with respect to the upper specification.

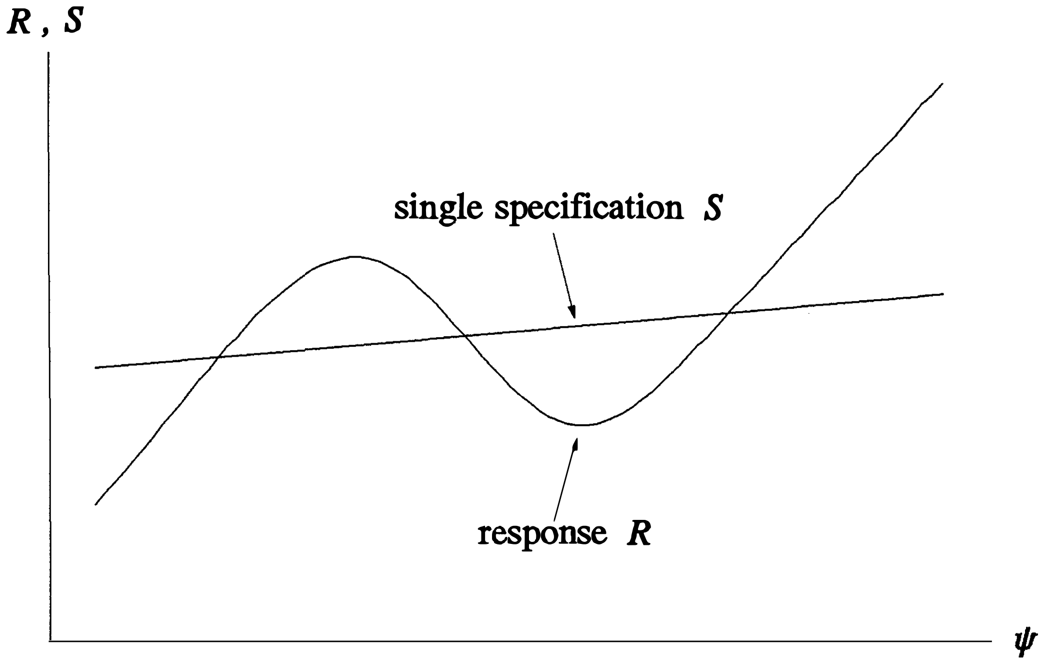

Fig. 2.2(a) Circuit response with a single specification.

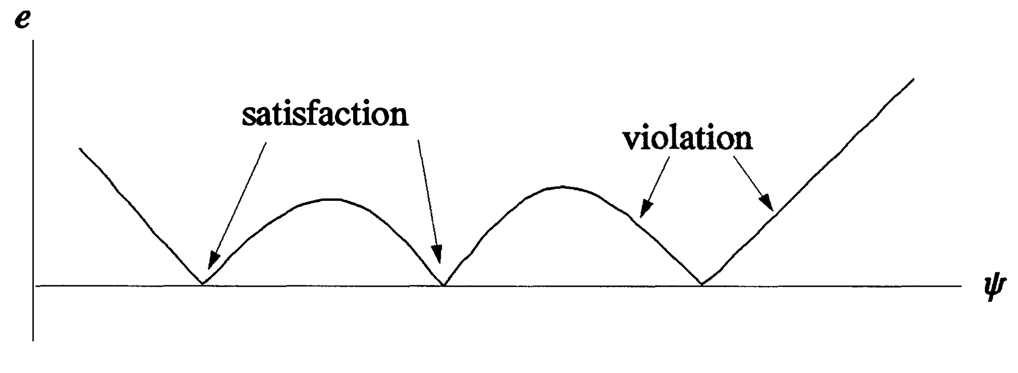

Fig. 2.2(b) Error function with respect to the single specification.

error function values indicate violating the corresponding specification while nonpositive values mean satisfaction.

For computational purposes, the circuit responses, the specifications and the error functions have to be discretized w.r.t. the independent variable ψ. Therefore, instead of a continuous error function $e(\boldsymbol{\phi}, \psi)$, we have a set of sampled error functions $e_j(\boldsymbol{\phi})$, $j = 1, 2, \ldots, m$, where

$$e_j(\boldsymbol{\phi}) = e(\boldsymbol{\phi}, \psi_j), \quad j = 1, 2, \ldots, m \tag{2-6}$$

In general, different weighting factors may be imposed at different sample points of ψ. Figs. 2.3 and 2.4 show the error functions discretized from the corresponding continuous error functions in Figs. 2.1 and 2.2.

Mathematically, a circuit optimization problem can be formulated as

$$\underset{\boldsymbol{x}}{\mathit{minimize}} \quad F(\boldsymbol{x}) \tag{2-7}$$

where $F(\boldsymbol{x})$ is a scalar objective function, composed of error functions, $\boldsymbol{x}$ are optimization variables, which may include the circuit parameters $\boldsymbol{\phi}$.

Temes and Zai (1969) [26] proposed the ℓ_p norm of the error functions $\boldsymbol{e}$ as a candidate for the objective function $F(\boldsymbol{x})$ in (2-7). It has the form

$$\| \boldsymbol{e} \|_p = \left[\sum_{j=1}^{m} |e_j|^p \right]^{1/p} \tag{2-8}$$

Two most well-known members of the ℓ_p optimization family are the least squares (ℓ_2) defined by

$$\underset{\boldsymbol{x}}{\mathit{minimize}} \quad F(\boldsymbol{x}) \triangleq \| \boldsymbol{e} \|_2 = \left[\sum_{j=1}^{m} |e_j(\boldsymbol{x})|^2 \right]^{1/2} \tag{2-9}$$

and the ℓ_1 which solves

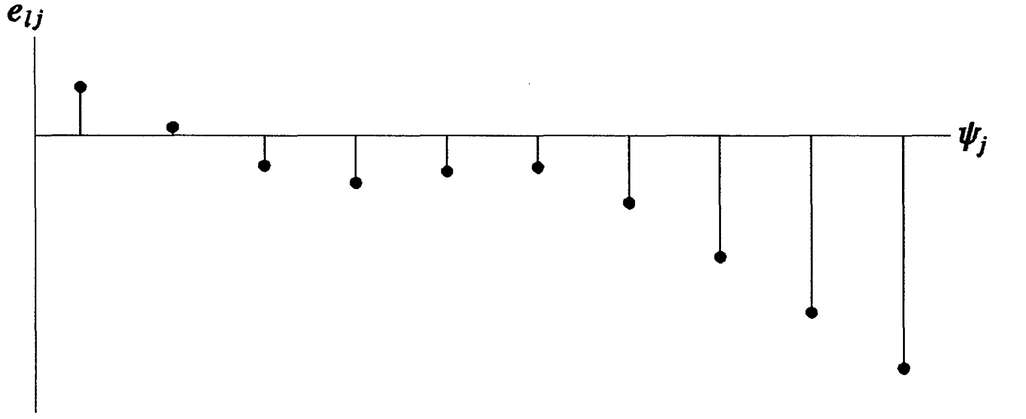

Fig. 2.3(a) Discretized error functions from Fig. 2.1(b).

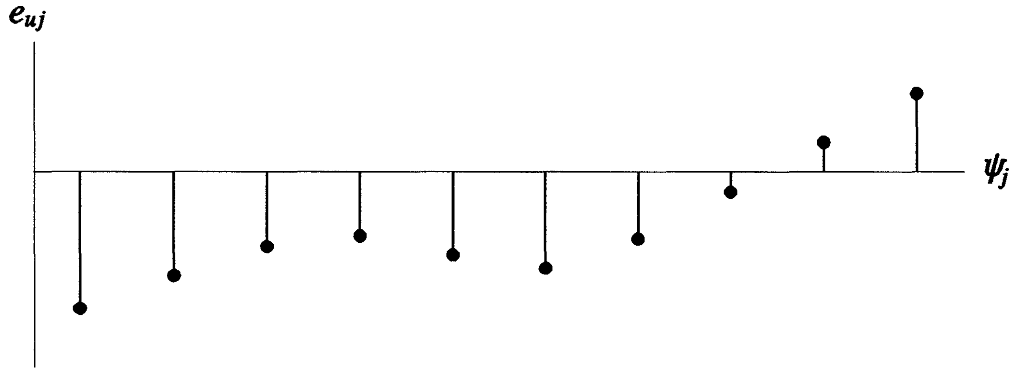

Fig. 2.3(b) Discretized error functions from Fig. 2.1(c).

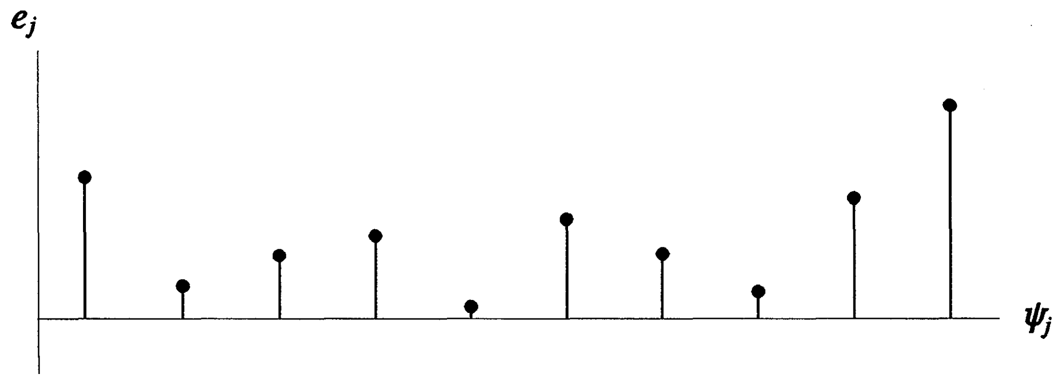

Fig. 2.4. Discretized error functions from Fig. 2.2(b).

$$\underset{x}{\mathit{minimize}} \quad F(x) \triangleq \|e\|_1 = \sum_{j=1}^{m} |e_j(x)| \tag{2-10}$$

Equivalently, assuming real errors, the ℓ_2 optimization problems can be considered as solving

$$\underset{x}{\mathit{minimize}} \quad F(x) \triangleq \sum_{j=1}^{m} e_j^2(x) \tag{2-11}$$

It is clear that the ℓ_2 method penalizes the errors quadratically. As speculated by statisticians (see Dutter and Huber [21], Shanno and Rocke [22]), this rapidity of growth in the penalty function is the source of failure of the ℓ_2 estimation in the presence of gross errors.

On the other hand, if the errors are penalized linearly, as in the ℓ_1 case, the estimator becomes the so-called sample median [14]: the estimator lacks the power to discriminate among small statistical variations and depends only on the order of the data [13].

2.2.2 The Huber Function and Huber-based Optimization

The Huber function, defined as [13-18,21,22]

$$\rho_k(e) \triangleq \begin{cases} e^2/2 & \mathit{if}\ |e| \le k \\ k|e| - k^2/2 & \mathit{if}\ |e| > k \end{cases} \tag{2-12}$$

where k is a positive constant, was devised as an intermediary between the least-squares and the ℓ_1 functions. Fig. 2.5 depicts the Huber function in comparison with the ℓ_1 and ℓ_2 functions.

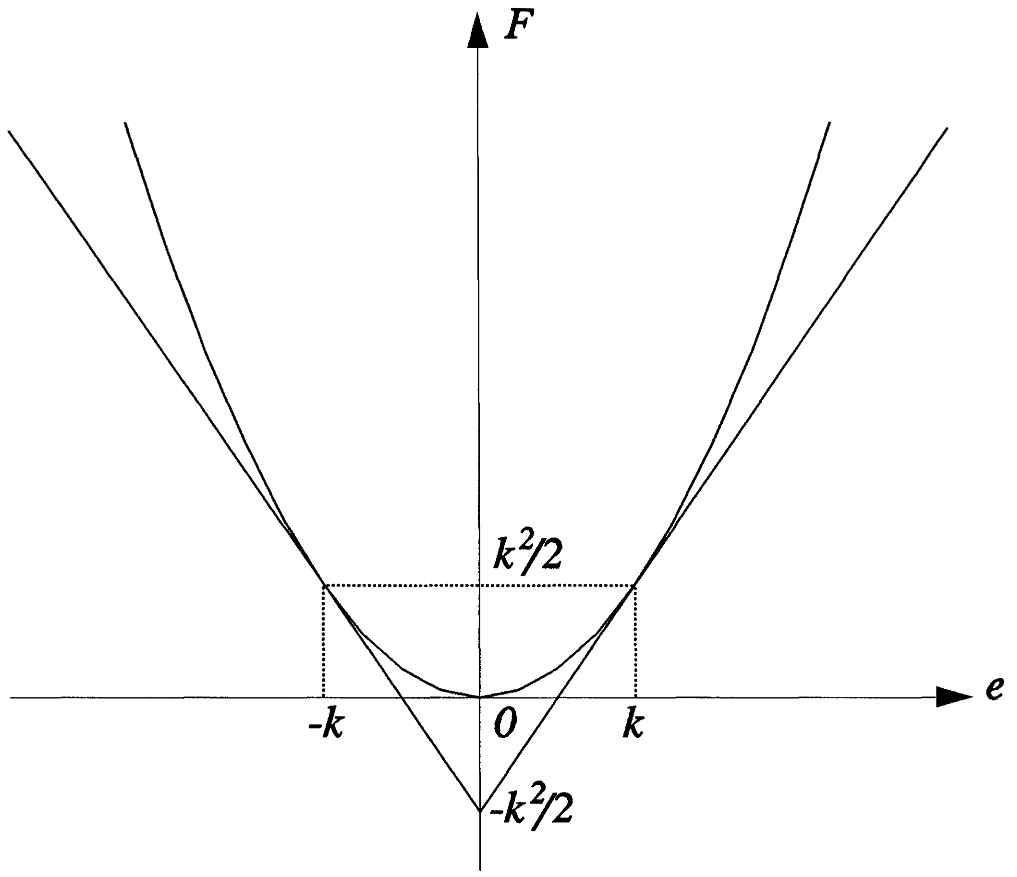

Fig. 2.5(a). The ℓ_1 and ℓ_2 functions. The ℓ_1 function is rescaled and shifted in accordance with the corresponding part in the Huber function. It has the form $F = k|e| - k^2/2$. The ℓ_2 function has the form $F = e^2/2$.

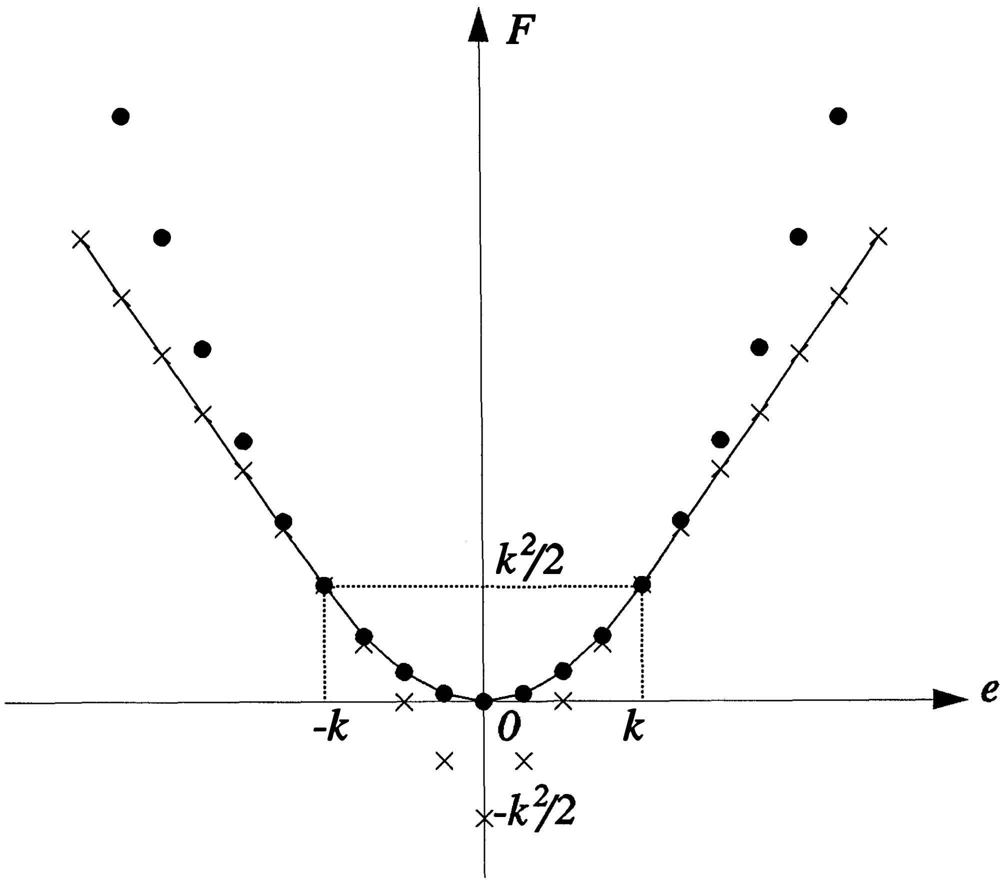

Fig. 2.5(b). The Huber, ℓ_1 and ℓ_2 functions. The strikes and dots represent the discrete points on the ℓ_1 and ℓ_2 curves, respectively, in Fig. 2.5(a). The continuous curve indicates the Huber function.

The corresponding Huber-based optimization problem is then defined as [13,15,16,18]

$$\underset{x}{\textit{minimize}} \quad F(x) \triangleq \sum_{j=1}^{m} \rho_k(e_j(x)) \tag{2-13}$$

As we can see from the figures, the errors are treated quadratically only when they are smaller than the threshold value k ("small errors"); when an error becomes an "outlier", the Huber function increases linearly. This linear growth for the outliers is the key to the robustness of the Huber-based method.

This adequate treatment of errors can be interpreted in a different way. The Huber function ρ_k is a hybrid of the least-squares (ℓ_2) (when $|e| \leq k$) and the ℓ_1 (when $|e| > k$) functions. The ℓ_1 is robust against gross errors [10,11]. Since the Huber function penalizes errors above the threshold (i.e., $|e| > k$) in the ℓ_1 sense, it is robust against those errors, i.e., the solution is not sensitive to those errors. For small errors, the ℓ_2 measure is invoked to acquire the least-squares' distinctive modeling power.

The adjustments in the function definition of ρ_k (e.g., constants 2, $k^2/2$, etc.) ensure a smooth transition between ℓ_2 and ℓ_1 at $|e| = k$. This means that the first derivative of ρ_k w.r.t. e is continuous.

The constant k in (2-12) and (2-13) defines the threshold between "large" and "small" errors. By varying k, we can alter the proportion of error functions to be treated in the ℓ_1 or ℓ_2 sense. Huber gave a look-up table [17] from which k can be determined according to the percentage of gross errors in the data.

Moreover, as Huber [14] indicated, the traditional ℓ_1 and ℓ_2 approaches are two limiting cases of (2-13): if k is set to a sufficiently large value, the optimization problem becomes least squares. On the other hand, as k approaches zero, ρ_k/k will approach the ℓ_1 function.

2.3 GRADIENT INVESTIGATION

In this section, we investigate the derivatives of the Huber function (2-12) and the gradient and Hessian of the Huber objective function (2-13) given by Ekblom and Madsen [15]. This is because such investigation can provide a better understanding from a mathematical point of view. Also, as we will manifest in the next chapter, modern state-of-the-art optimization algorithms employ gradient-based methods.

The first derivative of ρ_k w.r.t. e_j is given by

$$v_j \triangleq \frac{\partial \rho_k(e_j(\boldsymbol{x}))}{\partial e_j(x)} = \begin{cases} e_j(\boldsymbol{x}) & if\ |e_j(\boldsymbol{x})| \leq k \\ k & if\ e_j(\boldsymbol{x}) > k \\ -k & if\ e_j(\boldsymbol{x}) < -k \end{cases} \tag{2-14}$$

The growth of $|v_j|$ is bounded by k, holding constant if the point is an outlier (i.e., when $|e_j| > k$). This boundedness was introduced in an attempt to limit the influence of any gross errors [21]. A consequence of this, as observed by Gao Li and Madsen [16], is that the Huber-based solution is unaffected if some of the outliers are moved even further away.

The gradient vector of the Huber objective function F w.r.t. the optimization variables $\boldsymbol{x}$ is given by

$$\nabla F = \sum_{j=1}^{m} v_j \, e_j' \tag{2-15}$$

where

$$e_j' \triangleq \left[\frac{\partial e_j(x)}{\partial x_1} \; \frac{\partial e_j(x)}{\partial x_2} \; \cdots \; \frac{\partial e_j(x)}{\partial x_n} \right]^T \tag{2-16}$$

The structure of (2-15) is very similar to the gradient of ℓ_2 (least squares), which is

$$\nabla F_{\ell_2} = \sum_{j=1}^{m} e_j \, e_j' \tag{2-17}$$

By comparing (2-15) with (2-17), we can see that v_j serves as a weighting factor in the Huber gradient. For $|e_j| \le k$, v_j is defined in (2-14) as e_j, which is the same as in the ℓ_2 gradient given by (2-17). For $|e_j| > k$, v_j is held constant at the value of e_j at the threshold. In other words, the Huber gradient can be thought of as a modified ℓ_2 gradient where the gross errors are reduced to the threshold value.

The Hessian matrix of the Huber objective function F w.r.t. $\boldsymbol{x}$ can be expressed as

$$H = \sum_{j=1}^{m} (d_j \, e_j' \, e_j'^T + v_j \, e_j'') \tag{2-18}$$

where

$$d_j \triangleq \frac{\partial^2 \rho_k(e_j(x))}{\partial e_j^2(x)} = \begin{cases} 1 & if \; |e_j(x)| \le k \\ 0 & if \; |e_j(x)| > k \end{cases} \tag{2-19}$$

$$e_j'' \triangleq \frac{\partial e_j'^T}{\partial x} \tag{2-20}$$

Compared with the ℓ_2 Hessian matrix given by

$$H_{\ell_2} = \sum_{j=1}^{m} (e_j' \, e_j'^T + e_j \, e_j'') \tag{2-21}$$

the contribution of the gross errors is reduced. For each j in (2-18), if e_j is an outlier, the first item is completely eliminated and the effect of the second item is reduced to threshold.

2.4 A DATA FITTING EXAMPLE

To demonstrate the theoretical properties of the Huber formulation, we present a data fitting problem in the presence of large and small errors. This experiment was first performed by Bandler, Chen, Biernacki, Madsen, Li Gao and Yu [13].

The data is generated based on sampling the function $\sqrt{t}$. The function is uniformly sampled from $t = 0.02$ to $t = 1$, with a 0.02 interval between adjacent sampling points. Large errors are deliberately introduced at 5 of the sample points. The rest of the data is altered with some small variations. This deviation from the exact mathematical values is a simulated scenario of real-world data. The data can be denoted by $S_j = S(t_j)$, $t_j = 0.02, 0.04, \ldots, 1$.

The assumed mathematical model is a rational function defined as

$$R(\boldsymbol{x},t) = \frac{x_1 t + x_2 t^2}{1 + x_3 t + x_4 t^2} \tag{2-22}$$

where the parameters $\boldsymbol{x} = [x_1 \; x_2 \; x_3 \; x_4]^T$ are to be determined so that function R best fits the data S_j.

This problem can be formulated as an optimization procedure defined as

$$\underset{x}{\text{minimize}} \quad F(x) \triangleq \sum_{j=1}^{m} f(R(x,t_j) - S(t_j)) \tag{2-23}$$

where f can be any penalty term, such as Huber, ℓ_1 and ℓ_2 functions.

Starting from $x = [10 \ \ 10 \ \ 10 \ \ 10]^T$, ℓ_1, ℓ_2 and Huber optimization methods are invoked for comparison. The solutions are as follows.

ℓ_1:	$x_{\ell_1} = [11.28 \ \ 43.5602 \ \ 35.5686 \ \ 18.3375]^T$
ℓ_2:	$x_{\ell_2} = [45.2667 \ \ 564.122 \ \ 280.427 \ \ 429.965]^T$
Huber:	$x_{Huber} = [13.1555 \ \ 62.9922 \ \ 45.0118 \ \ 29.6621]^T$

A graphical representation of the solutions is given in Fig. 2.6. A portion of Fig. 2.6 is enlarged in Fig. 2.7 for a clearer view of the details.

As expected, the least-squares solution suffers significantly from the presence of the 5 erroneous points. This is due to the quadratic error entries into the ℓ_2 objective; gross errors have an overwhelming influence. On the other hand, the ℓ_1 solution, according to the optimality condition, is dictated by a subset of error functions which have zero values at the solution. In a sense, all the nonzero errors are viewed as large errors. This tendency towards a biased ℓ_1 solution, as dramatized in our example, is undesirable if we wish to model the small variations in the data.

The Huber solution features a flexible combination of the robustness of the ℓ_1 and the unbiasedness of the ℓ_2. In fact, the Huber solution is equivalent to an ℓ_2 solution with the gross errors reduced to the threshold value k. In our example, k is chosen as 0.04 according to the magnitude of the small variations in the data.

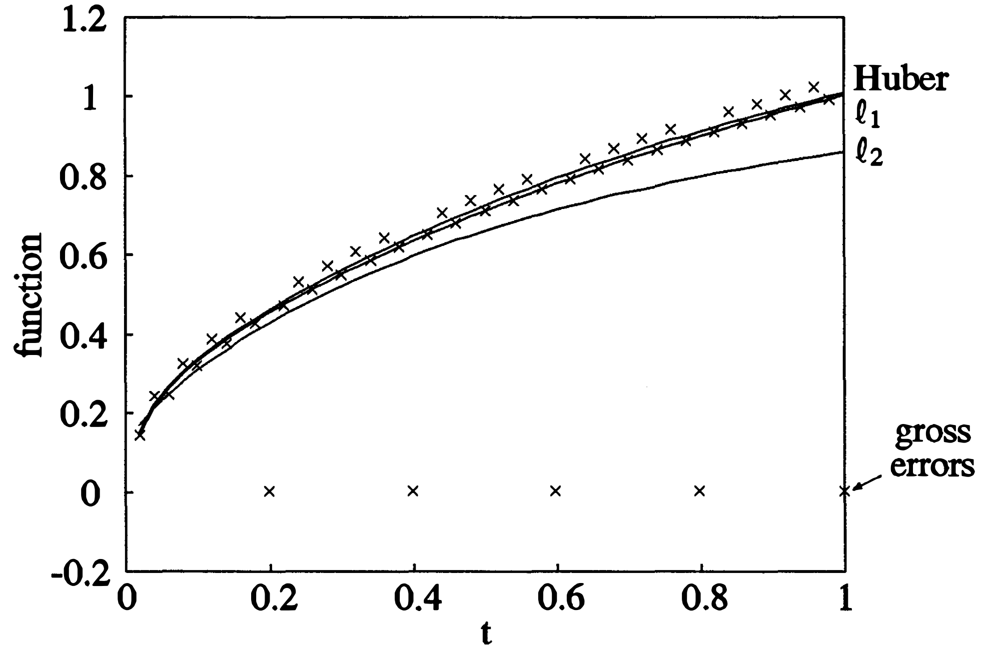

Fig. 2.6. ℓ_1, ℓ_2 and Huber solutions for data fitting in the presence of errors.

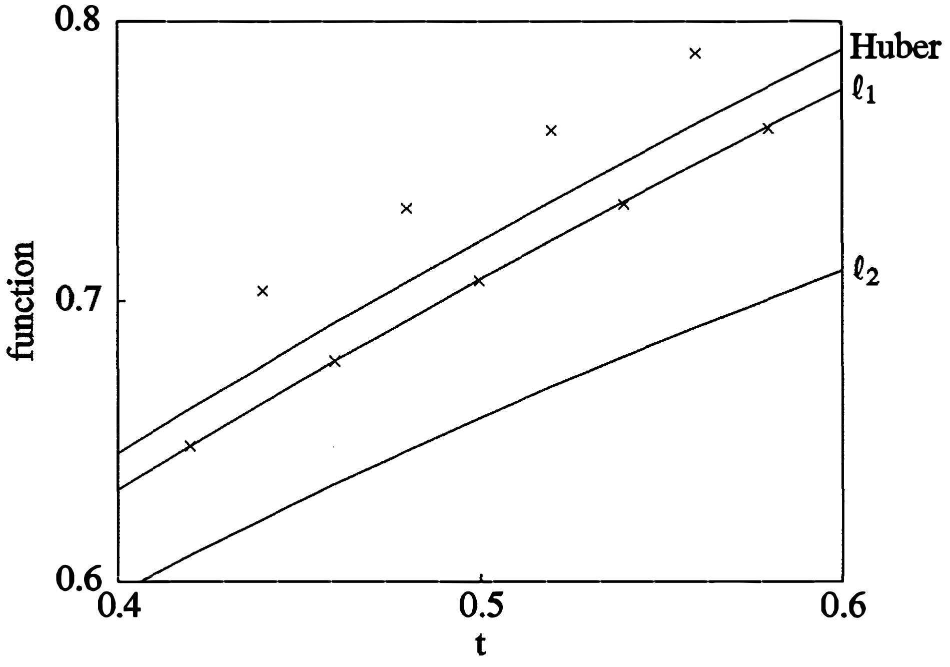

Fig. 2.7. An enlarged portion of Fig. 2.6.

2.5 ONE-SIDED HUBER-BASED OPTIMIZATION

One-sided Huber-based optimization is developed for engineering design problems with upper/lower specifications. For example, in designing an amplifier, we may require that the voltage gain should be greater than 20 dB within a certain frequency band. This is a lower specification: we do not desire the voltage gain to be exactly 20 dB; the design objective is achieved when the voltage gain is above that value.

The one-sided Huber-based optimization is defined as

$$\underset{\boldsymbol{x}}{\text{minimize}} \quad F(\boldsymbol{x}) \triangleq \sum_{j=1}^{m} \rho_k^+(e_j(\boldsymbol{x})) \tag{2-24}$$

where

$$\rho_k^+(e) = \begin{cases} 0 & \text{if } e \leq 0 \\ e^2/2 & \text{if } 0 < e \leq k \\ ke - k^2/2 & \text{if } e > k \end{cases} \tag{2-25}$$

This one-sided Huber function is truncated when negative because the corresponding design specification is satisfied. Similar to Fig. 2.5, Fig. 2.8 depicts the one-sided Huber function.

The gradient vector of the one-sided Huber objective function F w.r.t. $\boldsymbol{x}$ is given by

$$\nabla F = \sum_{j=1}^{m} v_j^+ \, e_j' \tag{2-26}$$

where

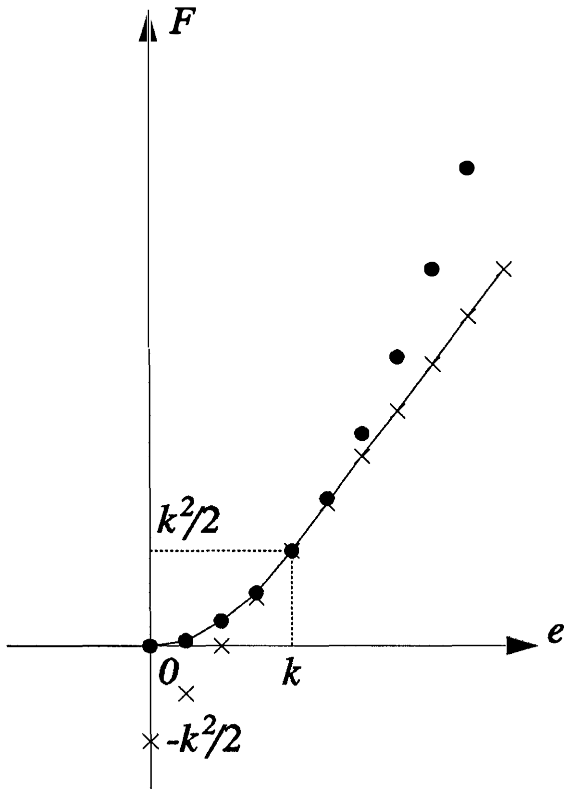

Fig. 2.8 The one-sided Huber function. The strikes and dots on the right half plane are the traces of the ℓ_1 and ℓ_2 curves, respectively, as is drawn in Fig. 2.5(a). The continuous curve indicates the one-sided Huber function. Notice it continues to the left half plane, where it remains zero.

$$v_j^+ \triangleq \frac{\partial \rho_k^+}{\partial e_j} = \begin{cases} 0 & if\ e_j \leq 0 \\ e_j & if\ 0 < e_j \leq k \\ k & if\ e_j > k \end{cases} \tag{2-27}$$

The Hessian matrix of the one-sided Huber objective function is given by

$$H = \sum_{j=1}^{m} (d_j^+ \, e_j' \, e_j'^T + v_j^+ \, e_j'') \tag{2-28}$$

where

$$d_j^+ \triangleq \frac{\partial^2 \rho_k^+}{\partial e_j^2} = \begin{cases} 0 & if\ e_j \leq 0 \\ 1 & if\ 0 < e_j \leq k \\ 0 & if\ e_j > k \end{cases} \tag{2-29}$$

Formulas (2-26) to (2-29) show similar structure to (2-14), (2-15), (2-18) and (2-19).

2.6 CONCLUDING REMARKS

We have presented the Huber functions through theoretical formulations: both one- and two-sided. The gradients and Hessians of the Huber objective functions have been derived. The robustness of the Huber-based optimization has been illustrated through both abstract derivation and a numerical example.

The Huber function ρ_k is a function increasing less rapidly than the quadratic functions. The impact of large errors is therefore reduced. Function ρ_k has a bounded gradient w.r.t. the error e. A consequence of this is that the Huber-based solution is insensitive to changes of the data points that are outside the threshold.

In data fitting, the Huber solution is robust against bad data points and still preserves the ability to model small, statistical variations.

The one-sided Huber-based optimization is tailored to design problems with upper/lower specifications.

Chapter 3

ALGORITHMS FOR HUBER-BASED OPTIMIZATION

3.1 INTRODUCTION

This chapter deals with algorithms, solving the mathematical problems formulated in Chapter 2.

A large number of algorithms have been developed for optimization over the years. Summaries of the methods can be found in Bandler (1973) [24], Bunday (1984) [27], and Bandler and Chen (1988) [1].

Optimization methods can be classified as direct search methods and gradient-based methods. Gradient-based methods use gradient information to accelerate convergence towards the optimum. In cases when the exact gradients are not easily obtainable, the use of approximated gradients is proposed (Bandler and Chen [1]).

The majority of gradient-based methods belong to the Gauss-Newton, quasi-Newton, and conjugate gradient families. Conjugate gradient methods require minimum memory storage, much less than that demanded by the Gauss-Newton or quasi-Newton methods. The trade-off is that longer computation time may be needed for proper scaling or preconditioning, exact line search, etc., to ensure convergence. In cases when the storage for all the matrices and vectors is not manageable, conjugate gradient methods may be used.

Madsen [28] developed a general framework of optimization algorithms, which includes two stages. When the current iterate is far away from the solution, a trust-region type method is invoked. When close to the solution, a switch to a quasi-Newton implementation is made for fast final convergence. Based on this framework, Ekblom and Madsen [15] provided a dedicated algorithm to minimize the two-sided Huber objective function, and proved that the first-stage trust-region method provides global convergence. Gao Li and Madsen [16] presented an implementation of this Ekblom Madsen algorithm (EMA). Bandler, Chen, Biernacki, Li Gao, Madsen and Yu [18] refined the code of the two-sided algorithm [16] and contributed a one-sided algorithm.

In this chapter, we provide a review of the Ekblom Madsen algorithm for both one- and two-sided problems. We will concentrate on the two-sided algorithm. However, as can be seen in Subsection 3.4.3, only a few adjustments are needed to convert it to solve a one-sided problem. The structure of this chapter is as follows.

Section 3.2 provides a detailed description of the first-stage method which exploits the exclusive structures of the Huber function and Huber objective function. In Section 3.3, the second-stage method is briefly described with a Broyden-Fletcher-Goldfarb-Shanno (BFGS) type implementation. The overall hybrid EMA, as well as the switch criteria between the two stages, is addressed in Section 3.4. Appropriate adjustments needed for solving one-sided problems are also presented in Section 3.4. It should be noticed that, except for Subsection 3.4.3, all the sections and subsections in this chapter address two-sided EMA.

A summary of various algorithms for Huber-based optimization can be found in Ekblom and Madsen [15].

3.2 THE FIRST-STAGE METHOD [15,16,28]

The first stage of the Ekblom Madsen algorithm [15] is a trust-region type method proposed by Madsen [28]. As is shown by Moré (1982) [29], trust-region methods are an important class of iterative methods for solving nonlinear optimization problems which are considered to be reliable, efficient and amazingly free of ad-hoc decisions. A complete survey of the development of algorithms and software for trust-region methods is given in [29].

The first-stage method solves the entire nonlinear Huber-based problem sequentially by converting the problem into a series of locally linearized subproblems. As proved by Ekblom and Madsen [15], it ensures global convergence.

3.2.1 Local Linearization [15,16,28]

The first-stage method is an iterative process. At this stage, one calculates a sequence of points $\{x_p\}$ intended to converge to a local minimum of F. At each iterate x_p, a linear function l_j is used to approximate the nonlinear error function e_j, $j = 1, 2, ..., m$, and thus a linearized model L_p of F is constructed. This model is a good approximation to F within a specified neighbourhood N_p of the pth iterate x_p. This neighbourhood N_p, named the trust region, is intended to reflect the domain in which the l_j approximations of the e_j are valid.

Assume a tentative step $\boldsymbol{h}$ is being searched at the pth iterate $\boldsymbol{x}_p$. If the search is successful, one goes on to the next iteration, i.e., $\boldsymbol{x}_{p+1} = \boldsymbol{x}_p + \boldsymbol{h}$. The problem is formulated as

$$\underset{\boldsymbol{h}}{\mathit{minimize}} \quad L_p(\boldsymbol{h}) \triangleq L(\boldsymbol{h}, \boldsymbol{x}_p) = \sum_{j=1}^{m} \rho_k(l_j(\boldsymbol{h}, \boldsymbol{x}_p)) \tag{3-1}$$

where

$$l_j(\boldsymbol{h}, \boldsymbol{x}_p) \triangleq e_j(\boldsymbol{x}_p) + [e_j'(\boldsymbol{x}_p)]^T \boldsymbol{h} \tag{3-2}$$

subject to the constraint $\boldsymbol{h} \in N_p$, where

$$N_p \triangleq \left\{ \boldsymbol{x} \mid \| \boldsymbol{x} - \boldsymbol{x}_p \| \le \delta_p \right\} \tag{3-3}$$

and where $\| \cdot \|$ denotes the Euclidean norm.

Similar to formulas (2-14,16,19,20), the following definitions are introduced.

$$v_j^* \triangleq \frac{\partial \rho_k(l_j(\boldsymbol{x}))}{\partial l_j(\boldsymbol{x})} = \begin{cases} l_j(\boldsymbol{x}) & \mathit{if}\ |l_j(\boldsymbol{x})| \le k \\ k & \mathit{if}\ l_j(\boldsymbol{x}) > k \\ -k & \mathit{if}\ l_j(\boldsymbol{x}) < -k \end{cases} \tag{3-4}$$

$$\boldsymbol{l}_j' \triangleq \left[\frac{\partial l_j(\boldsymbol{x})}{\partial x_1} \ \frac{\partial l_j(\boldsymbol{x})}{\partial x_2} \ \cdots \ \frac{\partial l_j(\boldsymbol{x})}{\partial x_n} \right]^T = \boldsymbol{e}_j' \tag{3-5}$$

$$d_j^* \triangleq \frac{\partial^2 \rho_k(l_j(\boldsymbol{x}))}{\partial l_j^2(\boldsymbol{x})} = \begin{cases} 1 & \mathit{if}\ |l_j(\boldsymbol{x})| \le k \\ 0 & \mathit{if}\ |l_j(\boldsymbol{x})| > k \end{cases} \tag{3-6}$$

$$\boldsymbol{l}_j'' \triangleq \frac{\partial \boldsymbol{l}_j'^T}{\partial \boldsymbol{x}} = \boldsymbol{0} \tag{3-7}$$

The Jacobian of e_j, $j = 1, 2, \ldots, m$ is

$$\boldsymbol{e}' \triangleq [\boldsymbol{e}_1'^T \ \boldsymbol{e}_2'^T \ \ldots \ \boldsymbol{e}_m'^T]^T \tag{3-8}$$

The vector $\boldsymbol{v}^*$ and the matrix $\boldsymbol{D}^*$ are also defined as

$$\boldsymbol{v}^* \triangleq [v_1^* \;\; v_2^* \;\; \dots \;\; v_m^*]^T \tag{3-9}$$

$$D^* \triangleq diag\{d_j^*\} \tag{3-10}$$

Consequently, the gradient and Hessian of the linearized model L_p are

$$\nabla L_p = \sum_{j=1}^{m} v_j^* \, l_j' = e'^T \boldsymbol{v}^* \tag{3-11}$$

$$H^* = \sum_{j=1}^{m} [d_j^* \, l_j' \, l_j'^T + v_j^* \, l_j''] = \sum_{j=1}^{m} d_j^* \, e_j' \, e_j'^T = e'^T D^* e' \tag{3-12}$$

respectively.

These formulas will be used in describing the algorithm later in this chapter.

3.2.2 The Trust-Region Methods in Searching for Optimal Step $\boldsymbol{h}$ [15,16,28]

The scheme of solving problem (3-1,2,3) employs the principle of the well-known Newton method and utilizes a Lagrange multiplier (see Bandler [24]).

The constrained minimization problem (3-1,2,3) can be converted to solving

$$\underset{\boldsymbol{h}}{minimize} \quad \Psi(\boldsymbol{h}) \triangleq L_p(\boldsymbol{h}) + \lambda(\boldsymbol{h}^T\boldsymbol{h} - \delta_p^2) \tag{3-13}$$

where λ is the Lagrange multiplier.

When $\| \boldsymbol{h}_p \| < \delta_p$, the problem becomes an unconstrained problem, and λ is set to zero. Otherwise, according to the Kuhn-Tucker conditions for constrained optimum, the solution for the pth iterate $\boldsymbol{h}_p$ satisfies

$$\begin{cases} \nabla_{\boldsymbol{h}} \Psi(\boldsymbol{h}, \lambda) = \boldsymbol{0} \\ \boldsymbol{h}^T\boldsymbol{h} - \delta_p^2 = 0 \end{cases} \tag{3-14}$$

where $\nabla_{\boldsymbol{h}}\Psi$ denotes the first derivative of Ψ w.r.t. $\boldsymbol{h}$ derived as

$$\nabla_{\boldsymbol{h}}\Psi \triangleq \frac{\partial \Psi}{\partial \boldsymbol{h}} = \nabla_{\boldsymbol{h}} L_p(\boldsymbol{h}) + 2\lambda \boldsymbol{h} \tag{3-15}$$

where $\nabla_{\boldsymbol{h}} L_p$ is the first derivative of the linear model L_p w.r.t. $\boldsymbol{h}$.

The Newton iteration for solving the system of equations (3-14) is

Newton Equation Solver 1 (NES1):

```
**while not** STOP **do begin**

    solve $\begin{bmatrix} H^*(\boldsymbol{h})+2\lambda I & 2\boldsymbol{h} \\ 2\boldsymbol{h}^T & 0 \end{bmatrix} \begin{bmatrix} \Delta \boldsymbol{h} \\ \Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla_{\boldsymbol{h}} L_p(\boldsymbol{h}) + 2\lambda\boldsymbol{h} \\ \boldsymbol{h}^T\boldsymbol{h} - \delta_p^2 \end{bmatrix}$    (3-16)

    $(\boldsymbol{h}, \lambda) := (\boldsymbol{h}, \lambda) + (\Delta\boldsymbol{h}, \Delta\lambda)$

**end;**

$(\boldsymbol{h}_p, \lambda_p) := (\boldsymbol{h}, \lambda)$
```

The STOP criterion could be the condition either the norm of the left hand side of equation (3-14) is less than ε_1 or $\|(\Delta\boldsymbol{h}, \Delta\lambda)\| < \varepsilon_2 \|(\boldsymbol{h}, \lambda)\|$, where ε_1 and ε_2 are small positive numbers.

In NES1, apart from $\boldsymbol{h}$, one tries to obtain λ_p iteratively, according to the fixed size of the trust region δ_p. In fact, λ_p and δ_p are dual through (3-14), i.e., one can also assign the value for λ_p and fix the value of δ_p accordingly. Moreover, the first set of equations in (3-14) is independent of δ_p. Therefore, when the value of λ_p is known, one can solve $\boldsymbol{h}$ without the last equation of (3-14) since there is no need to solve for δ_p. This leads to a modification to NES1, which is easier to implement.

Newton Equation Solver 2 (NES2):

```
while not STOP do begin
    solve $(H^*(h)+2\lambda_p I)\Delta h = -(\nabla_h L_p(h)+2\lambda_p h)$        (3-17)
    $h := h + \Delta h$
end;
$h_p := h$
```

Similarly, the STOP criterion could be either $\| \nabla_h L_p(h) + 2\lambda_p h \| < \varepsilon_1$ or $\| \Delta h \| < \varepsilon_2 \| h \|$.

3.2.3 Updating Trust-Region Size δ_p or Controlling Factor λ_p [15,16,28]

NES1 and NES2 are two different sub-algorithms of the first-stage method, solving the system of nonlinear equations (3-14). Their goal is to find the increment h_p to add to the current iterate x_p, minimizing Ψ in (3-13). The difference lies in that: in NES1, δ_p is assigned and λ_p is to be found; while in NES2, λ_p is assigned and δ_p is unknown.

Before starting the next iteration, the size of the trust region needs to be adjusted. This is accomplished directly by changing the trust-region size δ_p (for NES1), or indirectly by updating the controlling factor λ_p (for NES2).

In compliance with Madsen's framework [28], the adjustment scheme is based on the ratio between the reduction in the nonlinear Huber objective function F and the reduction in the local linear approximation L_p, i.e.,

$$r_p = \frac{F(x_p) - F(x_p + h_p)}{L_p(0) - L_p(h_p)} \tag{3-18}$$

When r_p is close to 1, one can afford a larger trust region; if r_p is too small, the size of the trust region must be reduced. The new point $\boldsymbol{x}_p + \boldsymbol{h}_p$ is accepted as the new iterate if the reduction in F exceeds a small multiple of the reduction in L_p. Otherwise, the tentative step $\boldsymbol{h}_p$ has to be recalculated with a reduced trust region.

Incorporating this adjustment scheme into the two NES's leads to two versions of the first-stage method, named first-stage 1 (FS1) and first-stage 2 (FS2) methods, depending on which nonlinear equation solver is used.

As is listed below, FS1 algorithm utilizes NES1 and updates the trust-region size δ_p.

First-Stage 1 (FS1):

```
Let $0 < s_1 << 0.25$ and $s_2 < 1 < s_3$.
given $x_0$ and $\delta_0$; $p := 0$;
**while not** OUTERSTOP **do begin**
    perform NES1;
    **if** $r_p \geq s_1$       **then** $\boldsymbol{x}_{p+1} := \boldsymbol{x}_p + \boldsymbol{h}_p$
                     **else** $\boldsymbol{x}_{p+1} := \boldsymbol{x}_p$;
    **if**      $r_p \leq 0.25$      **then** $\delta_{p+1} := s_2 \times \delta_p$
    **else if** $r_p \geq 0.75$      **then** $\delta_{p+1} := s_3 \times \delta_p$
                              **else** $\delta_{p+1} := \delta_p$;
    $p := p + 1$
**end**
```

The following FS2 utilizes NES2 and updates λ_p.

First-Stage 2 (FS2):

```
Let $0 < s_1 << 0.25$ and $s_3 < 1 < s_2$.
given $x_0$ and $\lambda_0$; $p := 0$;
while not OUTERSTOP do begin
    perform NES2;
    if $r_p \geq s_1$      then $x_{p+1} := x_p + h_p$
                           else $x_{p+1} := x_p$;
    if      $r_p \leq 0.25$    then $\lambda_{p+1} := s_2 \times \lambda_p$
    else if $r_p \geq 0.75$    then $\lambda_{p+1} := s_3 \times \lambda_p$
                               else $\lambda_{p+1} := \lambda_p$;
    $p := p + 1$
end
```

For OUTERSTOP in both versions, Ekblom and Madsen [15] suggested to use the criteria $\| \nabla F(x_p) \| < \varepsilon_3$ or $\| h_p \| < \varepsilon_4 \| x_p \|$, where ε_3 and ε_4 are small positive numbers. A close look into the two versions yields that reducing the value of the controlling factor λ corresponds to enlarging the size of the trust region.

Ekblom and Madsen [15] proved that the first-stage method has global convergence. Also in [16], it is shown that this method follows the usual convergence theory for trust-region methods.

As is indicated in [15], FS1 is a true trust-region method whereas FS2 is of the Levenberg-Marquardt type. FS2 is easier to implement, but might have slow final convergence. This potential slow final convergence can be eliminated by introducing a quasi-Newton method as the second-stage method. Throughout this chapter, unless otherwise indicated, an FS2 implementation is assumed for the first stage.

3.2.4 Comparison of the First-Stage Method and Conventional Newton Methods [15,16]

Ekblom and Madsen [15] and Gao Li and Madsen [16] compared the first-stage method with solving the Huber-based problems using conventional Newton methods. The following is a brief summary.

Suppose the error functions e_j are linear. A classical Newton iteration (without the trust region) is formulated as

$$\begin{cases} \boldsymbol{e}'^T \boldsymbol{D}(\boldsymbol{x}_p) \boldsymbol{e}' \boldsymbol{h} = -\boldsymbol{e}'^T \boldsymbol{v}(\boldsymbol{x}_p) \\ \boldsymbol{x}_{p+1} = \boldsymbol{x}_p + \boldsymbol{h}\alpha \end{cases} \tag{3-19}$$

where $\boldsymbol{e}'$ is defined in (3-8), and $\boldsymbol{D}$ and $\boldsymbol{v}$ are similar to those defined in (3-9,10), with v_j^* and d_j^* replaced by v_j and d_j, $j = 1, 2, ..., \mathrm{m}$, defined in (2-14) and (2-19), respectively, and α is a step length parameter.

Equation (3-19) can be used for solving nonlinear problems by letting $\boldsymbol{e}'$ hold the current value of the Jacobian. The iteration can be stabilized by a damping factor λ, which leads to the so-called "Generalized Levenberg-Marquardt Method" (GLMM, see Ekblom and Madsen [15]), which is described as

$$\begin{cases} [\boldsymbol{e}'^T \boldsymbol{D}(\boldsymbol{x}_p) \boldsymbol{e}' + \lambda \boldsymbol{I}] \boldsymbol{h} = -\boldsymbol{e}'^T \boldsymbol{v}(\boldsymbol{x}_p) \\ \boldsymbol{x}_{p+1} = \boldsymbol{x}_p + \boldsymbol{h} \end{cases} \tag{3-20}$$

The basic difference between the first-stage method and GLMM is that GLMM approximates the entire Huber objective function by a quadratic function while the first-stage method linearizes individual error functions. Therefore, the approximated Hessian of the GLMM model has bigger error if there is any outlier (see Bandler, Chen, Biernacki, Li Gao, Madsen and Yu [18]). Moreover, one iteration of (3-20) is equivalent to one step of FS2 with only one iteration of NES2, starting at $\boldsymbol{h} = \boldsymbol{0}$. In other words, FS2 has finer and more accurate adjustments during the search, and consequently, is more efficient.

Furthermore, as Li Gao and Madsen [16] indicated, GLMM often fails to give a decrease in the model function when $\boldsymbol{x}$ is far away from the solution. Therefore, this method only provides convergence if the starting value of the nonlinear iteration is close to the solution.

3.3 THE SECOND-STAGE METHOD [28,30]

Quasi-Newton methods (see, e.g., Luenberger [30]) are conjugate direction types of methods. They inherit the fast convergence associated with the Newton method while avoiding the costly inversion of the Hessian. The true Hessian of the original objective function, or its inverse, is approximated by a positive definite matrix, updated based on the information gathered during the process. Bandler and Chen [1] gave a brief description of quasi-Newton methods. More detailed discussion can be found in Luenberger [30].

In solving nonlinear optimization problems, Madsen [28] proposed a quasi-Newton procedure as the second-stage method. The second-stage quasi-Newton procedure in the Ekblom Madsen algorithm ensures fast final convergence for Huber-based problems, quite likely from a position that the utilization of the first-stage trust-region method is not as valid and efficient.

At the pth iterate x_p, a typical quasi-Newton step is described as

$$\begin{cases} B_p h_p = -\nabla F(x_p) \\ x_{p+1} = x_p + h_p \end{cases} \tag{3-21}$$

where h_p is the increment to be added to x_p, and ∇F is the gradient of the nonlinear Huber objective function defined in (2-15).

The matrix B_p in (3-21) is an estimate of the true Hessian of the nonlinear Huber objective function (2-18) at the pth iterate. Usually starting from the identity matrix, it is updated in each iteration based on the information accumulated during the procedure. The famous BFGS updating scheme (see [30]) is defined as follows.

$$B_{p+1} = \begin{cases} B_p + \dfrac{h_p h_p^T}{g_p^T h_p} - \dfrac{B_p h_p h_p^T B_p}{h_p^T B_p h_p} & \textit{if } h_p^T g_p > 0 \\ B_p & \textit{otherwise} \end{cases} \tag{3-22}$$

where

$$g_p \triangleq \nabla F(x_{p+1}) - \nabla F(x_p) \tag{3-23}$$

The condition $h_p^T g_p > 0$ is to preserve the positive definiteness of B_{p+1}.

A dual to this BFGS updating scheme, another class of quasi-Newton methods approximates and updates the inverse of the Hessian H^{-1} in a similar manner. The well-known Davidon-Fletcher-Powell (DFP) method is such a method (see, for example, Luenberger [30]). Nevertheless, as Luenberger [30] pointed out, the performance of DFP is highly sensitive to the accuracy of the line search along the conjugate direction, which is very CPU time-consuming.

3.4 THE OVERALL EKBLOM MADSEN ALGORITHM [15,16,18,28]

3.4.1 The Two-Sided EMA [15,16,18]

The Ekblom Madsen algorithm [15,16,18] proposed to solve the Huber-based problem (2-13) is a combination of the two methods described in Sections 3.2 and 3.3, respectively. Without losing generality, the BFGS code is assumed for the second-stage method.

At the beginning of the procedure, the initial estimates for the iterate $\boldsymbol{x}_0$, the Lagrange multiplier λ_0 and the Hessian approximation $\boldsymbol{B}_0$ are to be given.

As is shown in Section 3.2, the trust-region FS2 is first invoked. At the pth iterate, one iteration of FS2 is carried out for the next iterate $\boldsymbol{x}_{p+1}$. Equation (3-22) is invoked to keep the estimate of the Hessian $\boldsymbol{B}_{p+1}$ up to date.

Before performing the next FS2 iteration, determination has to be made as to whether one should switch to the second-stage method. When a local minimum $\boldsymbol{x}^*$ seems to be approached, the switch is made. The following criterion is used for this judgement,

$$\| \nabla F(\boldsymbol{x}_p) \| \leq 0.02 F(\boldsymbol{x}_p) \tag{3-24}$$

If (3-24) has been satisfied for three consecutive FS2 iterations, the new point $\boldsymbol{x}_{p+1}$ is temporarily stored in vector $\boldsymbol{x}_s$ and a switch to the second-stage method is then made. Otherwise, the next FS2 iteration is to be performed. The use of $\boldsymbol{x}_s$ will be mentioned later.

When switched to the second stage, the new iterate $\boldsymbol{x}_{p+1}$ and the corresponding Hessian approximation $\boldsymbol{B}_{p+1}$ are calculated by (3-21) and (3-22), respectively. The Lagrange multiplier λ_p is not used in this stage; it is re-indexed for algorithmic reasons, i.e., $\lambda_{p+1} = \lambda_p$.

Next, one has to determine whether he should continue with the second-stage method or switch back to the first-stage. If the condition

$$\| \nabla F(x_{p+1}) \| \leq 0.99 \| \nabla F(x_p) \| \tag{3-25}$$

is satisfied, the next quasi-Newton iteration is carried out. Otherwise, a switch to FS2 is made.

When a decision is made to switch back to the first stage, there are two optional points from which the trust-region procedure can be restarted. The two choices are 1) the point stored in the temporary vector x_s and 2) the point reached by the last quasi-Newton iteration x_{p+1}. They are compared in terms of the nonlinear objective function F. x_{p+1} is replaced by x_s, i.e., $x_{p+1} = x_s$, if

$$F(x_{p+1}) > F(x_s) \tag{3-26}$$

In this case, the first-stage method continues as if the quasi-Newton iteration has not taken place. The only difference is that the $\boldsymbol{B}$ matrix is updated. If (3-26) is not satisfied, x_{p+1} will be kept as is.

3.4.2 Comparison of the Two-Sided EMA and Other Generic Methods

Since the Huber objective function is continuous and has a continuous gradient, it may be tempting to conclude that it is a straightforward matter to formulate the objective function and then minimize it by a generic algorithm, such as an ordinary quasi-Newton method or a direct search method.

We conducted a comparison between the dedicated EMA and three generic algorithms available in the OSA90/hope system: quasi-Newton, conjugate gradient and simplex search.

We attempted to apply the generic algorithms to the data fitting problem of Section 2.4, which involves four variables. None of them is able to find the correct solution unless starting very close to the solution. It attests the need for the dedicated EMA for solving multidimensional problems.

In Chapter 4, we will demonstrate a comparison of dedicated and generic algorithms for MESFET statistical modeling, which shows that the dedicated EMA is more efficient than the generic ones.

As derived by (2-18), the Hessian of the Huber objective function is discontinuous whenever one of the error functions crosses the threshold value. This obviously poses a serious problem for generic algorithms that explicitly rely on the continuity of the Hessian matrix [18].

3.4.3 The One-Sided EMA [18]

Necessary adjustments to the EMA have been made by Bandler, Chen, Biernacki, Li Gao, Madsen and Yu [18] to develop a one-sided method for engineering design problems. We refer to it as one-sided EMA. This subsection is intended to addresses the differences between the one- and two-sided algorithms. A brief description of the one-sided EMA can be found in [31].

The basic difference is traced back to the problem formulation early in this chapter. When a one-sided problem is to be solved, the one-sided Huber function ρ_k^+

(2-25) is to be used instead in the problem formulation (3-1). Therefore, instead of (3-1), the locally linearized problem is defined as

$$\underset{\boldsymbol{h}}{\mathit{minimize}} \quad L_p(\boldsymbol{h}) \triangleq L(\boldsymbol{h}, \boldsymbol{x}_p) = \sum_{j=1}^{m} \rho_k^+(l_j(\boldsymbol{h}, \boldsymbol{x}_p)) \tag{3-27}$$

subject to (3-3), where $l_j(\boldsymbol{h}, \boldsymbol{x}_p)$ is defined in (3-2).

Correspondingly, (3-4) and (3-6) are replaced by

$$v_j^{+*} \triangleq \frac{\partial \rho_k^+(l_j(\boldsymbol{x}))}{\partial l_j(\boldsymbol{x})} = \begin{cases} 0 & \mathit{if}\ l_j \le 0 \\ l_j & \mathit{if}\ 0 < l_j \le k \\ k & \mathit{if}\ l_j > k \end{cases} \tag{3-28}$$

$$d_j^{+*} \triangleq \frac{\partial^2 \rho_k^+(l_j(\boldsymbol{x}))}{\partial l_j^2(\boldsymbol{x})} = \begin{cases} 0 & \mathit{if}\ l_j \le 0 \\ 1 & \mathit{if}\ 0 < l_j \le k \\ 0 & \mathit{if}\ l_j > k \end{cases} \tag{3-29}$$

In the algorithm, some relevant changes are needed. Firstly, in calculating the gradient $\nabla_{\boldsymbol{h}} L_p$ and the Hessian H^* in (3-17), v_j^{+*} and d_j^{+*} are used to replace v_j^* and d_j^*, respectively. Secondly, the gradient of the nonlinear function ∇F used in (3-21) and (3-23) should be calculated by (2-26) instead of (2-15).

3.5 CONCLUDING REMARKS

We have reviewed the dedicated Ekblom Madsen algorithms for Huber-based optimization, both one- and two-sided. The two stages of the algorithms are identified. The trust-region stage has been overviewed in great detail, while the implementation of the quasi-Newton method has been described at a more general level. Switching procedures and criteria between the two stages have been addressed for the overall method. Comparison has also been made between the first-stage trust-region method and classical Newton methods, as well as between the dedicated EMA and other generic optimizers.

The first-stage trust-region method is the center of this chapter. It utilizes the special structures of the Huber objective functions, and is shown to have more accurate and efficient control over the incremental step of the current iterate, compared to generic Newton methods. When the solution seems to be approached, a switch is made to the second-stage quasi-Newton method for fast final convergence. If the current iterate is tested as not approaching the solution, a switch is made back to the first stage.

It is demonstrated by numerical examples in this chapter, and later in Chapter 4, that this dedicated Ekblom Madsen algorithm is more effective and efficient than generic optimizers which formulate the Huber objective as their own objective.

Chapter 4

ROBUST STATISTICAL MODELING

4.1 INTRODUCTION

Statistical device modeling is a prerequisite for accurate yield-driven or cost-driven circuit analysis and optimization [1,32,33]. It is to represent the model parameter statistics, and by doing so, it can provide tools for generating random device outcomes which will reproduce the "close-to-reality" statistical distribution of the device responses.

Statistical modeling has been an active subject for more than one decade. Cai (1992) [34] gave a survey with emphasis on physics-based device modeling (PBM). In the following paragraphs, some highlights of the history are pinpointed.

Dutton, Divekar, Gonzalez, Hansen and Antoniadis (1977) [35] studied the correlation of IC fabrication process and electrical device parameter variations. In the same year, Divekar, Dutton and McCalla (1977) [36] experimented on parameter correlations for bipolar junction transistors. Computer tools were employed to perform factor analysis resulting in a simplified statistical circuit analysis model.

Rankin (1982) [37] gave a summary on two approaches of statistical modeling for integrated circuits (ICs), namely the equivalent circuit model (ECM) and the physical model approaches. The easier ECM approach was proven to be expensive or even practically impossible to practice, and therefore sufficient only for circuit designs based on worst-case parameter variations. The physical model approach was advocated, which needs modeling equations to relate the equivalent circuit to the device physical structure.

Herr and Barnes (1986) [38] gave a detailed description of an approach to statistical modeling in MOS VLSI circuit design. The methodology of parameter extraction was explicitly described and a statistical analysis routine was given in detail. Spanos and Director (1986) [39] investigated the problem of estimating the statistics of disturbances of integrated circuit (IC) production process, given the statistics of the process outputs. A simplified statistical model was obtained by assuming that variations at each hierarchy of the model were of normal distribution.

Bandler, Biernacki, Chen, Loman, Renault and Zhang (1989) [40] proposed a heuristic technique to handle arbitrary statistical distributions substantially different from normal. It could preserve the estimated statistics from the measured data while retaining the simplicity of the normal distribution. The confidence of the statistical modeling and a suggested sample size for efficient statistical data processing were also addressed.

A case study by Bandler, Biernacki, Chen, Song, Ye and Zhang (1991) [41] showed that the physics-based Ladbrooke model [42] for GaAs MESFETs provides a better estimate of device statistics, while the Materka and Kacprzak model [43] based on equivalent circuit parameters provides a better match for individual devices. Bandler, Biernacki, Cai, Chen, Ye and Zhang (1992) [32] furthered the statistical modeling experiment by including a physics-based Khatibzadeh and Trew MESFET model [44].

This chapter addresses robust statistical modeling, one of the most promising application areas of Huber-based optimization. Three stages of the statistical modeling process, namely, parameter extraction, statistical parameter estimation and statistical model verification, are identified. Mathematical formulations for parameter extraction and statistical estimation are provided. Utilization of the Huber function in parameter extraction is proposed. An example with the Materka and Kacprzak model for MESFETs is used to demonstrate the robustness of Huber statistical estimation. Comparison is made with the classical ℓ_2 estimator. The same example is also used to compare the dedicated Huber optimization algorithm with generic methods. Threshold selection for statistical estimation is addressed, with illustrations.

4.2 STATISTICAL PARAMETER EXTRACTION AND ESTIMATION

Generally, statistical modeling starts with measured responses of a number of devices supposedly identical. One typical approach is to extract circuit/model parameters for each device, and then postprocess the extracted model parameters to estimate their statistics.

A complete statistical modeling process can be divided into three steps [34]:

1) statistical parameter extraction

2) statistical parameter estimation

3) statistical model verification.

The following subsections address them separately.

4.2.1 Statistical Parameter Extraction [34]

To unveil the statistics of a model parameter, a sample of that parameter is needed. This requires measurements taken from a collection of devices, which may include DC, small-signal and large-signal data. For each device with certain measurements, model parameters are extracted by optimization. The samples of model parameters are collected when all the devices under consideration are processed. Needless to say, consistent and reliable parameter extraction is essential to correctly estimating the model statistics.

Suppose we have K devices for parameter extraction. The ith device has n model parameters denoted by

$$\boldsymbol{\phi}^i = [\phi_1{}^i \; \phi_2{}^i \; \dots \; \phi_n{}^i]^T \tag{4-1}$$

and m model responses denoted by

$$\boldsymbol{R}(\boldsymbol{\phi}^i) = [R_1(\boldsymbol{\phi}^i) \; R_2(\boldsymbol{\phi}^i) \; \dots \; R_m(\boldsymbol{\phi}^i)]^T \tag{4-2}$$

where the superscript $i = 1, 2, \dots, K$, denotes the ith device. There are also m measured responses corresponding to the model responses for each device, namely,

$$\boldsymbol{S}^i = [S_1{}^i \; S_2{}^i \; \dots \; S_m{}^i]^T \tag{4-3}$$

The parameters $\boldsymbol{\phi}^i$ for the ith device are extracted by optimization formulated as (Bandler, Chen and Daijavad [10])

$$\underset{\boldsymbol{\phi}^i}{minimize} \;\; F(\boldsymbol{\phi}^i) \triangleq \sum_{j=1}^{m} f(R_j(\boldsymbol{\phi}^i) - S_j^i) \tag{4-4}$$

where f dictates the optimization method to be used. For example, f could be ℓ_1 or ℓ_2 functions [1].

A closer look at (4-4) yields that parameter extraction is essentially a data fitting problem. The data to be fitted corresponds to the measured responses $\boldsymbol{S}^i$. The approximating function is dictated by $\boldsymbol{R}(\boldsymbol{\phi}^i)$. Parameters $\boldsymbol{\phi}^i$ are to be adjusted for the best fit. Because the Huber function is robust in data fitting problems, as is shown in Section 2.4, it could be a promising candidate for the fitting criterion f.

4.2.2 Statistical Parameter Estimation

Model statistics are represented by parameter distributions and correlations. Two commonly assumed parameter distributions are the uniform and the normal distributions. As given in many statistics books (see, e.g., [45]), the one-dimensional uniform distribution can be described by the probability density function (pdf)

$$p(\phi) = \begin{cases} \dfrac{1}{c - b} & b \leq \phi \leq c \\ 0 & otherwise \end{cases} \tag{4-5}$$

and the one-dimensional normal distribution by

$$p(\phi) = \frac{1}{\sigma_\phi \sqrt{2\pi}} \exp\left[- \frac{(\phi - \overline{\phi})^2}{2\sigma_\phi^2} \right] \tag{4-6}$$

for the one-dimensional random variable ϕ. In the normal distribution (4-6), $\overline{\phi}$,σ_ϕ and σ_ϕ^2 denote the mean, the standard deviation and the variance, respectively.

Physical and process-related parameters, denoted as low-level parameters in the simulation model (see Bandler and Chen [1]), are usually assumed to fit a normal distribution and as statistically independent. The model parameters at higher levels might be of any statistical distribution, possibly complicated and correlated, as a result of linear/nonlinear transformations [36].

If a multidimensional normal distribution is assumed for the model parameters, statistical parameter estimation is simply finding the mean values, the standard deviations and correlation coefficients. These statistics can be calculated consecutively, from the parameters extracted, using computer routines (see, for example, [46]). This is called postprocessing, or statistical estimation.

The mean value of the jth parameter ϕ_j, $\overline{\phi}_j$, is defined and estimated by

$$\overline{\phi}_j = \frac{\sum_{i=1}^{K} \phi_j^i}{K} \tag{4-7}$$

The standard deviation can be calculated, using the mean value obtained from (4-7), by

$$\sigma_{\phi_j} = \sqrt{\frac{\sum_{i=1}^{K} (\phi_j^i - \overline{\phi}_j)^2}{K - 1}} \tag{4-8}$$

The correlation coefficient between parameters ϕ_{j_1} and ϕ_{j_2}, denoted by $c_{j_1 j_2}$, is available by evaluating

$$c_{j_1 j_2} = \frac{\sum_{i=1}^{K} (\phi_{j_1}^i - \overline{\phi}_{j_1})(\phi_{j_2}^i - \overline{\phi}_{j_2})}{\sqrt{\sum_{i=1}^{K} (\phi_{j_1}^i - \overline{\phi}_{j_1})^2 \sum_{i=1}^{K} (\phi_{j_2}^i - \overline{\phi}_{j_2})^2}} \tag{4-9}$$

Alternatively, the mean value of parameter ϕ_j can be estimated by optimization if we define the error functions as

$$e_i(x) = x - \phi_j^i \,, \quad i = 1, 2, \ldots, K \tag{4-10}$$

where x is an optimizable variable. The following mathematical developments show that when an ℓ_2 optimization is finished, the value of x at the solution will be the value of the mean $\overline{\phi}_j$.

Incorporating the errors (4-10) into the objective function, the classical least-squares (ℓ_2) method solves

$$\underset{x}{minimize} \quad F(x) \triangleq \sum_{i=1}^{K} e_i^2(x) = \sum_{i=1}^{K} (x - \phi_j^i)^2 \tag{4-11}$$

At the least-squares solution, the following condition must hold

$$\nabla_x F \triangleq \frac{\partial F}{\partial x} = \sum_{i=1}^{K} 2(x^* - \phi_j^i) = 0 \tag{4-12}$$

where x^* denotes the ℓ_2 solution. It can be further derived from (4-12) that

$$\sum_{i=1}^{K} x^* - \sum_{i=1}^{K} \phi_j^i = 0 \tag{4-13}$$

Keeping in mind that x^* is a constant, i.e.,

$$\sum_{i=1}^{K} x^* = Kx^* \tag{4-14}$$

Therefore,

$$x^* = \frac{\sum_{i=1}^{K} \phi_j^i}{K} \tag{4-15}$$

Not surprisingly, the mean value given in (4-7) is actually the solution of the ℓ_2 optimization (4-11).

More generally, to estimate the mean, we solve

$$\underset{x}{minimize} \quad F(x) \triangleq \sum_{i=1}^{K} f(e_i(x)) \tag{4-16}$$

Again, f dictates the optimization method to be applied (e.g., ℓ_1 or ℓ_2). At the solution, the value of the optimization variable x is the expected value of the parameter ϕ_j, in the sense associated with f. For example, if f is the ℓ_1 function, the solution will be the median.

Similarly, to estimate the variance, we define

$$e_i(x) = x - (\phi_j^i - \overline{\phi}_j)^2, \qquad i = 1, 2, ..., K \tag{4-17}$$

to be an error function, where x is to be optimized. A reliable value of the mean is a prerequisite to estimate the variance for this approach. The standard deviation σ_{ϕ_j} can easily be evaluated from the variance obtained. From the same mathematical reasoning (4-11 to 4-15), it follows that the value obtained by (4-8) is an estimation of the standard deviation in the ℓ_2 sense.

In this chapter, we use the optimization approach to apply the Huber concept to statistical estimation.

For cases where the sample distribution is sufficiently different from normal, a combined discrete/normal approach (Bandler, Biernacki, Chen, Loman, Renault and Zhang (1989) [40]) was proposed.

4.2.3 Statistical Model Verification [34]

One way to check the validity of the model statistics obtained is through Monte Carlo simulation.

Generally speaking, Monte Carlo simulation is a multiple-response generation process. First, a computer-generated sample of devices is created by a random number generator in compliance with a given statistical distribution for the device model parameters. Second, simulating each device individually yields a sample of responses.

Therefore, the sample of simulated responses is statistically consistent with the parameter distribution given to the random number generator. We perform Monte Carlo simulation to verify the validity of the model statistics by providing the model statistics obtained by the methods described in Section 4.2.2 to the random number generator. If the sample of the simulated responses agrees statistically with the measured responses, the model statistics are said to be valid; otherwise statistical modeling needs to be reperformed.

The whole process of model verification is illustrated in Fig. 4.1.

4.3 ROBUST STATISTICAL MODELING USING HUBER FUNCTION

In statistical modeling, a large number of measurements is required to establish the sample of devices. Errors, either large or small, may occur in the process.

In this section, robust statistical modeling is demonstrated by an example on the Materka FET model [43] with some gross errors. A normal parameter distribution is assumed in the experiment. The experiment was originally carried out by Bandler, Biernacki, Chen, Song, Ye and Zhang (1991) [41], and later by Cai (1992) [34]. It was explored from the point of robust statistical modeling by Bandler, Chen, Biernacki, Madsen, Li Gao and Yu (1993) [13].

The Materka and Kacprzak model [43] is a nonlinear ECM FET model. It utilizes circuit elements (e.g., resistors, capacitors and inductors) connected in a

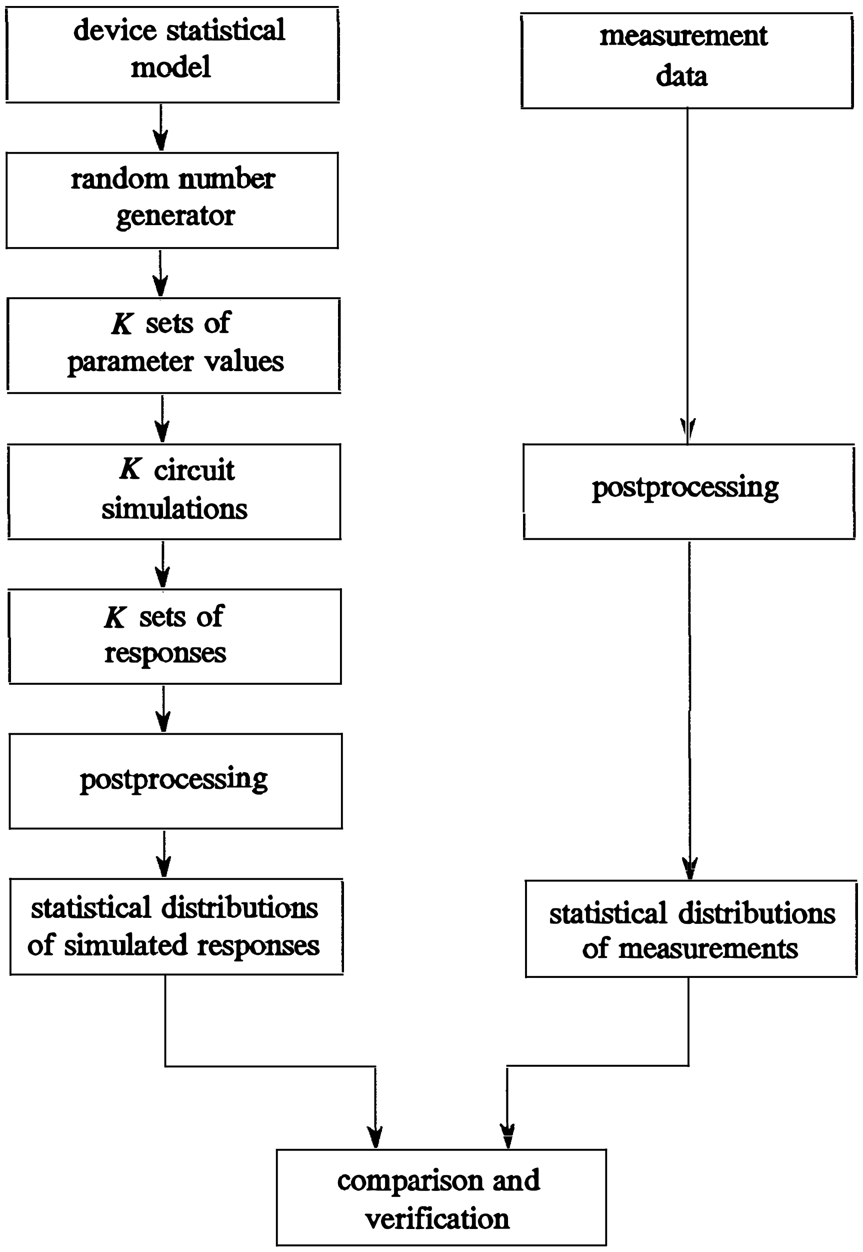
device statistical model
random number generator
K sets of parameter values
K circuit simulations
K sets of responses
postprocessing
statistical distributions of simulated responses
measurement data
postprocessing
statistical distributions of measurements
comparison and verification

Fig. 4.1 Statistical model verification using Monte Carlo simulation [34].

certain topology to represent the MESFET. The model includes twenty-one intrinsic parameters and nine extrinsic parameters. Only a selected number of model parameters are used for demonstration.

The measurements used for statistical modeling include 80 individual devices (data sets) selected from two wafers and were provided by Plessey Research Caswell [47]. Each device represents a 4-finger 0.5 μm GaAs MESFET with equal finger width of 75 μm. Each data set contains small-signal S parameters measured under three different bias conditions and at frequencies from 1 GHz to 21 GHz with an interval of 0.4 GHz between two adjacent frequencies. DC drain bias current is also included in the measurements.

HarPE™ [20] was used to extract the device model parameters. The measurements used for parameter extraction include DC bias currents at three bias points and S parameters for those bias points at frequencies from 1 GHz to 21 GHz with a 2 GHz interval. The linear parameter C_x is fixed at 2 pF for the model. Model parameters for each individual device were extracted by matching simultaneously the DC and small-signal S parameter responses to the corresponding measurements.

The optimizer we used for parameter extraction is the ℓ_2 optimizer available in the HarPE™ program. As we experimented, for this particular example, there is essentially no difference whether we use the ℓ_2 or the Huber methods. However, readers are encouraged to try the Huber methods.

4.3.1 Huber Estimator for Statistical Modeling of Devices [13]

Measurements containing gross measurement errors and/or involving faulty devices result in large deviations in the extracted parameters. For example, consider the run chart shown in Fig. 4.2 of an extracted model parameter, namely the FET time-delay τ. Most of the extracted values of τ are between 2 ps - 2.5 ps, but there are a few abnormal values due to faulty devices and/or gross measurement errors. In fact, the other model parameters extracted from those faulty devices also have abnormal values.

If this is the case, the least-squares estimates for the model statistics will be severely affected, whether obtained by ℓ_2 optimization or by direct evaluation from (4-7,8,9). In the earlier work [32,41] using the ℓ_2 estimator, the abnormal data sets were manually excluded from the statistical modeling process.

The Huber-based optimization can be used to automate and robustize statistical estimation. The Huber function is invoked to penalize the error functions defined in (4-10) and (4-17). Parameter statistics are found after optimization. More specifically, to estimate the mean value of τ, for example, we solve the following Huber-based optimization problem

$$\underset{x}{\text{minimize}} \quad F(x) \triangleq \sum_{i=1}^{80} \rho_k(x - \tau^i) \tag{4-18}$$

where τ^i, $i = 1, 2, \ldots, 80$, comprises the sample of τ's from parameter extraction, and x is the mean at the optimum.

Similarly, we estimate the variance by solving

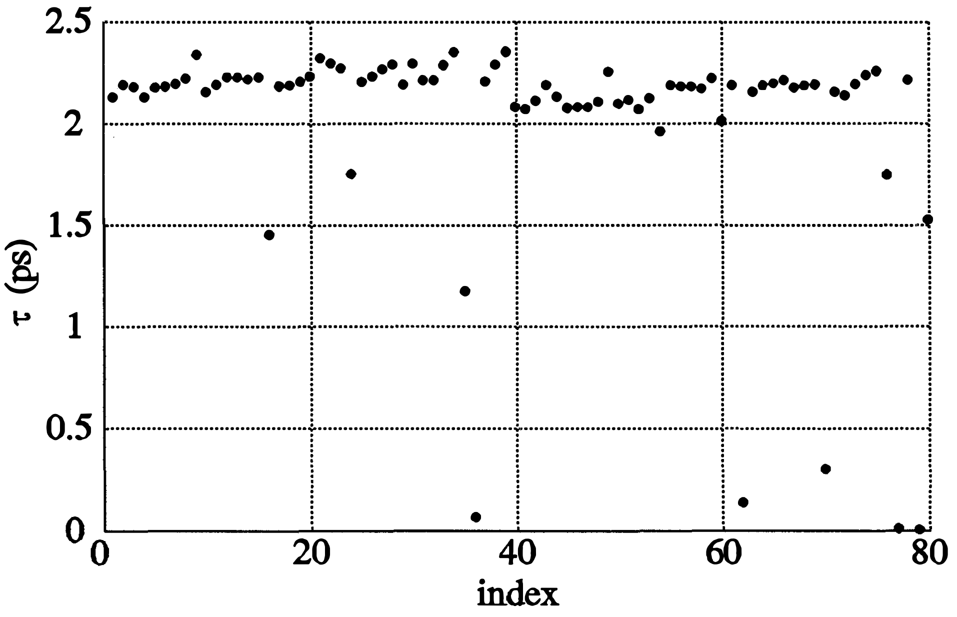

Fig. 4.2 Run chart of the extracted FET time-delay τ [13].

$$\underset{x}{\text{minimize}} \quad F(x) \triangleq \sum_{i=1}^{80} \rho_k(x - (\tau^i - \overline{\tau})^2) \tag{4-19}$$

The term $(\tau^i - \overline{\tau})^2$ in (4-19) can be thought of as the "sample of variances".

The threshold value k should be chosen to reflect the normal spread of the parameter values. For parameter τ, we choose $k = 0.25$ ps to estimate the mean, and $k = 0.0625$ ps^2 to estimate the variance.

Table 4.1 lists the means and standard deviations of a selected number of model parameters we obtained using the ℓ_2 and the Huber estimators. Reliable results can be obtained by manually excluding the abnormal data sets and invoking the ℓ_2 estimators afterwards. Table 4.1 uses the symbol ℓ_2^* to distinguish this approach from the normal ℓ_2.

As expected, the impact of the abnormal data points on the ℓ_2 estimates of the standard deviations is especially severe. These estimates will, by no means, correctly reproduce the distribution of device responses. The ℓ_2^* approach, though it is intuitively reliable, requires manual manipulation of the data and is not appropriate when there are data points which cannot be clearly classified as normal or abnormal. The Huber estimators, on the other hand, provide results close to ℓ_2^* method and yet does not require manual manipulation.

It should also be noted that although ℓ_1 is effective for individual device parameter extraction [10], it is not, in general, suitable for statistical postprocessing. The ℓ_1 estimate (median) depends on the order rather than the actual values of the sample.

TABLE 4.1
ESTIMATED STATISTICS OF SELECTED FET PARAMETERS

Parameter	$\overline{\phi}$ (ℓ_2)	$\overline{\phi}$ (Huber)	$\overline{\phi}$ (ℓ_2^*)	$\sigma_\phi(\ell_2)$	σ_ϕ(Huber)	$\sigma_\phi(\ell_2^*)$
L_G(nH)	0.04387	0.03464	0.03429	94.6%	21.8%	17.4%
G_{DS}(1/KΩ)	1.840	1.820	1.839	28.6%	6.3%	4.9%
I_{DSS}(mA)	47.36	47.53	47.85	14.0%	12.7%	11.3%
τ(ps)	2.018	2.154	2.187	26.3%	5.8%	3.4%
C_{10}(pF)	0.3618	0.3658	0.3696	8.2%	4.6%	3.5%
K_1	1.2328	1.231	1.233	15.5%	10.8%	8.7%

L_G represents the FET gate lead inductance, G_{DS} the drain-source conductance, I_{DSS} the drain saturation current, τ the time-delay, C_{10} and K_1 are parameters in the definition of the gate nonlinear capacitor.

$\overline{\phi}$ and σ_ϕ denote the mean and the standard deviation of the parameter ϕ, respectively.

ℓ_2^* denotes ℓ_2 estimates after 11 abnormal data sets are manually excluded [32,41].

4.3.2 The Selection of Threshold Values for Huber Statistical Estimation

It may have been noticed that, before the Huber function is invoked to penalize the error functions for statistical estimation, a value for the threshold k has to be selected. By engineering intuition, the choice of the threshold should be such that it defines all the statistical variations as "small errors", yet keeps gross errors as "large errors".

We performed the following experiments to test this subject.

Nine different sets of threshold values are attempted to estimate the statistics of the parameter τ (e.g., $k = 0.25$ ps for the mean and $k = 0.25^2$ ps^2 for the corresponding variance), and the results are listed in Table 4.2. It can be seen that the estimated means are very insensitive to various threshold values; even an infinite k does not alter it more than 10%. The estimated standard deviation is also fairly stable within a relatively large range of k. For instance, 10 percent deviation from 2.5, the selected threshold value, produces a deviation in the estimate of less than 10 percent.

Consistent with the theory, when k reaches infinity, the Huber estimates become identical to the ℓ_2 estimates. This justifies the flexibility of the Huber-based method to be a general-purpose optimizer: whenever the traditional least-squares solution is assumed more appropriate, the Huber method is capable of providing it by setting the threshold to a sufficiently large value.

In general, a fixed value of the threshold k defines a band of width $2k$, in which the data points are treated as small errors. The number of small errors varies as the optimizer moves the band.

TABLE 4.2
ESTIMATED STATISTICS FOR DIFFERENT VALUES OF k

k	$\overline{\tau}$	σ_τ
0.15	2.168	4.4%
0.2	2.161	5.1%
0.225	2.157	5.4%
0.25	2.154	5.8%
0.275	2.150	6.2%
0.3	2.147	6.6%
0.5	2.122	9.6%
1	2.079	15.7%
∞	2.018	26.3%

At the Huber solution, let

$$n_{s_j} = \begin{cases} 1 & if\ |e_j| \le k \\ 0 & otherwise \end{cases} \tag{4-20}$$

and

$$N_s = \sum_{j=1}^{K} n_{s_j} \tag{4-21}$$

N_s represents the size of the group of small errors. Clearly, N_s is a function of k. If k approaches zero (the ℓ_1 case), N_s approaches the number of points residing exactly on the solution; on the other hand, if k is sufficiently large, N_s will be K, the total number of devices. Intuitively, if we select two k's, $k_1 \le k_2$, and perform the Huber optimization for each k, we will have two N_s's, with $N_{s1} \le N_{s2}$. This is because a bigger threshold defines a larger band, accommodating more data points.

Figs. 4.3, 4.4 and 4.5 depict N_s versus k for parameters τ, L_G and C_{10}, respectively, where N_s is expressed as a percentage of the total number of devices K. As k increases, N_s grows monotonically. This justifies the foregoing intuition. As seen from the figures, N_s grows very sharply from $k = 0$, and becomes saturated at a certain point. This is due to the fact that most of the extracted parameters have only small statistical variations, concentrating close to the nominal value, while a fairly small number of catastrophic errors are far outside.

The "knee" of the N_s curve, which separates the fast growing range from the saturation range, corresponds to a solution which includes the majority of functions as "small errors". The value of k at the "knee" is consistent with our choice.

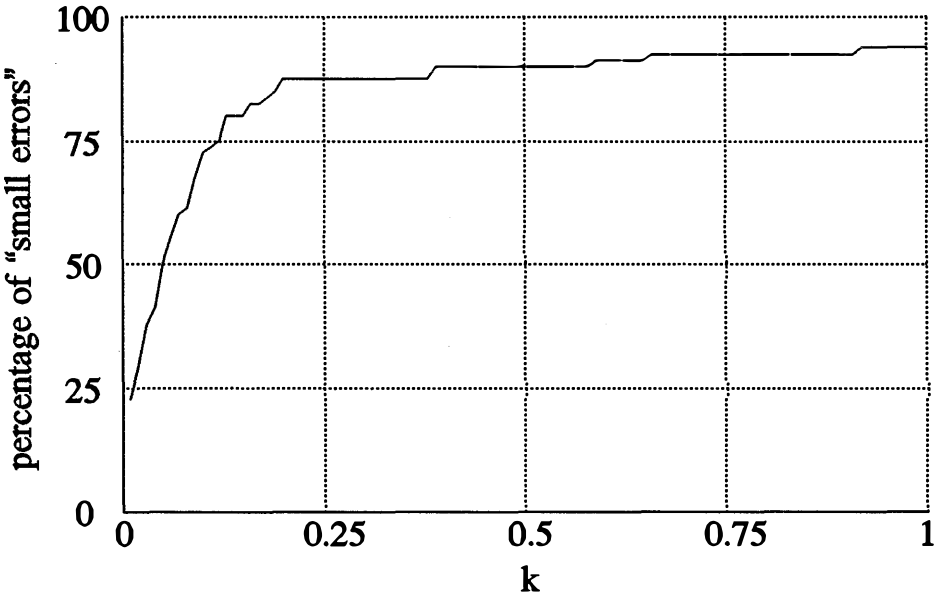

Fig. 4.3 Percentage of "small errors" for the FET time-delay τ versus the threshold k.

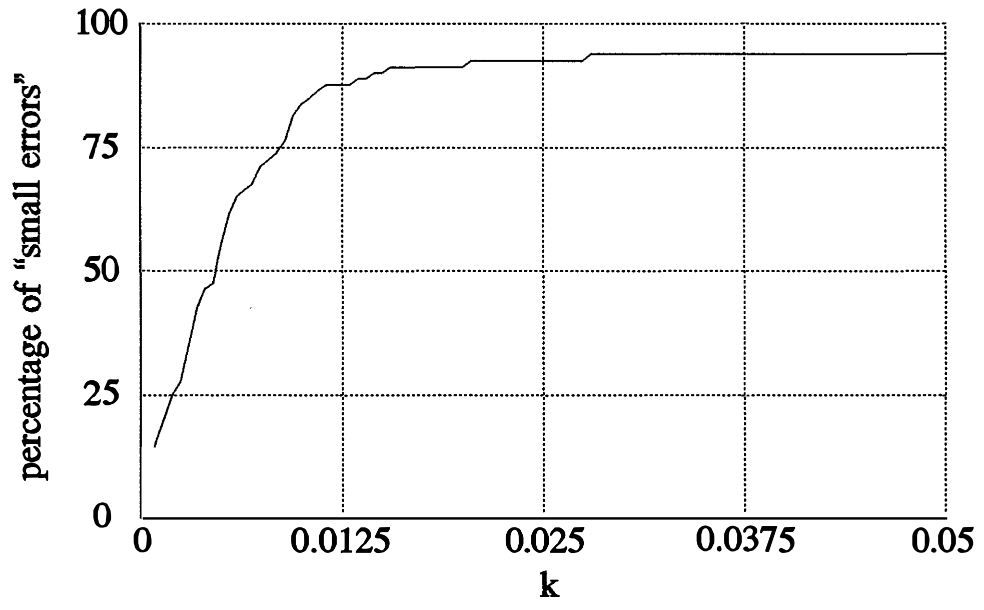

Fig. 4.4. Percentage of "small errors" for the FET gate lead inductance L_G versus the threshold k.

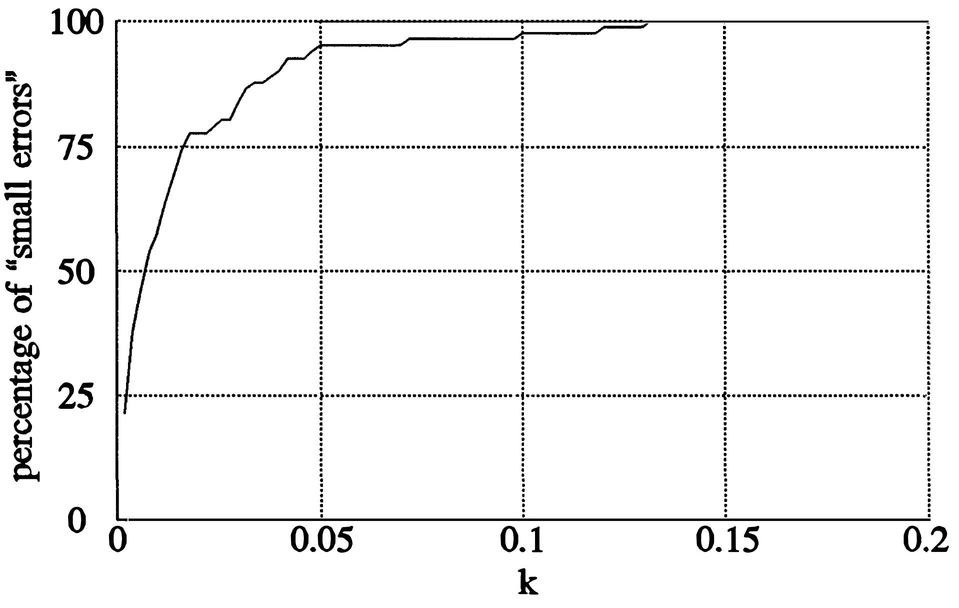

Fig. 4.5. Percentage of "small errors" for the FET model parameter C_{10} versus the threshold k.

4.3.3 Comparison of Dedicated Huber Algorithm and Generic Methods

As a continuation of Subsection 3.4.2, we compared the dedicated Huber algorithm described in Chapter 3 with three other generic methods available in the OSA90/hope™ system, namely, quasi-Newton, conjugate gradient and simplex search, in the context of FET statistical modeling.

The mean value of the FET parameter τ is to be estimated, and the Huber objective function (4-18) has been explicitly incorporated into the generic methods.

In this case, only one variable is involved and all the algorithms under test converged to the correct solution. Table 4.3 lists the number of function evaluations required by each algorithm from four different starting points. It shows that the dedicated EMA is more efficient than the generic ones.

4.4 CONCLUDING REMARKS

In this chapter, we presented statistical modeling using Huber functions. Parameter extraction for individual devices by matching the device responses to the corresponding measurements was addressed. The postprocessing of the extracted model parameters using the Huber estimator for statistical estimation was demonstrated. The selection and verification of the threshold values for Huber-based optimization were discussed. Comparison between the dedicated Huber optimization algorithm and generic methods was also made with the statistical modeling example.

TABLE 4.3
NUMBER OF FUNCTION EVALUATIONS REQUIRED BY DIFFERENT ALGORITHMS FROM OSA90/hope™

Algorithm	Starting Point			
	1.5	2	2.25	3
Dedicated Huber	4	4	4	4
Quasi-Newton	8	5	5	7
Conjugate-Gradient	13	13	11	14
Simplex	26	16	16	24

The optimization problem is to estimate the mean of FET parameter τ using the Huber objective function.

All four algorithms converge to the correct solution.

Experiments were carried out on Huber statistical estimation using the Materka and Kacprzak MESFET model. The Huber estimates are robust against gross errors and do not require manual manipulation of the data. When there is no gross error in the data, the Huber estimator will still be able to provide valid ℓ_2 estimates by setting the threshold value to infinity.

The choice of the threshold for Huber-based optimization is not arbitrary and should reflect the statistical spread of the data. Our choices were verified with illustrations.

A comparison with other generic optimization methods shows, in the context of the Huber-based FET statistical estimation, that the dedicated EMA is more efficient.

Furthermore, mathematical formulation for parameter extraction shows that the Huber function is a promising criterion for extracting parameters of individual devices.

Chapter 5
EFFECTIVE ANALOG FAULT DIAGNOSIS

5.1 INTRODUCTION

The problem of analog circuit fault diagnosis arose in the 1960s [48,49]. Due to progress in the area of electronic design, it has become a flourishing research activity. As Lin and Elcherif (1988) [50] indicated, analog fault diagnosis has become the third aspect on which the study of circuit theory centers.

The ever-increasing complexity of large circuits and systems stimulates intensive research activities, trying to develop effective fault diagnosis techniques. A complete review, as well as comparison with respect to various aspects of the performance, of these methods were given by Bandler and Salama (1985) [51]. A brief summary is given here as follows.

In this context, a fault means any change in the value of an element with respect to its nominal value which can cause the failure of the whole circuit. If this is the case the circuit is considered as faulty.

There are three levels of fault diagnosis, namely, fault detection, fault location and identification, and fault prediction. The main concern of the fault prediction problem is to replace the elements, which are about to fail, before an actual failure occurs and with minimum loss in the lifetime of the replaced elements. Fault detection is the minimum requirement for fault location and identification. It identifies whether a network is faulty. In this chapter, we assume that the circuit under test (CUT) has been identified as faulty, and concentrate only on fault location and identification methods.

Four approaches stand out of various fault location and identification methods. In the fault dictionary approach [50], a fault dictionary is constructed by simulating all possible combination of faults and is stored in the computer. Then the measurements are compared with those prestored circuit outputs, in an effort to identify the faults. It needs extensive off-line computation and mass storage for the fault dictionary.

The parameter identification approach completely solves for the actual parameter values of all the elements. In principle, it can be considered as the inverse process of circuit simulation. In circuit simulation, given the circuit parameters, circuit output is to be found; while in parameter identification, circuit output (the measurements) is given and the parameter values are to be determined, possibly through solving systems of linear/nonlinear equations. Surely, this approach is the best in terms of achieving the goal of fault location and identification. However, sufficient measurements are often unavailable.

The approach of fault verification gained a lot of interest recently. The method is to locate the faulty components without solving the parameter values by simulating the circuit under different conditions and then using some decision algorithm. Compared with the parameter identification techniques, this approach requires fewer measurements.

We consider the problem of analog fault location and identification from an optimization point of view, an approach Bandler and Salama [51] called the approximation techniques. This approach was considered computationally intensive [51]. However, the rapid advances in computer technology made it practical. In fact, fault diagnosis using optimization is extremely effective and powerful in dealing with insufficient measurements. When the parameter values cannot be identified due to an insufficient number of measurements, they can be determined by minimizing a certain measure, and the most likely faulty elements that exhibit large deviation from their nominal values are isolated. In particular, ℓ_2 approximation, quadratic optimization and ℓ_1 approximation have been proposed (see Bandler and Salama [51] and Bandler and Zhang (1988) [52]).

The ℓ_2 approximation technique was proposed under the assumption that all the catastrophic faults have been eliminated or are nonexistent and the circuit failure is due to component drifting out of tolerance (as from aging, temperature changes, etc.) [51,52]. Nevertheless, the method is not robust with respect to deviations of the nonfaulty elements inside their tolerance region.

The scheme of solving a series of quadratic programming problems was proposed by Merrill (1973) [53], based on the hypothesis that the circuit failure is due to one or a few faulty components which have much greater deviations than the rest of the components in the circuit. This hypothesis is consistent with most practical observations.

The ℓ_1 optimization has been successfully applied to analog fault location problems (Bandler, Kellermann and Madsen (1985) [11], and also [51,52]). It takes advantage of both the nature of the ℓ_1 norm and the linearity of the constraints [52].

Some special methods have been developed to support the foregoing analog fault diagnosis techniques. The utilization of different excitations, for example, can increase the number of measurements without increasing the number of accessible nodes (Bandler, Chen and Daijavad (1986) [10]).

We propose attacking the analog fault diagnosis problems using the Huber function. It should fall into the optimization approach category. In this chapter, the effectiveness of Huber-based optimization for fault diagnosis problems is demonstrated and compared with ℓ_1 optimization. Mathematical expressions are formulated for the optimization approach, and are then tailored for Huber-based optimization. A resistive mesh network example is used for illustration and comparison.

5.2 FORMULATION OF FAULT LOCATION PROBLEMS USING OPTIMIZATION [52]

Different approaches for fault diagnosis problems have very different formulations due to the nature of the methods. The mathematical expressions and/or algorithmic descriptions for the approaches summarized in Section 5.1 can be found in [51].

In this section, we formulate fault diagnosis problems as optimization problems, following the notation of [52]. For both conceptual and notational simplicity, the formulation uses only single frequency measurements.

Suppose the set of measurements obtained from the circuit under test (CUT) is denoted by

$$\boldsymbol{S} = [S_1\ S_2\ \dots\ S_m]^T \tag{5-1}$$

The circuit has n parameters

$$\boldsymbol{\phi} = [\phi_1\ \phi_2\ \dots\ \phi_n]^T \tag{5-2}$$

The circuit response corresponding to $\boldsymbol{S}$ is represented by

$$\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{\phi}, \omega) \tag{5-3}$$

where ω denotes frequency. In the case of a single frequency, ω is implied and $\boldsymbol{R}$ can be simplified as

$$\boldsymbol{R} = \boldsymbol{R}(\boldsymbol{\phi}) \tag{5-4}$$

Ordinarily, a faulty circuit contains only a few faults and possibly many small tolerances for the rest of the elements. For a circuit with its nominal design characterized by $\boldsymbol{\phi}^0$ and $\boldsymbol{R}^0$, the measurements taken from the actual (faulty) configuration can be expressed as

$$\boldsymbol{S} = \boldsymbol{R}(\boldsymbol{\phi}^0 + \Delta\boldsymbol{\phi}) \tag{5-5}$$

However, as experienced by many engineers, the measurements $\boldsymbol{S}$ are usually insufficient to identify all the actual parameter values $\boldsymbol{\phi} = \boldsymbol{\phi}^0 + \Delta\boldsymbol{\phi}$, let along their deviations from the nominal values $\Delta\boldsymbol{\phi}$. In other words, equation (5-5) is underdetermined. Therefore, optimization procedures are needed to find the most likely parameter values among an infinite number of solutions to (5-5).

Such an optimization procedure can be formulated as

$$\underset{\Delta\boldsymbol{\phi}}{\textit{minimize}}\quad U(\Delta\boldsymbol{\phi}) \tag{5-6}$$

subject to

$$\boldsymbol{R}(\boldsymbol{\phi}^0 + \Delta\boldsymbol{\phi}) - \boldsymbol{S} = \boldsymbol{0} \tag{5-7}$$

where U is an increasing scalar function of $|\Delta\boldsymbol{\phi}|$ and $\boldsymbol{R}(\boldsymbol{\phi}^0 + \Delta\boldsymbol{\phi})$ are the calculated responses from the current value of ϕ. Equation (5-7) is called the constraint equation.

To simplify the problem (5-6,7), two effective formulations, namely, the current/voltage source substitution model and the component connection model, have been proposed to transform the constraint equation into a system of linear equations. Representing the parameter changes by equivalent current/voltage sources, the current/voltage source substitution model [52] transforms the problem (5-6,7) into solving optimization with linear constraints. The scheme is to solve for those additional current/voltage sources, and then find the parameter value changes accordingly.

A complete description of the treatment of the component connection model can be found in [54-56].

The foregoing general formulation (5-6,7) has been implemented in the ℓ_1 sense based on an exact penalty function as follows [11]:

$$\underset{\phi}{\textit{minimize}} \sum_{j=1}^{n+m} |e_j(\phi)| \tag{5-8}$$

where the error functions are defined as

$$\begin{aligned} e_i(\phi) &= \Delta\phi_i/\phi_i^0, \quad i = 1, 2, ..., n \\ e_{n+i}(\phi) &= \beta_i(R_i - S_i), \quad i = 1, 2, ..., m \end{aligned} \tag{5-9}$$

where

$$\phi = \phi^0 + \Delta\phi \tag{5-10}$$

and β_i, $i = 1, 2, ..., m$, are appropriate multipliers for the penalty terms.

Due to the nature of the ℓ_1 norm, this approach is very effective in isolating the faulty elements [11].

5.3 EFFECTIVE FAULT DIAGNOSIS USING THE HUBER FORMULATION

The approach of fault diagnosis using Huber functions was first proposed by Bandler, Chen, Biernacki, Madsen, Li Gao and Yu [13].

The effectiveness of the ℓ_1 approach in fault diagnosis comes from the fact that the ℓ_1 solution is robust against a few large error functions. The similar robustness of Huber-based optimization (see Chapter 2 for details) leads to the idea of invoking Huber-based optimization for locating the possible faults, based on the same assumption that the faults in the CUT exhibit much bigger (relative) deviations from the nominal values than the nonfaulty elements.

5.3.1 A Huber Formulation for Fault Isolation Problems

The formulation of using the Huber function for analog circuit fault location and identification is very similar to that using the ℓ_1 norm (5-8,9,10). It is as follows:

$$\underset{\phi}{minimize} \sum_{j=1}^{n+m} \rho_k(e_j(\phi)) \tag{5-11}$$

where e_i, $i = 1, 2, ..., n+m$, and ϕ are given in (5-9) and (5-10), respectively, and ρ_k is the Huber function defined by (2-12).

Clearly, compared with (5-8), the modulus of the error functions are replaced by the Huber function ρ_k.

5.3.2 The Resistive Mesh Network Example

We use a resistive mesh network to examine the applicability of the Huber-based optimization to solving analog fault location problems. The same fault location example was used by Bandler and Salama [51] for demonstrating the network decomposition approach. It was experimented on by Bandler, Chen, Biernacki, Madsen, Gao and Yu [13] for the Huber-based approach.

The resistive network under test is shown in Fig. 5.1 [11,13,57]. The nominal element values are $G_i = 1.0$ with tolerances $\epsilon_i = \pm 0.05$, $i = 1, 2, ..., 20$. Node 12 is taken as the reference node, and nodes 4, 5, 8 and 9 are assumed to be internal and inaccessible for measurement. The voltage measurements at the other nodes are used for fault location.

The actual parameter values of a faulty network are listed in Table 5.1. Two faults are assumed in the circuit, namely G_2 and G_{18}, with relative error to be -50%. This situation is consistent with the assumption that faults are relatively much larger.

A DC current source of 1A is applied to node 1 as the excitation to the network. Instead of performing the real experiment, the voltage measurement data $\boldsymbol{S}$ is obtained by circuit simulation using the actual parameter values. The nominal parameter values are used as the starting point for optimization.

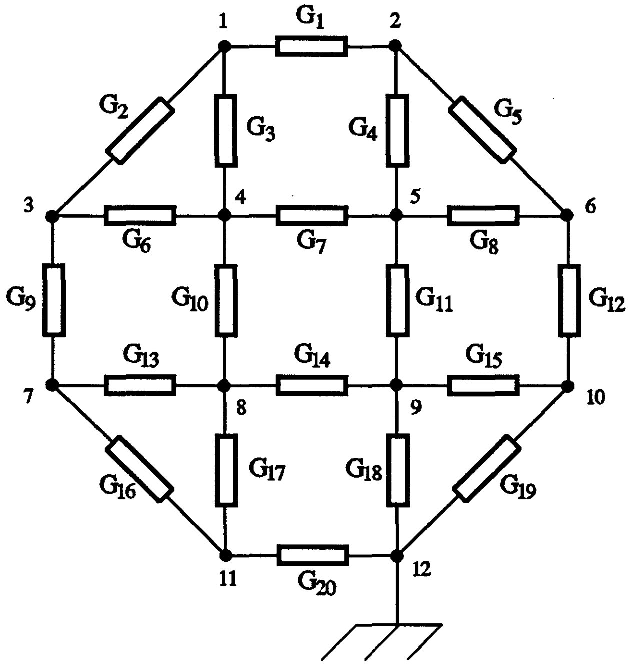

Fig. 5.1 The resistive mesh circuit [11,13,57].

TABLE 5.1
FAULT LOCATION OF THE RESISTIVE MESH CIRCUIT

Element	Nominal Value	Actual Value	Percentage Deviation		
			Actual	ℓ_1	Huber
G_1	1.0	0.98	-2.0	0.00	-0.11
G_2	1.0	0.50	-50.0*	-48.89	-47.28
G_3	1.0	1.04	4.0	0.00	-2.46
G_4	1.0	0.97	-3.0	0.00	-1.18
G_5	1.0	0.95	-5.0	-2.70	-3.16
G_6	1.0	0.99	-1.0	0.00	-0.06
G_7	1.0	1.02	2.0	0.00	-0.19
G_8	1.0	1.05	5.0	0.00	-0.41
G_9	1.0	1.02	2.0	2.41	3.75
G_{10}	1.0	0.98	-2.0	0.00	0.39
G_{11}	1.0	1.04	4.0	0.00	-0.37
G_{12}	1.0	1.01	1.0	2.73	1.32
G_{13}	1.0	0.99	-1.0	0.00	-0.26
G_{14}	1.0	0.98	-2.0	0.00	-0.50
G_{15}	1.0	1.02	2.0	0.00	-0.05
G_{16}	1.0	0.96	-4.0	-3.36	-2.67
G_{17}	1.0	1.02	2.0	0.00	-0.61
G_{18}	1.0	0.50	-50.0*	-50.09	-47.33
G_{19}	1.0	0.98	-2.0	-1.41	-3.81
G_{20}	1.0	0.96	-4.0	-4.40	-4.72

* Faults

The threshold k for the Huber function is chosen as 0.05, commensurate with the tolerances of the elements. The penalty multipliers β_i in (5-9) are set to 1000, sufficiently large to ensure that the nonlinear constraints (circuit equations) are satisfied. The results from the ℓ_1 optimization and Huber optimization are compared in Table 5.1.

As can be seen in Table 5.1, from both ℓ_1 and Huber optimizations, the faulty parameters yield very large relative deviations, while the nonfaulty ones have very small deviations. Therefore, the faults can be found by comparing the percentage variations with a preset threshold value. In this case, for example, the threshold value could be set to $\pm 10\%$.

Misleadingly, the ℓ_1 solution has a fairly big group of elements which exhibit zero deviations. This is because, as pointed out in Chapter 2, the solution is dictated by these points. In the Huber solution, it is rarely the case since the Huber function treats the small error functions in the least-squares sense.

We tested this example for 4 other different starting points. The Huber approach correctly located the faults in all the cases. The ℓ_1 method was successful in 3 of the cases, but failed in one of the cases.

5.4 CONCLUDING REMARKS

We have investigated the applicability of the Huber-based optimization to analog circuit fault diagnosis problems. General formulations of the problems have been introduced, and transformed for the use of Huber-based optimization. A resistive mesh network example has been used for demonstrating the effectiveness of the Huber approach. Comparison has been made with the conventional ℓ_1 method.

The approach of fault location using optimization has a big advantage in dealing with insufficient measurements. Due to the robustness of Huber-based optimization, it is effective in solving fault diagnosis problems. As demonstrated by the resistive mesh example, the Huber method appears to be as effective as the conventional ℓ_1 approach. Moreover, the Huber approach seems to be less influenced by different starting points.

Chapter 6

EFFICIENT PREPROCESSING FOR LARGE-SCALE MULTIPLEXER DESIGN

6.1 INTRODUCTION

As technology advances, system designers are no longer dealing with simple circuit topologies and small numbers of elements and parameters. From an easy-to-use home electrical appliance to a more sophisticated communication satellite, today's real electrical systems may contain enormous numbers of components, with thousands of parameters to be determined and adjusted during design and manufacturing.

This obviously poses a serious challenge to CAD systems in handling large-scale design problems. As Bandler and Zhang (1987) [58] pointed out, the reasons of failure are mainly the prohibitive computer storage and CPU time required, and the inclination towards an ill-conditioned problem. Special techniques, specifically decomposition methods (see, e.g., Himmelblau (1973) [59], Sangiovanni-Vincentelli, Chen and Chua (1977) [60], and Bandler and Zhang (1987) [58]), have been proposed to adapt the existing CAD techniques to solving large-scale problems.

The active research activities in the area of microwave filter and multiplexer design (Atia (1974) [61]) were motivated by the fast-growing satellite communications technology (Kudsia (1982) [62]).

In order to communicate efficiently, the signals transmitted are usually preprocessed at the transmission end and postprocessed at the receiving end. In preprocessing, the source signals are modulated with different carrier frequencies so that different information will be carried in different frequency bands called channels. After the modulation, those separate channels are summed up to be contained in a broader band to avoid energy leakage among the input channels. At the receiving end of the transmission, the signals are retrieved by, firstly, separating the channels from the broader band, and secondly, demodulating for each channel.

Microwave multiplexers are used in combining channels to the broader band, and inversely, separating the channels from the broader band (Cristal and Matthaei (1964) [63]).

The scale of the multiplexer design problem is usually very large. To give a rough idea, a typical 16-channel 12-GHz multiplexer design [58] involves 240 variables and 399 nonlinear functions. Such dimensionality is prohibitive to both computer-oriented optimization and manual post-production tuning. Automatic decomposition techniques have, therefore, been developed (e.g., see, Bandler and Zhang [58]) to divide the entire problem into a series of smaller-scale suboptimization problems. Exact simulation and sensitivity analysis of multiplexing networks was also proposed by Bandler, Daijavad and Zhang (1986) [64], facilitating the use of gradient-based optimization methods.

This chapter deals with design of multiplexers using one-sided Huber function for preliminary optimization. The multiplexer under consideration is a contiguous band multiplexer consisting of multi-coupled cavity filters. In Section 6.2, we provide a review of simulation and sensitivity analysis of multiplexing networks required by design optimization. A multiplexer design example is demonstrated in Section 6.3. One-sided Huber-based optimization is invoked as a preparatory optimization. Comparison is made with minimax optimization in terms of providing a good starting point for full-scale design optimization.

6.2 EXACT SIMULATION AND SENSITIVITY ANALYSIS OF MULTIPLEXING NETWORKS [64,65]

Modern gradient-based optimization techniques require robust and efficient methods for circuit simulation and sensitivity analysis. For multiplexing network simulation, Bandler, Daijavad and Zhang [64,65] proposed a novel approach, utilizing the concept of cascade analysis. We provide a brief review of the method.

6.2.1 The Overall Configuration of a Multiplexer [64]

A typical equivalent circuit for a multiplexer is shown in Fig. 6.1. As is seen, there are N branches (channels) distributed along a waveguide manifold. Each branch consists of a multi-coupled cavity filter, input-output transformers, and an impedance inverter. A waveguide section separates two adjacent channels, and a nonideal series junction connects the branch with the manifold. Without losing generality, the main cascade (the manifold) is terminated by a short circuit and the

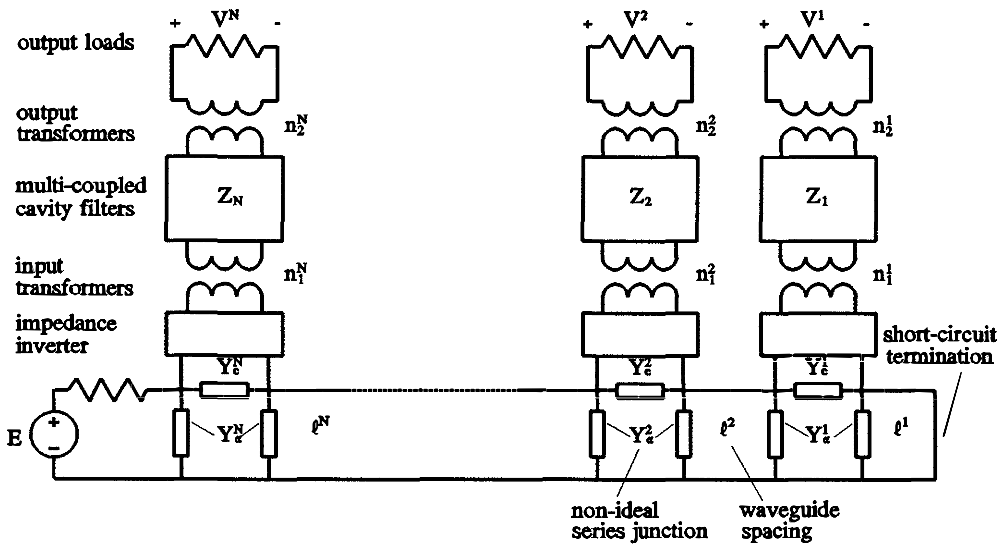

Fig. 6.1 Equivalent circuit of a contiguous band multiplexer. Each channel has a multi-coupled cavity filter with input and output transformers as well as impedance inverter. The main cascade is a waveguide manifold with a short-circuit termination. Branches are connected to the main cascade through nonideal series junctions [64].

individual channel output load is 50 Ω.

6.2.2 Cascade Analysis [64,65]

The method proposed in [64,65] utilizes cascade analysis which involves transmission matrices. For a two-port network shown in Fig. 6.2, the transmission matrix is a 2x2 matrix $\boldsymbol{Q}_{12}$ so that the following must hold

$$\begin{bmatrix} V_1 \\ I_1 \end{bmatrix} = \boldsymbol{Q}_{12} \begin{bmatrix} V_2 \\ -I_2 \end{bmatrix} \tag{6-1}$$

In a cascade of two-port networks, the equivalent transmission matrix between the reference planes i and j, denoted as $\boldsymbol{Q}_{ij}$, can be obtained from forward and reverse analysis.

In a forward (reverse) analysis, $\boldsymbol{Q}_{ij}$ is computed by initializing row vectors $\boldsymbol{e}_1^T$ and $\boldsymbol{e}_2^T$ (column vectors $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$) at reference plane $i(j)$ and successively premultiplying (postmultiplying) each transmission matrix by the resulting row (column) vector until reference plane $j(i)$ is reached. $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are unit vectors given by $[1 \quad 0]^T$ and $[0 \quad 1]^T$, respectively.

For example, the first element A_{ij} of the equivalent transmission matrix

$$\boldsymbol{Q}_{ij} = \begin{bmatrix} A_{ij} & B_{ij} \\ C_{ij} & D_{ij} \end{bmatrix} \tag{6-2}$$

can be calculated by [65]

$$A_{ij} = \boldsymbol{e}_1^T \boldsymbol{Q}_{i-1} \boldsymbol{Q}_{i-2} \cdots \boldsymbol{Q}_l \cdots \boldsymbol{Q}_{j+1} \boldsymbol{Q}_j \boldsymbol{e}_1 \tag{6-3}$$

where $\boldsymbol{Q}_l$, $l = j, j+1, \ldots, i-1$, represents the two-port networks between the ith and jth reference planes.

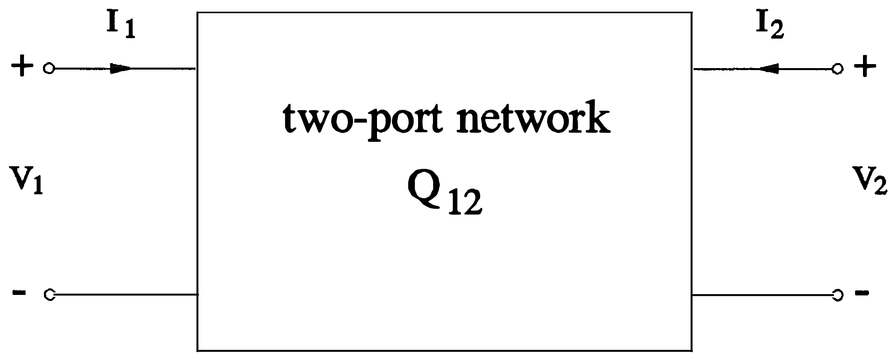

Fig. 6.2 Block representation of a two-port network characterized by its transmission matrix Q_{12}.

Equation (6-3) can be rewritten as

$$A_{ij} = (\boldsymbol{u}_1^{i,l+1})^T \boldsymbol{Q}_l \boldsymbol{v}_1^{lj} \tag{6-4}$$

where

$$(\boldsymbol{u}_1^{i,l+1})^T = \boldsymbol{e}_1^T \boldsymbol{Q}_{i-1} \boldsymbol{Q}_{i-2} \cdots \boldsymbol{Q}_{l+1} \tag{6-5}$$

is a partial forward analysis and

$$\boldsymbol{v}_1^{l,i} = \boldsymbol{Q}_{l-1} \cdots \boldsymbol{Q}_{j+1} \boldsymbol{Q}_j \boldsymbol{e}_1 \tag{6-6}$$

is a partial reverse analysis.

Consequently, the sensitivity formulation of the transmission matrix w.r.t. parameter ϕ is readily available. For example,

$$\begin{aligned} \frac{\partial A_{ij}}{\partial \phi} &= \sum_{l=j}^{i-1} [\boldsymbol{e}_1^T \boldsymbol{Q}_{i-1} \cdots \boldsymbol{Q}_{l+1} \frac{\partial \boldsymbol{Q}_l}{\partial \phi} \boldsymbol{Q}_{l-1} \cdots \boldsymbol{Q}_j \boldsymbol{e}_1] \\ &= \sum_{l \in \boldsymbol{I}_\phi} \frac{\partial A_{ij}^l}{\partial \phi} \end{aligned} \tag{6-7}$$

where

$$\frac{\partial A_{ij}^l}{\partial \phi}$$

is the result of a forward or reverse analysis between reference planes i and j with the lth matrix replaced by its derivative w.r.t. ϕ, and $\boldsymbol{I}_\phi$ is an index set whose elements identify the transmission matrices containing ϕ. Second-order sensitivities can be obtained in a similar manner.

Therefore, if the reference planes i and j are set to the appropriate ports under consideration, we will be able to find the response of interest as well as its sensitivities. For instance, to find the output voltage V^N of the Nth branch (see Fig.

6.1), we set i and j to the input port and the Nth output port, respectively.

For the 3-port nonideal junctions along the main cascade (see Fig. 6.1), Bandler, Daijavad and Zhang [64,65] proposed a scheme for reduction from the 3-port structure to a 2-port representation.

The transmission matrices for various components of the multiplexer structure were listed by Bandler, Daijavad and Zhang [64], were utilized in their algorithm [64] to calculate the multiplexer responses and their sensitivities.

6.3 DESIGN OF A 5-CHANNEL 12 GHz MULTIPLEXER

We present a design of a 5-channel 12 GHz contiguous band multiplexer as follows.

There are five 6th-order multicavity filters mounted on the waveguide manifold, one for each channel. Variables for each channel include six coupling parameters, six cavity resonances, two input and output transformer ratios, and the distance measure from the channel filter to the short-circuit main cascade termination.

The overall problem involves 75 variable and 124 nonlinear functions. The common port return loss and individual channel insertion loss are the circuit responses under consideration. Design optimization involves as an indispensable part of the procedure. This design example has been used to demonstrate an automatic decomposition technique (Bandler and Zhang [58]) and an efficient quadratic approximation for statistical design (Biernacki, Bandler, Song and Zhang (1989) [66]).

6.3.1 Formulation of the Problem [67]

The overall design optimization is a minimax problem, and the objective function to be minimized is given by [67]

$$F(\boldsymbol{\phi}) = \max_{j \in J} e_j(\boldsymbol{\phi}) \tag{6-8}$$

where $\boldsymbol{\phi}$ is a vector of design parameters and J is an index set. The error function $e_j(\boldsymbol{\phi})$, $j \in J$, can be of the form

$$w_{Uk}^1(\omega_i)(R_k^1(\boldsymbol{\phi},\omega_i) - S_{Uk}^1(\omega_i)) \tag{6-9}$$

$$-w_{Lk}^1(\omega_i)(R_k^1(\boldsymbol{\phi},\omega_i) - S_{Lk}^1(\omega_i)) \tag{6-10}$$

$$w_U^2(\omega_i)(R^2(\boldsymbol{\phi},\omega_i) - S_U^2(\omega_i)) \tag{6-11}$$

$$-w_L^2(\omega_i)(R^2(\boldsymbol{\phi},\omega_i) - S_L^2(\omega_i)) \tag{6-12}$$

where $R_k^1(\boldsymbol{\phi},\omega_i)$ is the insertion loss for the kth channel at the ith frequency, $R^2(\boldsymbol{\phi},\omega_i)$ is the return loss at the common port at the ith frequency, $S_{Uk}^1(\omega_i)$, $S_{Lk}^1(\omega_i)$, $S_U^2(\omega_i)$, $S_L^2(\omega_i)$ are the respective specifications, and w_{Uk}^1, w_{Lk}^1, w_U^2, w_L^2 are the arbitrary user-chosen nonnegative weighting factors.

For such a large-scale problem, one usually wishes to optimize a small number of dominant variables in order to obtain a good starting point for the full-scale minimax optimization. We demonstrate that the one-sided Huber-based optimization provides better results in the preparatory optimization than minimax optimization.

6.3.2 The Experimental Procedure

We first try to optimize a small number of dominant variables. We know that the multiplexer responses are highly sensitive to the spacing lengths which are initially set to half the wavelength corresponding to the channel center frequencies. We select the spacings and the channel input transformer ratios (10 variables) and consider a lower specification of 20 dB on the common port return loss. The common port return loss and individual channel insertion loss responses at the starting point are shown in Fig. 6.3.

The minimax solution for preliminary optimization is shown in Fig. 6.4 and the one-sided Huber solution is shown in Fig. 6.5. The worst-case errors in these two figures are similar. Since the worst-case errors cannot be further reduced by changing the selected variables, the minimax optimizer gains nothing from directing effort elsewhere. Using one-sided Huber optimization, on the other hand, we were able to obtain a good starting point for subsequent optimization. As can be seen from the figures, one-sided Huber optimization further reduced the common port return loss in the frequency range of the first two channels, compared to the minimax solution.

The one-sided Huber optimization took 28 min on a SUN SPARCstation 1+.

From the solution shown in Fig. 6.5, we increase the number of variables from 10 to 45, include an upper specification of 2 dB on the channel insertion loss, and restart the one-sided Huber optimization for better results shown in Fig. 6.6. Then a minimax optimization with the full set of 75 variables is performed, resulting in the multiplexer responses shown in Fig. 6.7.

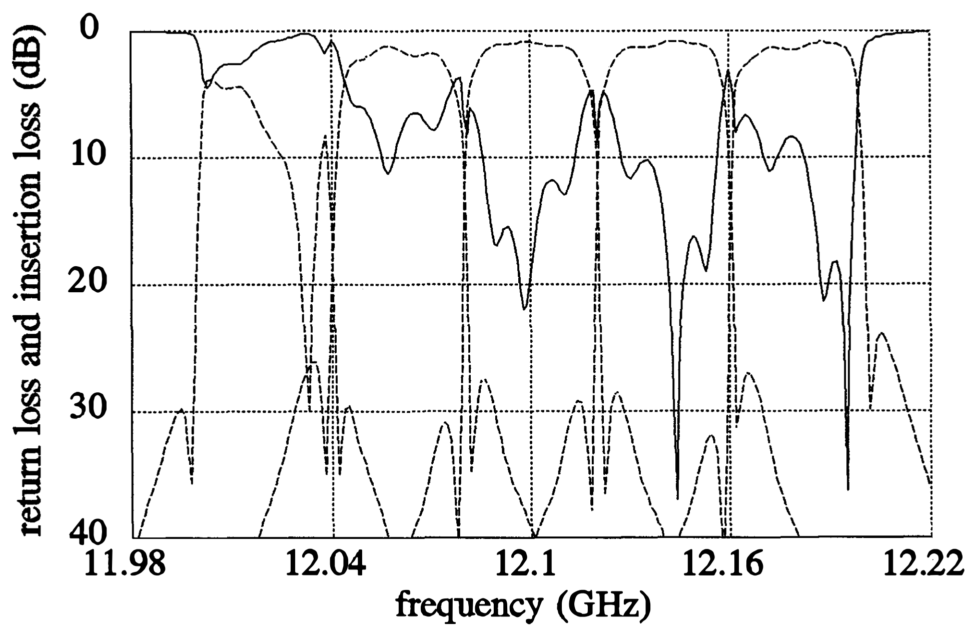

Fig. 6.3 Multiplexer responses at the starting point, showing the common port return loss (————) and the individual channel insertion losses (------).

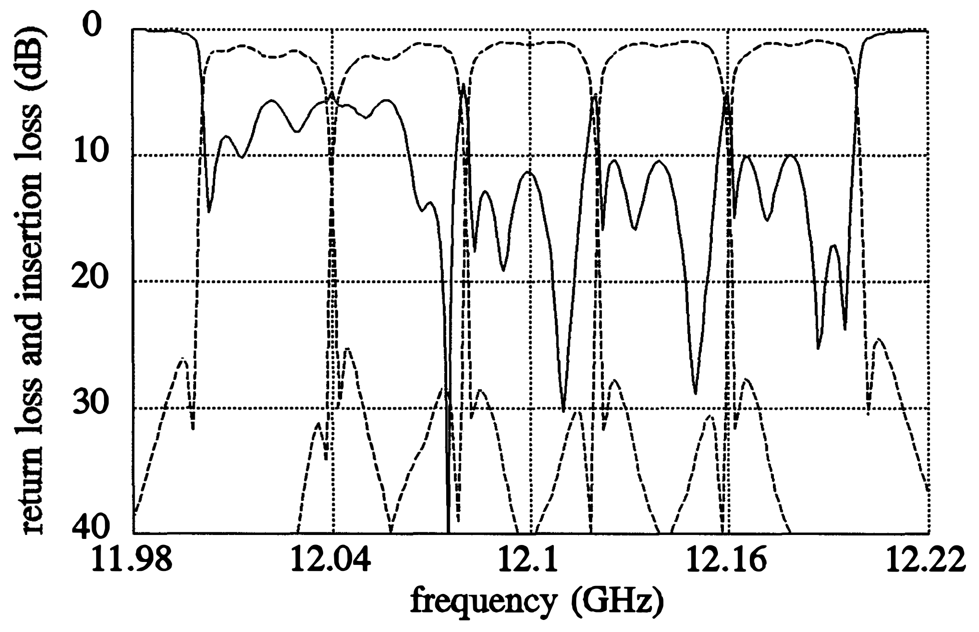

Fig. 6.4 Multiplexer responses after the minimax optimization with 10 variables: spacings and channel input transformer ratios; the common port return loss (————) and the individual channel insertion losses (------). This result hardly improved upon the starting point shown in Fig. 6.3.

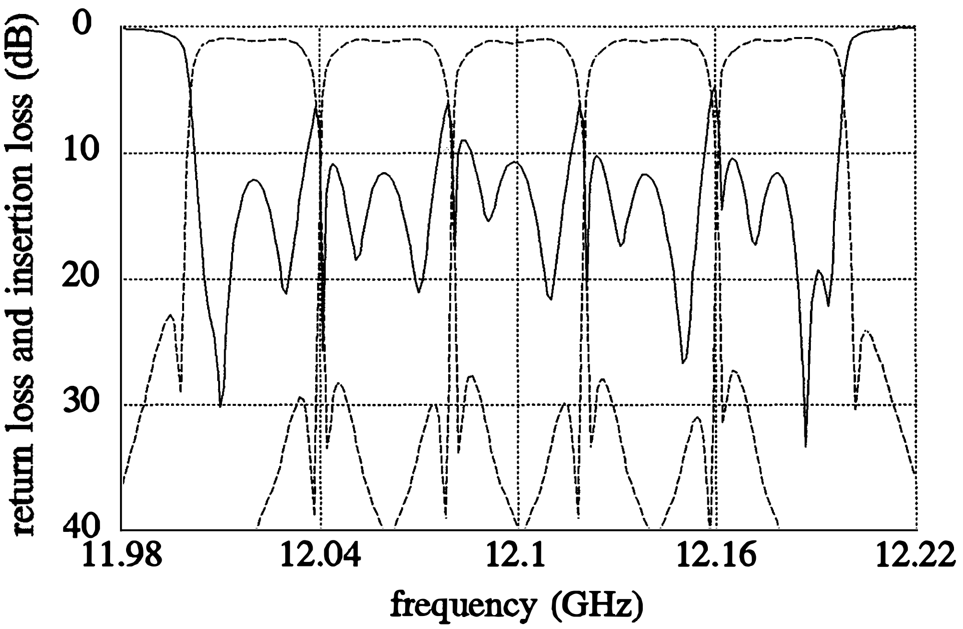

Fig. 6.5 Multiplexer responses after the one-sided Huber optimization with 10 variables: spacings and channel input transformer ratios; the common port return loss (————) and the individual channel insertion losses (------). This result is significantly better than the minimax solution shown in Fig. 6.4.

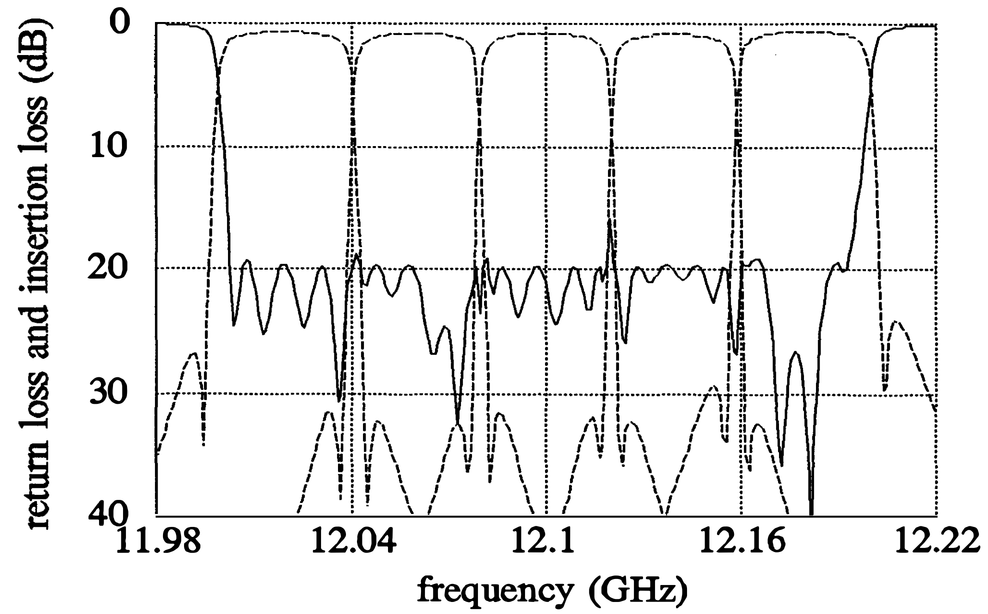

Fig. 6.6 Multiplexer responses after the one-sided Huber optimization with 45 variables, showing the common port return loss (———) and the individual channel insertion losses (------).

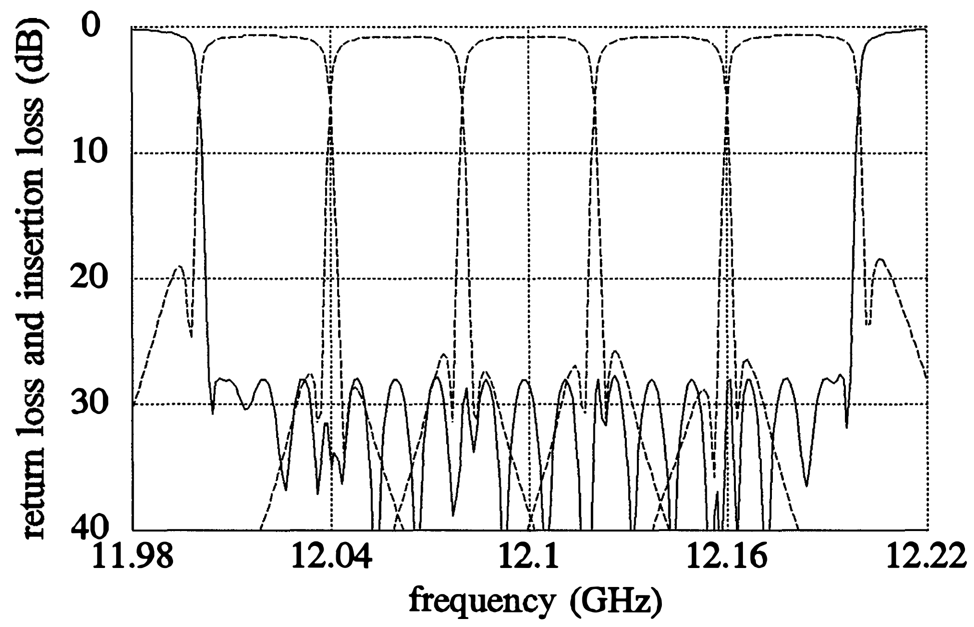

Fig. 6.7 Multiplexer responses after the minimax optimization with the full set of 75 variables, showing the common port return loss (———) and the individual channel insertion losses (------).

6.4 CONCLUDING REMARKS

We have demonstrated the advantage of using the one-sided Huber function in preparatory optimization for large-scale multiplexer design problems. A brief review has been given to the simulation and sensitivity analysis of multiplexing network. A microwave multiplexer design example has shown that the one-sided Huber approach provides a better starting point for the final full-scale optimization, compared to the minimax method. The one-sided Huber optimization seems to be more powerful in reducing nonmaximum errors, which are, by definition, not taken into consideration in the minimax method.

Chapter 7
CONCLUSIONS

In this thesis we have presented a novel robust approach to circuit design, modeling and diagnosis, using Huber functions.

In Chapter 2, we have formulated the Huber-based optimization problems. We have explored the robustness of the Huber approach by investigating the gradients and Hessians of the Huber objective functions. Through a data fitting example, we have shown that the Huber solution is robust against gross errors, and yet has the discriminatory power to model small statistical variations in the data. We have created a one-sided Huber-based formulation as an extension for engineering design problems.

Dedicated Ekblom Madsen algorithms for Huber-based optimization have been reviewed in Chapter 3. The first-stage trust-region method has been described in great detail. It has been shown that the EMA's utilize the special structures of the Huber objective functions and have finer and more accurate adjustment than conventional Newton methods. The second-stage quasi-Newton method has been presented with a BFGS implementation. The overall dedicated optimization algorithm has been described, for both one- and two-sided problems. Comparison with other generic methods has shown that the dedicated EMA is more efficient and appropriate for Huber-based optimization.

We have applied the Huber concept to three areas of circuit design optimization and modeling.

The Huber measure has been proposed for both statistical parameter extraction and estimation in Chapter 4. We have demonstrated through a MESFET statistical modeling example that the Huber statistical estimator is robust against gross measurement errors and/or faulty devices, and yet does not require manual manipulation of data. We have discussed that the threshold value for Huber-based optimization should reflect the statistical spread of the data, and, by experiment, we have justified our choice of the threshold.

In Chapter 5, the Huber function has been formulated for use in analog circuit fault diagnosis problems through an exact penalty function approach. By a resistive mesh circuit example, it has been shown that the Huber-based optimization is as effective as the ℓ_1 method in locating possible faults.

We have presented a large-scale microwave multiplexer design problem in Chapter 6. The one-sided Huber-based optimization has been employed to prepare a starting point for full-scale minimax optimization. In comparison with the preliminary minimax optimization, the starting point obtained by the one-sided Huber approach has further reduced the nonmaximum errors.

In this thesis, we have introduced the unique Huber concept and presented novel results for analog circuit CAD. We have demonstrated that the Huber concept is consistent with practical engineering intuition. It should have a profound impact on modeling, design, fault diagnosis and statistical processing of circuits and devices.

It could be a very interesting topic for future projects to investigate the performance of the one-sided Huber-based optimization in statistical/yield-driven design.

To fully automate the Huber process, an automatic threshold selection routine is needed. Gao Li and Madsen [16] proposed a nonlinear algorithm which can estimate the threshold k at the same time as it converges toward the solution to the Huber problem. However, the ratio between the numbers of "small" and "large" errors has to be entered as a prerequisite. Some knowledge on thresholding in the signal processing area might be useful in this respect.

Appendix

CIRCUIT AND DATA FILES, PROGRAMS

In preparing the examples used in the thesis, we have used the OSA90/hope™ and HarPE™ programs. The circuit files are in the OSA90/hope format.

A CIRCUIT FILE FOR THE DATA FITTING EXAMPLE

```
!  Example huber1_4.ckt
!  illustrate the robustness of Huber optimization
!  data fitting in the presence of both large and small errors
!  compare L1, L2 and Huber solutions for presentation

Control
   Two_sided_perturbation;
end

Expression
   !  data sampled from sqrt(t) without perturbation

   Data[50] = [
      0.1414  0.2000  0.2449  0.2828  0.3162  0.3464  0.3742  0.4000  0.4243  0.4472
      0.4690  0.4899  0.5099  0.5292  0.5477  0.5657  0.5831  0.6000  0.6164  0.6325
      0.6481  0.6633  0.6782  0.6928  0.7071  0.7211  0.7348  0.7483  0.7616  0.7746
      0.7874  0.8000  0.8124  0.8246  0.8367  0.8485  0.8602  0.8718  0.8832  0.8944
      0.9055  0.9165  0.9274  0.9381  0.9487  0.9592  0.9695  0.9798  0.9899  1.0000
   ];

   !  specify the data points to which errors to be added

   perturb[50] = [
       0   -1    0   -1    0   -1    0   -1    0    1
       0   -1    0   -1    0   -1    0   -1    0    1
       0   -1    0   -1    0   -1    0   -1    0    1
       0   -1    0   -1    0   -1    0   -1    0    1
       0   -1    0   -1    0   -1    0   -1    0    1
   ];

   K: 1;                 !  index

   goal = if (perturb[K] > 0) (0)              ! large errors introduced to these points
          else (data[K] - perturb[K] * 0.04);  ! small variations are added

   t = K * 0.02;         !  scale down the interval to (0.02,1)

   x[3,4] = [?11.28?   ?43.5602? ?35.5686? ?18.3375?             ! L1 solution
             ?45.2667? ?564.122? ?280.427? ?429.965?             ! L2 solution
```

```
            ?13.1155? ?62.9922? ?45.0118? ?29.6621?];           ! Huber solution

   I: 1;         ! I=1 is L1; I=2 is L2; I=3 is Huber

   x1 = x[I,1];
   x2 = x[I,2];
   x3 = x[I,3];
   x4 = x[I,4];

   Function: (x1 * t + x2 * t * t) / (1 + x3 * t + x4 * t * t);    ! the approximating function

   Error = Function - Goal;
end

sweep
   !  produce the whole picture
   I: 1 2 3
   K: from 1 to 50 step=1  t Goal Function
   {Parametric  X=t  I=all Y=Goal.white.x & Function.white
    Ymin=-0.2 Ymax=1.2 NYticks=7 Y_title="Function"
    Xmin=0 Xmax=1 NXticks=5};

   !  produce an enlarged view
   I: 1 2 3
   K: from 20 to 30 step=1  t Goal Function
   {Parametric  X=t  I=all Y=Goal.white.x & Function.white
    Ymin=0.6 Ymax=0.8 NYticks=2 Y_title="Function"
    Xmin=0.4 Xmax=0.6 NXticks=2};
end

Specification
   K: from 1 to 50 step=1  Function = Goal;
end
```

B CIRCUIT FILE FOR THE HUBER STATISTICAL ESTIMATION PROBLEM

```
!  Example huber2_1.ckt
!  using Huber norm for statistical estimation
!  the FET model parameters are extracted from statistical measurements
!  the starting points and threshold values are recorded below

Control
   no_default_bounds;
   optimizer=huber;
end

Expression
   !  extracted parameters for the FETs

   LG[80] = [0.0249343 0.0382752 0.0244437 0.0305276 0.0320525 0.0351226 0.0384052 0.0404104
            0.0203612 0.0295276 0.0338138 0.0378106 0.0401943 0.0439019 0.0400248 0.19135
            0.0253422 0.0309073 0.0351278 0.0362827 0.0361359 0.040135 0.0377837 0.00740458
            0.0277345 0.0336417 0.0338764 0.0354762 0.0342489 0.0377881 0.0304654 0.0321601
            0.0340442 0.0361249 0.198991 0.0291953 0.0262987 0.026106 0.024076 0.0352524
            0.0397872 0.0311892 0.0342306 0.0215204 0.0439407 0.0443833 0.0302681 0.0349599
            0.0146805 0.0396286 0.0420346 0.0426165 0.0406111 0.0411889 0.0255615 0.0235038
            0.0326257 0.037303 0.0387938 0.0378664 0.0419138 0.23049 0.02803 0.0333612
            0.035629 0.03844 0.0375648 0.0342395 0.0396309 0.0257349 0.0290948 0.0318604
            0.0368525 0.0378124 0.0357473 0.192996 0.189776 0.0312339 0.0367897 0.0195684];

   GDS[80] = [1.75002 1.85644 1.7744 1.75832 1.65872 1.66355 1.7331 1.81709
             1.65813 1.65851 1.69109 1.77409 1.79057 1.72831 1.71837 1.30787
             1.73567 1.73028 1.71128 1.66934 1.75592 1.76912 1.83407 6.07143
             1.72364 1.83527 1.79107 1.76081 1.85241 1.87098 1.82652 1.81948
             1.81915 1.89489 1.43998 1.57586 1.77922 1.7889 1.89212 1.96435
             2.00173 1.93487 1.85441 1.91102 1.90767 1.91848 1.88535 1.94333
             1.83223 1.87512 1.92374 1.88014 1.95414 1.85075 1.96955 1.94416
             1.86048 1.87525 1.92236 2.0376 1.93543 1.45029 1.93602 1.92614
             1.82955 1.88543 1.92705 1.85356 1.8931 1.73942 1.9085 1.85441
             1.85696 1.81298 1.98998 1.37121 0.303805 1.80162 1.94484 1.45825];

   IDSS[80] = [44.046 44.8625 48.2952 39.5738 37.2699 36.206 43.9213 43.5043
              32.3414 37.5004 39.5005 42.4566 42.8292 45.2523 42.9082 43.4862
              42.6225 45.0563 40.6203 40.0449 44.5258 46.4652 47.7613 41.3232
              40.1933 41.7346 42.2244 43.2122 47.3998 48.6415 49.4291 48.6238
              46.3291 54.4703 38.7388 46.9418 41.2691 44.3932 49.2017 54.9949
              58.3515 49.2055 47.4198 49.3502 51.2795 50.9717 48.6745 50.1952
              47.0994 50.0122 51.7302 49.0134 52.9089 55.0526 53.3896 53.2196
              52.7415 51.4503 50.0552 54.7584 51.5082 53.3831 58.6271 56.4666
              51.8816 54.6814 43.5047 47.4531 48.7861 54.7284 51.9124 50.5944
              51.5291 56.008 58.0683 42.1383 19.2931 46.7418 56.2348 58.5505];

   TAU[80] = [2.12742 2.18777 2.17859 2.12656 2.17465 2.1775 2.19197 2.21798
             2.33714 2.15528 2.18736 2.2225 2.22436 2.21208 2.22253 1.45189
             2.17705 2.18329 2.20177 2.22969 2.31806 2.29611 2.27188 1.75367
             2.20324 2.22871 2.26579 2.29229 2.19074 2.29612 2.20923 2.21387
             2.28671 2.35304 1.17145 0.0636748 2.20641 2.29232 2.34982 2.08324
             2.07417 2.11242 2.19065 2.13068 2.0772 2.08495 2.08396 2.1062
             2.25574 2.09906 2.11657 2.07227 2.12352 1.96091 2.19077 2.18401
             2.18293 2.1715 2.22586 2.00958 2.18663 0.132958 2.15652 2.18759
             2.20099 2.21515 2.17997 2.19088 2.19255 0.29736 2.15612 2.13826
             2.19294 2.24055 2.26081 1.75062 0.0063146 2.21265 0.00329595 1.52677];

   C10[80] = [0.362563 0.382896 0.365961 0.364807 0.365247 0.364622 0.378009 0.374774
             0.337379 0.362562 0.364358 0.36599 0.374062 0.380752 0.383404 0.326109
             0.371675 0.374101 0.369312 0.371127 0.377512 0.381166 0.397379 0.230942
             0.368924 0.364903 0.368021 0.375926 0.367993 0.397168 0.381855 0.383397
             0.394855 0.411779 0.324919 0.367993 0.381204 0.391402 0.401683 0.359261
```

```
               0.365136 0.344007 0.366882 0.354401 0.360487 0.353281 0.353136 0.356968
               0.363364 0.361773 0.355389 0.364088 0.362828 0.337045 0.358678 0.360806
               0.360243 0.369753 0.362128 0.373589 0.366805 0.293263 0.373038 0.363664
               0.361583 0.365701 0.355855 0.367136 0.367538 0.332379 0.364487 0.363581
               0.365443 0.377038 0.380442 0.315894 0.263792 0.365288 0.242706 0.406539];

   K1[80] = [1.28545 1.33028 1.20188 1.38141 1.41652 1.41198 1.29941 1.33642
             1.44405 1.37958 1.30465 1.22191 1.31711 1.31551 1.36507 0.924604
             1.31541 1.29288 1.35519 1.34767 1.26382 1.24079 1.36574 1.51348
             1.34128 1.26382 1.25654 1.24532 1.22441 1.32982 1.27337 1.27061
             1.37133 1.37705 0.991581 2.10468 1.39566 1.38151 1.3701 1.06317
             1.06969 1.08956 1.22502 1.10365 1.15051 1.05698 1.17259 1.20432
             1.06294 1.13359 1.08532 1.16405 1.08851 1.01455 1.07219 1.06215
             1.10886 1.20701 1.14688 1.1827 1.13237 0.645973 1.16308 1.10293
             1.15461 1.09642 1.30415 1.22499 1.2621 1.52151 1.22725 1.24006
             1.21956 1.19866 1.1638 0.958443 0.713158 1.28814 0.99874 1.71735];

! starting points and threshold values
! parameter     LG       GDS      IDSS    TAU      C10      K1
! x_mean        0.03     1.8       50     2.25     0.35     1.1
! thres         0.015    0.2       15     0.25     0.03     0.2
! x_variance    2.25e-4  0.04     225     0.0625   9e-4     0.04
! the starting point for x_variance is delta * delta
! the threshold value for optimizing x_variance is also delta * delta.

#define  X   K1          !  specify the parameter to be used

   x_mean = 1.23055;     !  set to be optimizable if estimate the mean
   K: 1;

   x_dev = X[k] - x_mean;

   x_dev2 = x_dev * x_dev;

   x_variance = ?0.0175463?;      ! set to optimizable if estimate the variance
                                  ! should get mean beforehand; idle when estimate mean

   x_std_dev = 100 * sqrt(x_variance) / x_mean;   ! express standard deviation in pctg
end

sweep
!  produce the run chart of these parameters

   K: from 1 to 80 step=1 LG[k] GDS[k] IDSS[k] TAU[k] C10[k] K1[k]
   {Xsweep  Title="Run Chart of Statistical Model Parameter"
    X_title="Index"  Xmin=0 Xmax=80 NXticks=4
    Ymin=0 Ymax=2.5 NYticks=5
    Y=TAU[K].dot  Y_title="TAU (ps)"};
end

spec
!   these two specifications are for estimating the mean and standard deviation, respectively
!   one should be commented out if the other is in use

!  for estimating the mean

!   K: from 1 to 80 step=1 x_mean = X[K];

!  for estimating the standard deviation

   K: from 1 to 80 step=1  x_variance = x_dev2;
end

report
    K1         ??????    $x_mean$        ?????         $x_std_dev$%
end
```

C CIRCUIT FILE FOR RESISTIVE MESH NETWORK

```
!  meshres1.ckt
!  the resistive mesh example for fault locations
!  for all the optimization processes the accuracy is set to 1e-06
!  experiment carried out on apollo

Control
   no_default_bounds;
   two_sided_perturbation;
end

Model
   !  conductances to be found

   g[1:20]= [?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1?
             ?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1? ?1?];

! *************** CASE V ****************
!  starting point:
!  g[1:20]= [?1.5? ?1.5? ?0.5? ?0.5? ?1.5?
!            ?1.5? ?0.5? ?1.5? ?0.5? ?0.5?
!            ?1.5? ?0.5? ?0.5? ?1.5? ?0.5?
!            ?1.5? ?0.5? ?1? ?0.5? ?0.5?];

!  L1 solution with CPU time 18 seconds
!  g[1:20]= [?1? ?0.512245? ?1? ?1? ?0.977388?
!            ?1? ?1? ?1? ?1.02849? ?1?
!            ?1? ?1.03451? ?1? ?1? ?1?
!            ?0.975798? ?1? ?0.478366? ?1? ?0.963325?];

!  Huber solution with CPU time 1 minute 50 seconds
!  the following solution is achieved by commenting out
!  the no_default_bounds in the control block.
!  If the line is not commented out, the Huber optimization
!  will terminate, showing an error msg: LSM: negative or zero parameter
!  values encountered.
!  g[1:20]= [?0.998943? ?0.527167? ?0.975341? ?0.988195? ?0.968394?
!            ?0.999467? ?0.99808? ?0.995907? ?1.03748? ?1.0039?
!            ?0.996223? ?1.01312? ?0.997402? ?0.995045? ?0.9995?
!            ?0.973123? ?0.99384? ?0.527273? ?0.961542? ?0.952621?];

! *************** CASE IV ****************
!  starting point
!  g[1:20]= [?1.5? ?1.1? ?0.5? ?0.9? ?0.75?
!            ?1.3? ?1.26? ?0.7? ?0.6? ?1.5?
!            ?1? ?1.2? ?0.6? ?1.1? ?1.3?
!            ?1? ?1.1? ?1? ?0.8? ?0.8?];

!  L1 "tentative" solution with CPU time 1 minute 32 seconds
!  g[1:20]= [?1.08702? ?0.656285? ?0.672593? ?1? ?1?
!            ?1.0009? ?1? ?1? ?1.03499? ?1.1609?
!            ?1? ?0.950037? ?0.899102? ?1? ?1?
!            ?0.908244? ?1? ?0.790898? ?0.802527? ?0.878271?];

!  After restarting, several times, the L1 optimization from
!  the above solution, the following solution was obtained.  And the
!  optimization can not go even further.  It seems that the previous solution
!  is a local L1 minimum, but easy to get out of it, while the following
!  one is a "deep" one which is not easy to get out of.
!  g[1:20]= [?1.05979? ?0.603228? ?0.782906? ?1? ?1?
!            ?1? ?1? ?1? ?1.0428? ?1.05059?
!            ?1? ?0.988123? ?1? ?1? ?1?
!            ?0.921347? ?1? ?0.680585? ?0.874652? ?0.898886?];
!  Huber solution with CPU time 1 minute 22 seconds
!  g[1:20]= [?0.998922? ?0.527168? ?0.97539? ?0.988231? ?0.968432?
!            ?0.999456? ?0.998075? ?0.995891? ?1.03754? ?1.00396?
!            ?0.996294? ?1.01323? ?0.997395? ?0.995083? ?0.999489?
!            ?0.973309? ?0.993925? ?0.526757? ?0.961853? ?0.952817?];

! **************** CASE III ****************
!  starting from nominal values
!  g[1:20]= [?1? ?1? ?1? ?1? ?1?
!            ?1? ?1? ?1? ?1? ?1?
!            ?1? ?1? ?1? ?1? ?1?
!            ?1? ?1? ?1? ?1? ?1?];

!  L1 solution with CPU time 32 seconds for optimization
!  g[1:20]= [?1? ?0.511079? ?1? ?1? ?0.972984?
!            ?1? ?1? ?1? ?1.02408? ?1?
!            ?1? ?1.02726? ?1? ?1? ?1?
!            ?0.966401? ?1? ?0.499147? ?0.985868? ?0.956008?];

!  Huber solution with CPU time 55 seconds
!  g[1:20]= [?0.998928? ?0.52717? ?0.975381? ?0.988241? ?0.968438?
!            ?0.999438? ?0.998062? ?0.995893? ?1.03754? ?1.00397?
!            ?0.996285? ?1.01324? ?0.997392? ?0.995049? ?0.999483?
!            ?0.973316? ?0.993942? ?0.52674? ?0.961859? ?0.95283?];

! *************** CASE II *****************
!  starting point:
!  g[1:20]= [?1.1? ?0.91? ?1.3? ?1.02? ?0.8?
!            ?0.92? ?1.4? ?0.6? ?1.2? ?1?
!            ?0.8? ?1? ?0.75? ?1? ?1?
!            ?1? ?0.85? ?1? ?0.91? ?1?];

!  L1 solution with CPU time 49 seconds for optimization
!  g[1:20]= [?1? ?0.512247? ?1? ?1? ?0.977387?
!            ?1? ?1? ?1? ?1.0285? ?1?
!            ?1? ?1.03451? ?1? ?1? ?1?
!            ?0.975801? ?1? ?0.478361? ?1? ?0.963327?];

!  Huber solution with CPU time 1 minute 8 second
!  g[1:20]= [?0.998925? ?0.527154? ?0.975395? ?0.988207? ?0.968415?
!            ?0.999459? ?0.99807? ?0.995913? ?1.0375? ?1.00395?
!            ?0.996267? ?1.01319? ?0.997376? ?0.995054? ?0.99949?
!            ?0.973237? ?0.993884? ?0.526955? ?0.961733? ?0.952743?];

! *************** CASE I *******************
!  starting point
!  g[1:20]= [?1? ?1? ?1? ?1.2? ?1?
!            ?1? ?1? ?1? ?1? ?1?
!            ?1? ?1? ?1? ?0.95?
!            ?1? ?1? ?1? ?1? ?1? ?1?];

!  L1 solution with CPU time 39 seconds for optimization
!  g[1:20]= [?1.0238? ?0.524592? ?0.940039? ?1? ?0.976645?
!             ?1? ?1? ?1? ?1.00306? ?1?
!             ?1? ?1? ?1? ?1? ?1?
!             ?0.904762? ?1? ?0.63465? ?0.907852? ?0.903645?];

!  Huber solution with CPU time 48 seconds
!  g[1:20]= [?0.998925? ?0.527171? ?0.975376? ?0.988233? ?0.968421?
!            ?0.999459? ?0.998095? ?0.995911? ?1.03753? ?1.00395?
!            ?0.996291? ?1.01321? ?0.997389? ?0.995045? ?0.999493?
!            ?0.973285? ?0.993912? ?0.526833? ?0.961805? ?0.95279?];

!  define the circuit

   res  1  2 r = (1/g[1]);
   res  1  3 r = (1/g[2]);
   res  1  4 r = (1/g[3]);
   res  2  5 r = (1/g[4]);
   res  2  6 r = (1/g[5]);
   res  3  4 r = (1/g[6]);
   res  4  5 r = (1/g[7]);
   res  5  6 r = (1/g[8]);
   res  3  7 r = (1/g[9]);
   res  4  8 r = (1/g[10]);
   res  5  9 r = (1/g[11]);
   res  6 10 r = (1/g[12]);
   res  7  8 r = (1/g[13]);
   res  8  9 r = (1/g[14]);
   res  9 10 r = (1/g[15]);
   res  7 11 r = (1/g[16]);
   res  8 11 r = (1/g[17]);
   res  9  0 r = (1/g[18]);
   res 10  0 r = (1/g[19]);
   res 11  0 r = (1/g[20]);

! current source is applied to the circuit

   Isource 1 0  IDC = 1A;

!  voltages of accessible nodes
   VLABEL  1 0 NAME=V1;
   VLABEL  2 0 NAME=V2;
   VLABEL  3 0 NAME=V3;
   VLABEL  6 0 NAME=V6;
   VLABEL  7 0 NAME=V7;
   VLABEL 10 0 NAME=V10;
   VLABEL 11 0 NAME=V11;

   CIRCUIT;

   V[7] = [V1 V2  V3  V6  V7  V10  V11];

! measured data obtain by simulating the circuit with actual parameter values

   V0[7] = [1.25297 0.874709 0.832924 0.652006 0.610618 0.381562 0.401964];
   Vdif[7] = V - V0;

   dg[20] = (g - 1) * 100;        !  calculate the percentage deviation parameters
end

Spec
   DC: g=1  Vdif=0 w=1000;        !  w is the penalty multiplier
end

Report
   $dg$          !  generate the list of percentage deviation
end
```

D CIRCUIT FILE FOR THE MULTIPLEXER PROBLEM

D.1 Starting Point

```
!  Example  mux05_1.ckt
!  5-channel multiplexer optimization using Datapipes

#define YES            1
#define NO             0

#define ITYPE          3  ! uniform channel data, couplings included in input
#define N_CHANNELS     5  ! number of channels
#define FILTER_ORDER   6  ! order of channel filters
#define N_COUPLINGS   12  ! number of couplings in each multi-cavity filter

#define CENTER_FREQ1     12.18GHZ  ! center frequency of the first channel
#define CHANNEL_SPACING  40MHZ     ! spacing between channel center frequencies
#define BAND_WIDTH       39MHZ     ! channel usable band width
#define Q_FACTOR         12000     ! unloaded Q factor of the channel filters
#define DIAMETER         1.07      ! diameters of channel filters in inches
#define WIDTH            0.75      ! waveguide width in inches
#define G_TERMINATION    1         ! terminating conductance of each channel

#define NONIDEAL_JUNCTION  YES
#define LOSSY_FILTER       YES
#define DISPERSIVE_FILTER  YES

#define DIAGONAL       0   ! coupling types
#define SCREW          1
#define IRIS          -1

#define N_CHANNEL_DATA  (3 + N_COUPLINGS)  ! spacing & transformer ratios

Model
   !  define coupling positions and types

   Coupling_Position[1:(3 * N_COUPLINGS)] =
      [ 1, 1, DIAGONAL,  2, 2, DIAGONAL,  3, 3, DIAGONAL,
        4, 4, DIAGONAL,  5, 5, DIAGONAL,  6, 6, DIAGONAL,
        1, 2, SCREW,     2, 3, IRIS,      3, 4, SCREW,
        3, 6, IRIS,      4, 5, IRIS,      5, 6, SCREW ];

   ! channel data includes manifold spacing & transformer ratios
   Channel1[1:N_CHANNEL_DATA] =
      [ ?0.7?     ?1?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel2[1:N_CHANNEL_DATA] =
      [ ?1.2?      ?1?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel3[1:N_CHANNEL_DATA] =
      [ ?0.65?      ?1?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];
   Channel4[1:N_CHANNEL_DATA] =
      [ ?0.63?      ?1?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel5[1:N_CHANNEL_DATA] =
      [ ?0.8?      ?1?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Datapipe: SIM  FILE="simux"           !  the path for "simux" must be correct
      N_INPUT=(15 + 3 * N_COUPLINGS + N_CHANNELS * N_CHANNEL_DATA)
      INPUT = (ITYPE, N_CHANNELS, FILTER_ORDER, FREQ, CENTER_FREQ1,
               CHANNEL_SPACING, Q_FACTOR, DIAMETER, BAND_WIDTH, G_TERMINATION,
               WIDTH, NONIDEAL_JUNCTION, LOSSY_FILTER, DISPERSIVE_FILTER,
               N_COUPLINGS,  Coupling_Position,
               Channel1, Channel2,  Channel3,  Channel4,  Channel5)
      N_OUTPUT=(N_CHANNELS + 1)
      OUTPUT = (RET_LOSS, INS_LOSS[1:N_CHANNELS]);

   ! make the responses (losses) upside-down

   RETURN_LOSS: -RET_LOSS;
   INSERTION_LOSS[1:N_CHANNELS]: -INS_LOSS;
end

Sweep
   FREQ: from (CENTER_FREQ1 - N_CHANNELS * CHANNEL_SPACING)
         to (CENTER_FREQ1 + CHANNEL_SPACING)  step=1MHZ
   RETURN_LOSS  INSERTION_LOSS
   {Xsweep  Title="Multiplexer Responses"
    Y_Title="Common Port Return Loss and Channel Insertion Loss"
    Y=return_loss.white & insertion_loss[1].red & insertion_loss[2].red &
      insertion_loss[3].red & insertion_loss[4].red & insertion_loss[5].red
    Ymin=-40  NYTicks=4  Comments="Starting Point"};
end

Spec
   FREQ: from (CENTER_FREQ1 - (N_CHANNELS - 1/2) * CHANNEL_SPACING + 3MHz)
         to (CENTER_FREQ1 + CHANNEL_SPACING / 2 - 3MHz)  step=2MHZ
   RETURN_LOSS < -20;
end
```

D.2 Partial Minimax Solution

```
!  Example  mux05_2.ckt
!  5-channel multiplexer optimization using Datapipes
!  partial minimax solution
#define YES             1
#define NO              0

#define ITYPE           3  ! uniform channel data, couplings included in input
#define N_CHANNELS      5  ! number of channels
#define FILTER_ORDER    6  ! order of channel filters
#define N_COUPLINGS    12  ! number of couplings in each multi-cavity filter

#define CENTER_FREQ1      12.18GHZ  ! center frequency of the first channel
#define CHANNEL_SPACING   40MHZ     ! spacing between channel center frequencies
#define BAND_WIDTH        39MHZ     ! channel usable band width
#define Q_FACTOR          12000     ! unloaded Q factor of the channel filters
```

```
#define DIAMETER         1.07      ! diameters of channel filters in inches
#define WIDTH            0.75      ! waveguide width in inches
#define G_TERMINATION    1         ! terminating conductance of each channel

#define NONIDEAL_JUNCTION  YES
#define LOSSY_FILTER       YES
#define DISPERSIVE_FILTER  YES

#define DIAGONAL       0   ! coupling types
#define SCREW          1
#define IRIS          -1

#define N_CHANNEL_DATA  (3 + N_COUPLINGS)  ! spacing & transformer ratios
Control
   no_default_bounds;
   two_sided_perturbation;
   Optimizer=Minimax;
end

Model
   ! define coupling positions and types
   Coupling_Position[1:(3 * N_COUPLINGS)] =
      [ 1, 1, DIAGONAL,  2, 2, DIAGONAL,  3, 3, DIAGONAL,
        4, 4, DIAGONAL,  5, 5, DIAGONAL,  6, 6, DIAGONAL,
        1, 2, SCREW,     2, 3, IRIS,      3, 4, SCREW,
        3, 6, IRIS,      4, 5, IRIS,      5, 6, SCREW ];

   ! channel data includes manifold spacing & transformer ratios
   Channel1[1:N_CHANNEL_DATA] =
      [ ?0.625982?     ?1.00851?      1
        0    0    0
        0    0    0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel2[1:N_CHANNEL_DATA] =
      [ ?1.2038?      ?0.970376?      1
        0    0    0
        0    0    0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel3[1:N_CHANNEL_DATA] =
      [ ?0.567152?      ?0.943777?      1
        0    0    0
        0    0    0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel4[1:N_CHANNEL_DATA] =
      [ ?0.508982?      ?0.855796?      1
        0    0    0
        0    0    0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel5[1:N_CHANNEL_DATA] =
      [ ?1.04428?      ?0.746218?      1
        0    0    0
        0    0    0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Datapipe: SIM  FILE="simux"
      N_INPUT=(15 + 3 * N_COUPLINGS + N_CHANNELS * N_CHANNEL_DATA)
      INPUT = (ITYPE, N_CHANNELS, FILTER_ORDER, FREQ, CENTER_FREQ1,
               CHANNEL_SPACING, Q_FACTOR, DIAMETER, BAND_WIDTH, G_TERMINATION,
              WIDTH, NONIDEAL_JUNCTION, LOSSY_FILTER, DISPERSIVE_FILTER,
              N_COUPLINGS,  Coupling_Position,
              Channel1, Channel2,  Channel3,  Channel4,  Channel5)
      N_OUTPUT=(N_CHANNELS + 1)  OUTPUT = (RET_LOSS, INS_LOSS[1:N_CHANNELS]);

   ! make the responses (losses) upside-down
   RETURN_LOSS: -RET_LOSS;
   INSERTION_LOSS[1:N_CHANNELS]: -INS_LOSS;
end

Sweep
   FREQ: from (CENTER_FREQ1 - N_CHANNELS * CHANNEL_SPACING)
         to (CENTER_FREQ1 + CHANNEL_SPACING)  step=1MHZ
   RETURN_LOSS  INSERTION_LOSS
   {Xsweep  Title="Multiplexer Responses"
    Y_Title="Common Port Return Loss and Channel Insertion Loss"
    Y=return_loss.white & insertion_loss[1].red & insertion_loss[2].red &
      insertion_loss[3].red & insertion_loss[4].red & insertion_loss[5].red
    Ymin=-40  NYTicks=4  Comments="Minimax Optimization" & "of spacings and input" &
    "transformer ratios"};
end

Spec
   FREQ: from (CENTER_FREQ1 - (N_CHANNELS - 1/2) * CHANNEL_SPACING + 3MHz)
         to (CENTER_FREQ1 + CHANNEL_SPACING / 2 - 3MHz)  step=2MHZ
   RETURN_LOSS < -20;
end
```

D.3 Partial One-Sided Huber Solution

```
!  Example  mux05_3.ckt
!  5-channel multiplexer optimization using Datapipes
!  Huber threshold=1  28 mins.

#define YES            1
#define NO             0

#define ITYPE          3  ! uniform channel data, couplings included in input
#define N_CHANNELS     5  ! number of channels
#define FILTER_ORDER   6  ! order of channel filters
#define N_COUPLINGS   12  ! number of couplings in each multi-cavity filter

#define CENTER_FREQ1     12.18GHZ  ! center frequency of the first channel
#define CHANNEL_SPACING  40MHZ     ! spacing between channel center frequencies
#define BAND_WIDTH       39MHZ     ! channel usable band width
#define Q_FACTOR         12000     ! unloaded Q factor of the channel filters
#define DIAMETER         1.07      ! diameters of channel filters in inches
#define WIDTH            0.75      ! waveguide width in inches
#define G_TERMINATION    1         ! terminating conductance of each channel

#define NONIDEAL_JUNCTION  YES
#define LOSSY_FILTER       YES
#define DISPERSIVE_FILTER  YES

#define DIAGONAL       0   ! coupling types
#define SCREW          1
#define IRIS          -1

#define N_CHANNEL_DATA  (3 + N_COUPLINGS)  ! spacing & transformer ratios
Control
   no_default_bounds;
   two_sided_perturbation;
   optimizer=Huber;
end
```

```
Model
   ! define coupling positions and types
   Coupling_Position[1:(3 * N_COUPLINGS)] =
      [ 1, 1, DIAGONAL,  2, 2, DIAGONAL,  3, 3, DIAGONAL,
        4, 4, DIAGONAL,  5, 5, DIAGONAL,  6, 6, DIAGONAL,
        1, 2, SCREW,     2, 3, IRIS,      3, 4, SCREW,
        3, 6, IRIS,      4, 5, IRIS,      5, 6, SCREW ];

   ! channel data includes manifold spacing & transformer ratios
   Channel1[1:N_CHANNEL_DATA] =
      [ ?0.674786?     ?1.08476?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel2[1:N_CHANNEL_DATA] =
      [ ?1.21093?      ?0.98799?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel3[1:N_CHANNEL_DATA] =
      [ ?0.602474?      ?0.914973?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel4[1:N_CHANNEL_DATA] =
      [ ?0.615386?      ?0.920306?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Channel5[1:N_CHANNEL_DATA] =
      [ ?0.594315?      ?0.846835?      1
        0   0   0
        0   0   0
        0.7  0.5  0.4
        -0.4  0.8  0.8 ];

   Datapipe: SIM  FILE="simux"
      N_INPUT=(15 + 3 * N_COUPLINGS + N_CHANNELS * N_CHANNEL_DATA)
      INPUT = (ITYPE, N_CHANNELS, FILTER_ORDER, FREQ, CENTER_FREQ1,
               CHANNEL_SPACING, Q_FACTOR, DIAMETER, BAND_WIDTH, G_TERMINATION,
               WIDTH, NONIDEAL_JUNCTION, LOSSY_FILTER, DISPERSIVE_FILTER,
               N_COUPLINGS,  Coupling_Position,
               Channel1, Channel2,  Channel3,  Channel4,  Channel5)
      N_OUTPUT=(N_CHANNELS + 1)
      OUTPUT = (RET_LOSS, INS_LOSS[1:N_CHANNELS]);

   ! make the responses (losses) upside-down
   RETURN_LOSS: -RET_LOSS;
   INSERTION_LOSS[1:N_CHANNELS]: -INS_LOSS;
end

Sweep
   FREQ: from (CENTER_FREQ1 - N_CHANNELS * CHANNEL_SPACING)
         to (CENTER_FREQ1 + CHANNEL_SPACING)  step=1MHZ
   RETURN_LOSS  INSERTION_LOSS
   {Xsweep  Title="Multiplexer Responses"
    Y_Title="Common Port Return Loss and Channel Insertion Loss"
    Y=return_loss.white & insertion_loss[1].red & insertion_loss[2].red &
      insertion_loss[3].red & insertion_loss[4].red & insertion_loss[5].red
    Ymin=-40  NYTicks=4  Comments="Huber+ Optimization" & "of spacings and input" &
    "transformer ratios"};
end

Spec
   FREQ: from (CENTER_FREQ1 - (N_CHANNELS - 1/2) * CHANNEL_SPACING + 3MHz)
         to (CENTER_FREQ1 + CHANNEL_SPACING / 2 - 3MHz)  step=2MHZ
   RETURN_LOSS < -20;
end
```

D.4 Partial One-Sided Huber Solution with 45 Variables

```
!  Example  mux05_4.ckt
!  5-channel multiplexer optimization using Datapipes
!  Huber threshold=1  optimization following mux05_3.ckt with more variables

#define YES            1
#define NO             0

#define ITYPE          3  ! uniform channel data, couplings included in input
#define N_CHANNELS     5  ! number of channels
#define FILTER_ORDER   6  ! order of channel filters
#define N_COUPLINGS   12  ! number of couplings in each multi-cavity filter

#define CENTER_FREQ1     12.18GHZ  ! center frequency of the first channel
#define CHANNEL_SPACING  40MHZ     ! spacing between channel center frequencies
#define BAND_WIDTH       39MHZ     ! channel usable band width
#define Q_FACTOR         12000     ! unloaded Q factor of the channel filters
#define DIAMETER         1.07      ! diameters of channel filters in inches
#define WIDTH            0.75      ! waveguide width in inches
#define G_TERMINATION    1         ! terminating conductance of each channel

#define NONIDEAL_JUNCTION  YES
#define LOSSY_FILTER       YES
#define DISPERSIVE_FILTER  YES

#define DIAGONAL       0   ! coupling types
#define SCREW          1
#define IRIS          -1

#define N_CHANNEL_DATA  (3 + N_COUPLINGS)  ! spacing & transformer ratios
Control
   no_default_bounds;
   two_sided_perturbation;
   optimizer=Huber;
end

Model
   ! define coupling positions and types
   Coupling_Position[1:(3 * N_COUPLINGS)] =
      [ 1, 1, DIAGONAL,  2, 2, DIAGONAL,  3, 3, DIAGONAL,
        4, 4, DIAGONAL,  5, 5, DIAGONAL,  6, 6, DIAGONAL,
        1, 2, SCREW,     2, 3, IRIS,      3, 4, SCREW,
        3, 6, IRIS,      4, 5, IRIS,      5, 6, SCREW ];

   ! channel data includes manifold spacing & transformer ratios
   Channel1[1:N_CHANNEL_DATA] =
      [ ?0.626466?     ?1.29311?      ?1.09512?
        0   0   0
        0   0   0
        ?0.8668?  ?0.575554?  ?0.444766?
        ?-0.406919?  ?0.864905?  ?0.84915? ];

   Channel2[1:N_CHANNEL_DATA] =
      [ ?1.14717?      ?0.765969?      ?1.1201?
        0   0   0
        0   0   0
        ?0.560553?  ?0.516304?  ?0.407215?
        ?-0.420774?  ?0.824661?  ?0.791898? ];

   Channel3[1:N_CHANNEL_DATA] =
      [ ?0.675946?      ?0.820185?      ?1.08424?
        0   0   0
        0   0   0
        ?0.609648?  ?0.544548?  ?0.436598?
        ?-0.400579?  ?0.839031?  ?0.804051? ];

   Channel4[1:N_CHANNEL_DATA] =
      [ ?0.654115?      ?0.803595?      ?1.12017?
        0   0   0
        0   0   0
        ?0.597031?  ?0.520451?  ?0.40886?
        ?-0.427871?  ?0.847752?  ?0.829882? ];

   Channel5[1:N_CHANNEL_DATA] =
      [ ?0.635451?      ?0.787674?      ?1.14712?
        0   0   0
        0   0   0
        ?0.640097?  ?0.557019?  ?0.419756?
        ?-0.458404?  ?0.927792?  ?0.917915? ];

   Datapipe: SIM  FILE="simux"
      N_INPUT=(15 + 3 * N_COUPLINGS + N_CHANNELS * N_CHANNEL_DATA)
      INPUT = (ITYPE, N_CHANNELS, FILTER_ORDER, FREQ, CENTER_FREQ1,
               CHANNEL_SPACING, Q_FACTOR, DIAMETER, BAND_WIDTH, G_TERMINATION,
               WIDTH, NONIDEAL_JUNCTION, LOSSY_FILTER, DISPERSIVE_FILTER,
               N_COUPLINGS,  Coupling_Position,
               Channel1, Channel2,  Channel3,  Channel4,  Channel5)
      N_OUTPUT=(N_CHANNELS + 1)
      OUTPUT = (RET_LOSS, INS_LOSS[1:N_CHANNELS]);

   ! make the responses (losses) upside-down
   RETURN_LOSS: -RET_LOSS;
   INSERTION_LOSS[1:N_CHANNELS]: -INS_LOSS;

   K: 1;

   Center_freq = center_freq1 - (K - 1) * CHANNEL_SPACING;
   lower_freq = center_freq - 18MHZ;
   upper_freq = center_freq + 18MHZ;
end

Sweep
   FREQ: from (CENTER_FREQ1 - N_CHANNELS * CHANNEL_SPACING)
         to (CENTER_FREQ1 + CHANNEL_SPACING)  step=1MHZ
   RETURN_LOSS  INSERTION_LOSS
   {Xsweep  Title="Multiplexer Responses"
    Y_Title="Common Port Return Loss and Channel Insertion Loss"
    Y=return_loss.white & insertion_loss[1].red & insertion_loss[2].red &
      insertion_loss[3].red & insertion_loss[4].red & insertion_loss[5].red
    Ymin=-40  NYTicks=4  Comments="Huber+ Optimization" &
    "with more variables"};
end

Spec
   FREQ: from (CENTER_FREQ1 - (N_CHANNELS - 1/2) * CHANNEL_SPACING + 3MHz)
         to (CENTER_FREQ1 + CHANNEL_SPACING / 2 - 3MHz)  step=2MHZ
   RETURN_LOSS < -20;

   K: from 1 to N_CHANNELS step=1
   FREQ=lower_freq  INSERTION_LOSS[K] > -2;
   K: from 1 to N_CHANNELS step=1
   FREQ=upper_freq  INSERTION_LOSS[K] > -2;
end
```

D.5 Minimax Solution with Full Set of Variables

```
!  Example  mux05_5.ckt
!  5-channel multiplexer optimization using Datapipes
!  Minimax optimization from the Huber solution with all variables

#define YES            1
#define NO             0

#define ITYPE          3  ! uniform channel data, couplings included in input
#define N_CHANNELS     5  ! number of channels
#define FILTER_ORDER   6  ! order of channel filters
#define N_COUPLINGS   12  ! number of couplings in each multi-cavity filter

#define CENTER_FREQ1     12.18GHZ  ! center frequency of the first channel
#define CHANNEL_SPACING  40MHZ     ! spacing between channel center frequencies
#define BAND_WIDTH       39MHZ     ! channel usable band width
#define Q_FACTOR         12000     ! unloaded Q factor of the channel filters
#define DIAMETER         1.07      ! diameters of channel filters in inches
#define WIDTH            0.75      ! waveguide width in inches
#define G_TERMINATION    1         ! terminating conductance of each channel

#define NONIDEAL_JUNCTION  YES
#define LOSSY_FILTER       YES
#define DISPERSIVE_FILTER  YES

#define DIAGONAL       0   ! coupling types
#define SCREW          1
#define IRIS          -1

#define N_CHANNEL_DATA  (3 + N_COUPLINGS)  ! spacing & transformer ratios
Control
   no_default_bounds;
   two_sided_perturbation;
   optimizer=minimax;
end

Model
   ! define coupling positions and types
   Coupling_Position[1:(3 * N_COUPLINGS)] =
      [ 1, 1, DIAGONAL,  2, 2, DIAGONAL,  3, 3, DIAGONAL,
        4, 4, DIAGONAL,  5, 5, DIAGONAL,  6, 6, DIAGONAL,
        1, 2, SCREW,     2, 3, IRIS,      3, 4, SCREW,
        3, 6, IRIS,      4, 5, IRIS,      5, 6, SCREW ];

   ! channel data includes manifold spacing & transformer ratios
   Channel1[1:N_CHANNEL_DATA] =
      [ ?0.683728?     ?1.42336?      ?1.15506?
        ?-1 -0.0199673 1?   ?-1 -0.140397 1?   ?-1 -0.0166459 1?
        ?-1 0.0156095 1?   ?-1 0.0583418 1?   ?-1 0.00684918 1?
        ?0.897985?  ?0.576187?  ?0.416189?
        ?-0.452261?  ?0.851923?  ?0.851719? ];

   Channel2[1:N_CHANNEL_DATA] =
      [ ?1.17059?      ?0.860318?      ?1.14977?
        ?-1 0.0277699 1?   ?-1 -0.00124854 1?   ?-1 0.00166732 1?
        ?-1 -0.0077476 1?   ?-1 0.0249734 1?   ?-1 0.0169127 1?
        ?0.603631?  ?0.549561?  ?0.349295?
        ?-0.582818?  ?0.964327?  ?0.82081? ];
   Channel3[1:N_CHANNEL_DATA] =
      [ ?0.644967?      ?0.896397?      ?1.15165?
        ?-1 0.0694339 1?   ?-1 -0.000463352 1?   ?-1 0.00136821 1?
        ?-1 0.00773093 1?   ?-1 -0.0037287 1?   ?-1 -0.0078298 1?
        ?0.611359?  ?0.549923?  ?0.344509?
        ?-0.590034?  ?0.964327?  ?0.814993? ];

   Channel4[1:N_CHANNEL_DATA] =
      [ ?0.627786?      ?0.894384?      ?1.16053?
        ?-1 0.0881768 1?   ?-1 0.0028601 1?   ?-1 0.000219793 1?
        ?-1 0.00485265 1?   ?-1 -0.000234877 1?   ?-1 -0.000796921 1?
        ?0.617605?  ?0.548846?  ?0.367279?
        ?-0.55435?  ?0.946189?  ?0.851538? ];

   Channel5[1:N_CHANNEL_DATA] =
      [ ?0.611222?      ?0.901974?      ?1.22487?
        ?-1 0.228978 1?   ?-1 0.0614491 1?   ?-1 -0.00136067 1?
        ?-1 -0.0255624 1?   ?-1 -0.0431502 1?   ?-1 -0.000375926 1?
        ?0.683225?  ?0.557142?  ?0.381107?
        ?-0.528463?  ?0.89546?  ?0.896695? ];

   Datapipe: SIM  FILE="simux"
      N_INPUT=(15 + 3 * N_COUPLINGS + N_CHANNELS * N_CHANNEL_DATA)
      INPUT = (ITYPE, N_CHANNELS, FILTER_ORDER, FREQ, CENTER_FREQ1,
               CHANNEL_SPACING, Q_FACTOR, DIAMETER, BAND_WIDTH, G_TERMINATION,
               WIDTH, NONIDEAL_JUNCTION, LOSSY_FILTER, DISPERSIVE_FILTER,
               N_COUPLINGS,  Coupling_Position,
               Channel1, Channel2,  Channel3,  Channel4,  Channel5)
      N_OUTPUT=(N_CHANNELS + 1)
      OUTPUT = (RET_LOSS, INS_LOSS[1:N_CHANNELS]);

   ! make the responses (losses) upside-down
   RETURN_LOSS: -RET_LOSS;
   INSERTION_LOSS[1:N_CHANNELS]: -INS_LOSS;

   K: 1;

   Center_freq = center_freq1 - (K - 1) * CHANNEL_SPACING;
   lower_freq = center_freq - 18MHZ;
   upper_freq = center_freq + 18MHZ;
end

Sweep
   FREQ: from (CENTER_FREQ1 - N_CHANNELS * CHANNEL_SPACING)
         to (CENTER_FREQ1 + CHANNEL_SPACING)  step=1MHZ
   RETURN_LOSS  INSERTION_LOSS
   {Xsweep  Title="Multiplexer Responses"
    Y_Title="Common Port Return Loss and Channel Insertion Loss"
    Y=return_loss.white & insertion_loss[1].red & insertion_loss[2].red &
      insertion_loss[3].red & insertion_loss[4].red & insertion_loss[5].red
    Ymin=-40  NYTicks=4  Comments="Minimax Optimization" &
      "from Huber solution" &
      "with all variables"};
end

Spec
   FREQ: from (CENTER_FREQ1 - (N_CHANNELS - 1/2) * CHANNEL_SPACING + 3MHz)
         to (CENTER_FREQ1 + CHANNEL_SPACING / 2 - 3MHz)  step=2MHZ
   RETURN_LOSS < -30;

   K: from 1 to N_CHANNELS step=1
   FREQ=lower_freq  INSERTION_LOSS[K] > -2 w=10;

   K: from 1 to N_CHANNELS step=1
   FREQ=upper_freq  INSERTION_LOSS[K] > -2 w=10;
end
```

E CIRCUIT FILE FOR COMPARING DIFFERENT OPTIMIZERS

```
! Example add3_1.ckt
! compare dedicated algorithm and generic optimizers
! using Huber norm to estimate the mean of tau

Control
   two_sided_perturbation;
   no_default_bounds;
end

Expression
   ! the measurement data

   TAU[80] = [2.12742 2.18777 2.17859 2.12656 2.17465 2.1775 2.19197 2.21798
              2.33714 2.15528 2.18736 2.2225 2.22436 2.21208 2.22253 1.45189
              2.17705 2.18329 2.20177 2.22969 2.31806 2.29611 2.27188 1.75367
              2.20324 2.22871 2.26579 2.29229 2.19074 2.29612 2.20923 2.21387
              2.28671 2.35304 1.17145 0.0636748 2.20641 2.29232 2.34982 2.08324
              2.07417 2.11242 2.19065 2.13068 2.0772 2.08495 2.08396 2.1062
              2.25574 2.09906 2.11657 2.07227 2.12352 1.96091 2.19077 2.18401
              2.18293 2.1715 2.22586 2.00958 2.18663 0.132958 2.15652 2.18759
              2.20099 2.21515 2.17997 2.19088 2.19255 0.29736 2.15612 2.13826
              2.19294 2.24055 2.26081 1.75062 0.0063146 2.21265 0.00329595 1.52677];

! threshold values: 0.25

#define  X   TAU

! starting point 2.25
! method  solution  # of function evaluations
! Huber   2.15369   4
! Q-N     2.15368   5
! C-G     2.15368   11
! Simplex 2.15438   16

! starting point 2.0
! method  solution  # of function evaluations
! Huber   2.15367   4
! Q-N     2.15368   5
! C-G     2.15368   13
! Simplex 2.1525    16

! starting point 3.0
! method  solution  # of function evaluations
! Huber   2.15372   4
! Q-N     2.15368   7
! C-G     2.15368   14
! Simplex 2.15344   24

! starting point 1.5
! method  solution  # of function evaluations
! Huber   2.15361   4
! Q-N     2.15368   8
! C-G     2.15368   13
! Simplex 2.15344   26

   x_mean = ?2.15344?;
   K: 1;

   x_dev = X[k] - x_mean;

! construct explicitly the Huber objective function for generic optimizers
```

```
   error = x_mean - X[K];
   error_abs = abs(error);

   Hobj = if (error_abs < 0.25) (error_abs * error_abs / 2)
          else (if (error > 0.25) (0.25 * (error - 0.125))
                else (-0.25 * (error + 0.125)));
end

sweep
   K: from 1 to 80 step=1 TAU[k]
   {Xsweep  Title="Run Chart of Statistical Model Parameter"
    X_title="Index"  Xmin=0 Xmax=80 NXticks=4
    Ymin=0 Ymax=2.5 NYticks=5
    Y=TAU[K].dot  Y_title="TAU (ps)"};
end

spec
!  specification for the dedicated Huber algorithm

!   K: from 1 to 80 step=1 x_mean = X[K];

!  specification for generic optimizer

   K: from 1 to 80 step=1 Hobj;

end
```

F CIRCUIT FILE FOR DISPLAYING THE PERCENTAGE OF "SMALL ERRORS"

F.1 Parameter C_{10}

F.1.1 The Circuit File

```
!  file C10.ckt

expression

  i = 1;                 ! index
  k = i / 500;           ! rescale for correct display interval

  !  number of small errors for different threshold
  !  numbers obtained from the file "number_c.dat" which is
  !  an output file of the C program "number_c.c"

  c10_number[1:100] = [
        17 30 37 43 46 51 55 59 62 62
        62 63 64 64 67 69 70 70 71 72
        74 74 74 75 76 76 76 76 76 76
        76 76 76 76 76 77 77 77 77 77
        77 77 77 77 77 77 77 77 77 78
        78 78 78 78 78 78 78 78 78 79
        79 79 79 79 79 80 80 80 80 80
        80 80 80 80 80 80 80 80 80 80
        80 80 80 80 80 80 80 80 80 80
        80 80 80 80 80 80 80 80 80 80
        ];

  !  calculate the percentage
    c10_percent[100] = 100 * c10_number / 80;
end

sweep
  i: from 1 to 100 step 1 k c10_percent[i]
  {parametric Title = "percentage of assumed small errors for C10"
            X = k  Y = C10_percent[i]
            Ymin = 0  Ymax = 100  NYticks = 4
            Xmin = 0  Xmax = 0.2 NXticks = 4
            X_title = "k"  Y_title = "percentage"};
end
```

F.1.2 C Program to Calculate the Number of Small Errors

```
/* File: number_c.c */

#include "stdio.h"
#include "string.h"
#include "math.h"

main ()

{

  int i, number;
  float j, mean,up_boundary, lw_boundary, c10[80];
  char k_value[20];
  FILE *file_data = NULL;
  FILE *file_mean = NULL;
  FILE *file_out = NULL;

  file_data = fopen("c10_data.dat", "r");         /* contains the extracted parameters */
  file_mean = fopen("c10_detail.dat", "r");       /* contains the estimated means and */
                                                  /* standard deviations from OSA90/hope */
  file_out = fopen("number_c10.dat", "w");        /* contains the number of small errors */

  for (i = 1; i < 81; i++)
    fscanf(file_data, "%f", &c10[i-1]);           /* reads in the extracted parameters */

  fprintf(file_out, "C10\n");                     /* a reminder */

  for (j = 0.002; j < 0.2; j = j + 0.002) {

    number = 0;

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);             /* to read correct text, skip 2 col's */

    fprintf(file_out, "%s  ", k_value);           /* output the current threshold value */

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);             /* to read correct text, skip 2 col's */

    fscanf(file_mean, "%f", &mean);               /* read the corresponding mean */

    fprintf(file_out, "mean_value = %f  ", mean);

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);             /* adjust the pointer to the beginning
                                                     of the file "c10_detail.dat" */

    /* calculate the boundary of the band */

    up_boundary = mean + j;
    lw_boundary = mean - j;

    /* calculate the number of the small errors */

    for (i = 1; i < 81; i++)
      if( (c10[i-1] >= lw_boundary) && (c10[i-1] <= up_boundary) )
        number++;

    fprintf(file_out, "number = %d\n", number);

  }
}
```

F.1.3 Eighty Extracted C_{10} Values (file name:c10_data.dat)

```
0.362563 0.382896 0.365961 0.364807 0.365247 0.364622 0.378009 0.374774
0.337379 0.362562 0.364358 0.36599 0.374062 0.380752 0.383404 0.326109
0.371675 0.374101 0.369312 0.371127 0.377512 0.381166 0.397379 0.230942
0.368924 0.364903 0.368021 0.375926 0.367993 0.397168 0.381855 0.383397
0.394855 0.411779 0.324919 0.367993 0.381204 0.391402 0.401683 0.359261
0.365136 0.344007 0.366882 0.354401 0.360487 0.353281 0.353136 0.356968
0.363364 0.361773 0.355389 0.364088 0.362828 0.337045 0.358678 0.360806
0.360243 0.369753 0.362128 0.373589 0.366805 0.293263 0.373038 0.363664
0.361583 0.365701 0.355855 0.367136 0.367538 0.332379 0.364487 0.363581
0.365443 0.377038 0.380442 0.315894 0.263792 0.365288 0.242706 0.406539
```

F.1.4 Data File Containing C_{10} Statistics for Different Threshold Values (file name:c10_detail.dat)

```
C10 k=0.002 (k*k=x.xxxe-4)  ???   0.36567    ???    8.20412%
C10 k=0.004 (k*k=x.xxxe-4)  ???   0.365836    ???    8.2004%
C10 k=0.006 (k*k=x.xxxe-4)  ???   0.366069    ???    8.19517%
C10 k=0.008 (k*k=x.xxxe-4)  ???   0.366314    ???    8.1897%
C10 k=0.01 (k*k=x.xxxe-4)  ???   0.366413    ???    8.18748%
C10 k=0.012 (k*k=x.xxxe-4)  ???   0.366487    ???    8.18584%
C10 k=0.014 (k*k=x.xxxe-4)  ???   0.366586    ???    8.18362%
C10 k=0.016 (k*k=x.xxxe-4)  ???   0.366594    ???    8.18343%
C10 k=0.018 (k*k=x.xxxe-4)  ???   0.366498    ???    8.18557%
C10 k=0.020 (k*k=x.xxxe-4)  ???   0.366367    ???    8.1885%
C10 k=0.022 (k*k=x.xxxe-4)  ???   0.36624    ???    8.19134%
C10 k=0.024 (k*k=x.xxxe-4)  ???   0.366141    ???    8.19355%
C10 k=0.026 (k*k=x.xxxe-4)  ???   0.366034    ???    8.19595%
C10 k=0.028 (k*k=x.xxxe-4)  ???   0.365911    ???    8.19871%
C10 k=0.030 (k*k=x.xxxe-4)  ???   0.365815    ???    8.20087%
C10 k=0.032 (k*k=x.xxxe-4)  ???   0.365712    ???    8.20317%
C10 k=0.034 (k*k=x.xxxe-4)  ???   0.365579    ???    8.20616%
C10 k=0.036 (k*k=x.xxxe-4)  ???   0.365464    ???    8.20875%
C10 k=0.038 (k*k=x.xxxe-4)  ???   0.365325    ???    8.21186%
C10 k=0.040 (k*k=x.xxxe-4)  ???   0.365197    ???    8.21474%
C10 k=0.042 (k*k=x.xxxe-4)  ???   0.365104    ???    8.21684%
C10 k=0.044 (k*k=x.xxxe-4)  ???   0.364997    ???    8.21925%
C10 k=0.046 (k*k=x.xxxe-4)  ???   0.364887    ???    8.22171%
C10 k=0.048 (k*k=x.xxxe-4)  ???   0.364766    ???    8.22445%
C10 k=0.050 (k*k=x.xxxe-4)  ???   0.364649    ???    8.22708%
C10 k=0.052 (k*k=x.xxxe-4)  ???   0.364544    ???    8.22946%
C10 k=0.054 (k*k=x.xxxe-4)  ???   0.364439    ???    8.23183%
C10 k=0.056 (k*k=x.xxxe-4)  ???   0.364334    ???    8.23421%
C10 k=0.058 (k*k=x.xxxe-4)  ???   0.364228    ???    8.23659%
C10 k=0.060 (k*k=x.xxxe-4)  ???   0.364123    ???    8.23897%
C10 k=0.062 (k*k=x.xxxe-4)  ???   0.364018    ???    8.24136%
C10 k=0.064 (k*k=x.xxxe-4)  ???   0.363913    ???    8.24374%
C10 k=0.066 (k*k=x.xxxe-4)  ???   0.363807    ???    8.24612%
C10 k=0.068 (k*k=x.xxxe-4)  ???   0.363702    ???    8.24851%
C10 k=0.070 (k*k=x.xxxe-4)  ???   0.363597    ???    8.2509%
C10 k=0.072 (k*k=x.xxxe-4)  ???   0.363515    ???    8.25276%
C10 k=0.074 (k*k=x.xxxe-4)  ???   0.363437    ???    8.25453%
C10 k=0.076 (k*k=x.xxxe-4)  ???   0.363359    ???    8.2563%
C10 k=0.078 (k*k=x.xxxe-4)  ???   0.363281    ???    8.25807%
C10 k=0.08 (k*k=x.xxxe-4)  ???   0.363203    ???    8.25985%
C10 k=0.082 (k*k=x.xxxe-4)  ???   0.363125    ???    8.26162%
C10 k=0.084 (k*k=x.xxxe-4)  ???   0.363047    ???    8.26339%
C10 k=0.086 (k*k=x.xxxe-4)  ???   0.362969    ???    8.26517%
C10 k=0.088 (k*k=x.xxxe-4)  ???   0.362891    ???    8.26694%
C10 k=0.090 (k*k=x.xxxe-4)  ???   0.362813    ???    8.26872%
C10 k=0.092 (k*k=x.xxxe-4)  ???   0.362735    ???    8.27049%
C10 k=0.094 (k*k=x.xxxe-4)  ???   0.362657    ???    8.27227%
C10 k=0.096 (k*k=x.xxxe-4)  ???   0.36258    ???    8.27405%
C10 k=0.098 (k*k=x.xxxe-4)  ???   0.362502    ???    8.27583%
C10 k=0.100 (k*k=x.xxxe-4)  ???   0.362441    ???    8.2772%
C10 k=0.102 (k*k=x.xxxe-4)  ???   0.36239    ???    8.27837%
C10 k=0.104 (k*k=x.xxxe-4)  ???   0.362339    ???    8.27955%
C10 k=0.106 (k*k=x.xxxe-4)  ???   0.362287    ???    8.28072%
C10 k=0.108 (k*k=x.xxxe-4)  ???   0.362236    ???    8.28189%
C10 k=0.110 (k*k=x.xxxe-4)  ???   0.362185    ???    8.28306%
C10 k=0.112 (k*k=x.xxxe-4)  ???   0.362134    ???    8.28424%
C10 k=0.114 (k*k=x.xxxe-4)  ???   0.362082    ???    8.28541%
C10 k=0.116 (k*k=x.xxxe-4)  ???   0.362031    ???    8.28658%
C10 k=0.118 (k*k=x.xxxe-4)  ???   0.36198    ???    8.28776%
C10 k=0.120 (k*k=x.xxxe-4)  ???   0.361938    ???    8.2887%
C10 k=0.122 (k*k=x.xxxe-4)  ???   0.361913    ???    8.28928%
C10 k=0.124 (k*k=x.xxxe-4)  ???   0.361888    ???    8.28986%
C10 k=0.126 (k*k=x.xxxe-4)  ???   0.361862    ???    8.29044%
```

```
C10 k=0.128 (k*k=x.xxxe-4)  ???   0.361837   ???    8.29102%
C10 k=0.130 (k*k=x.xxxe-4)  ???   0.361812   ???    8.2916%
C10 k=0.132 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.134 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.136 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.138 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.140 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.142 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.144 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.146 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.148 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.150 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.152 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.154 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.156 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.158 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.160 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.162 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.164 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.166 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.168 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.170 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.172 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.174 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.176 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.178 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.180 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.182 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.184 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.186 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.188 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.190 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.192 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.194 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.196 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.198 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
C10 k=0.200 (k*k=x.xxxe-4)  ???   0.361801   ???    8.29185%
```

F.1.5 Output File from "number_c.c" Containing the Numbers of Small Errors (file name:number_c.dat)

```
C10
k=0.002  mean_value = 0.365670  number = 17
k=0.004  mean_value = 0.365836  number = 30
k=0.006  mean_value = 0.366069  number = 37
k=0.008  mean_value = 0.366314  number = 43
k=0.01  mean_value = 0.366413  number = 46
k=0.012  mean_value = 0.366487  number = 51
k=0.014  mean_value = 0.366586  number = 55
k=0.016  mean_value = 0.366594  number = 59
k=0.018  mean_value = 0.366498  number = 62
k=0.020  mean_value = 0.366367  number = 62
k=0.022  mean_value = 0.366240  number = 62
k=0.024  mean_value = 0.366141  number = 63
k=0.026  mean_value = 0.366034  number = 64
k=0.028  mean_value = 0.365911  number = 64
k=0.030  mean_value = 0.365815  number = 67
k=0.032  mean_value = 0.365712  number = 69
k=0.034  mean_value = 0.365579  number = 70
k=0.036  mean_value = 0.365464  number = 70
k=0.038  mean_value = 0.365325  number = 71
k=0.040  mean_value = 0.365197  number = 72
k=0.042  mean_value = 0.365104  number = 74
k=0.044  mean_value = 0.364997  number = 74
k=0.046  mean_value = 0.364887  number = 74
```

```
k=0.048  mean_value = 0.364766  number = 75
k=0.050  mean_value = 0.364649  number = 76
k=0.052  mean_value = 0.364544  number = 76
k=0.054  mean_value = 0.364439  number = 76
k=0.056  mean_value = 0.364334  number = 76
k=0.058  mean_value = 0.364228  number = 76
k=0.060  mean_value = 0.364123  number = 76
k=0.062  mean_value = 0.364018  number = 76
k=0.064  mean_value = 0.363913  number = 76
k=0.066  mean_value = 0.363807  number = 76
k=0.068  mean_value = 0.363702  number = 76
k=0.070  mean_value = 0.363597  number = 76
k=0.072  mean_value = 0.363515  number = 77
k=0.074  mean_value = 0.363437  number = 77
k=0.076  mean_value = 0.363359  number = 77
k=0.078  mean_value = 0.363281  number = 77
k=0.08  mean_value = 0.363203  number = 77
k=0.082  mean_value = 0.363125  number = 77
k=0.084  mean_value = 0.363047  number = 77
k=0.086  mean_value = 0.362969  number = 77
k=0.088  mean_value = 0.362891  number = 77
k=0.090  mean_value = 0.362813  number = 77
k=0.092  mean_value = 0.362735  number = 77
k=0.094  mean_value = 0.362657  number = 77
k=0.096  mean_value = 0.362580  number = 77
k=0.098  mean_value = 0.362502  number = 77
k=0.100  mean_value = 0.362441  number = 78
k=0.102  mean_value = 0.362390  number = 78
k=0.104  mean_value = 0.362339  number = 78
k=0.106  mean_value = 0.362287  number = 78
k=0.108  mean_value = 0.362236  number = 78
k=0.110  mean_value = 0.362185  number = 78
k=0.112  mean_value = 0.362134  number = 78
k=0.114  mean_value = 0.362082  number = 78
k=0.116  mean_value = 0.362031  number = 78
k=0.118  mean_value = 0.361980  number = 78
k=0.120  mean_value = 0.361938  number = 79
k=0.122  mean_value = 0.361913  number = 79
k=0.124  mean_value = 0.361888  number = 79
k=0.126  mean_value = 0.361862  number = 79
k=0.128  mean_value = 0.361837  number = 79
k=0.130  mean_value = 0.361812  number = 79
k=0.132  mean_value = 0.361801  number = 80
k=0.134  mean_value = 0.361801  number = 80
k=0.136  mean_value = 0.361801  number = 80
k=0.138  mean_value = 0.361801  number = 80
k=0.140  mean_value = 0.361801  number = 80
k=0.142  mean_value = 0.361801  number = 80
k=0.144  mean_value = 0.361801  number = 80
k=0.146  mean_value = 0.361801  number = 80
k=0.148  mean_value = 0.361801  number = 80
k=0.150  mean_value = 0.361801  number = 80
k=0.152  mean_value = 0.361801  number = 80
k=0.154  mean_value = 0.361801  number = 80
k=0.156  mean_value = 0.361801  number = 80
k=0.158  mean_value = 0.361801  number = 80
k=0.160  mean_value = 0.361801  number = 80
k=0.162  mean_value = 0.361801  number = 80
k=0.164  mean_value = 0.361801  number = 80
k=0.166  mean_value = 0.361801  number = 80
k=0.168  mean_value = 0.361801  number = 80
k=0.170  mean_value = 0.361801  number = 80
k=0.172  mean_value = 0.361801  number = 80
k=0.174  mean_value = 0.361801  number = 80
k=0.176  mean_value = 0.361801  number = 80
k=0.178  mean_value = 0.361801  number = 80
k=0.180  mean_value = 0.361801  number = 80
```

```
k=0.182  mean_value = 0.361801  number = 80
k=0.184  mean_value = 0.361801  number = 80
k=0.186  mean_value = 0.361801  number = 80
k=0.188  mean_value = 0.361801  number = 80
k=0.190  mean_value = 0.361801  number = 80
k=0.192  mean_value = 0.361801  number = 80
k=0.194  mean_value = 0.361801  number = 80
k=0.196  mean_value = 0.361801  number = 80
k=0.198  mean_value = 0.361801  number = 80
k=0.020  mean_value = 0.361801  number = 80
```

F.2 Parameter τ

F.2.1 The Circuit File

```
! file tau.ckt

expression

  i = 1;                 ! index
  k = i / 100;           ! rescale for correct display interval

  ! number of small errors for different threshold
  ! numbers obtained from the file "number_t.dat" which is
  ! an output file of the C program "number_t.c"

  tau_number[100] = [
        18 23 30 33 40 44 48 49 54 58
        59 60 64 64 64 66 66 67 68 70
        70 70 70 70 70 70 70 70 70 70
        70 70 70 70 70 70 70 70 72 72
        72 72 72 72 72 72 72 72 72 72
        72 72 72 72 72 72 72 72 73 73
        73 73 73 73 73 74 74 74 74 74
        74 74 74 74 74 74 74 74 74 74
        74 74 74 74 74 74 74 74 74 74
        74 75 75 75 75 75 75 75 75 75
        ];

  ! calculate the percentage
  tau_percent[100] = 100 * tau_number / 80;
end

sweep
i: from 1 to 100 step 1 k tau_percent[i]
{parametric Title = "percentage of assumed small errors for TAU"
            X = k  Y = tau_percent[i]
            Ymin = 0  Ymax = 100 NYticks = 4
            Xmin = 0  Xmax = 1   NXticks = 4
            X_title = "k"  Y_title = "percentage"};
end
```

F.2.2 C Program to Calculate the Number of Small Errors

```
/* File: number_t.c */
#include "stdio.h"
#include "string.h"
#include "math.h"

main ()

{

  int i, number;
  float j, mean,up_boundary, lw_boundary, tau[80];
  char k_value[20];
  FILE *file_data = NULL;
  FILE *file_mean = NULL;
  FILE *file_out = NULL;

  file_data = fopen("tau_data.dat", "r");         /* contains the extracted parameters */
  file_mean = fopen("tau_detail.dat", "r");       /* contains the estimated means and */
                                                  /* standard deviations from OSA90/hope */
  file_out = fopen("number_tau.dat", "w");        /* contains the number of small errors */
  for (i = 1; i < 81; i++)
    fscanf(file_data, "%f", &tau[i-1]);          /* reads in the extracted parameters */

  fprintf(file_out, "TAU\n");                    /* a reminder */

  for (j = 0.01; j < 1.02; j = j + 0.01) {

    number = 0;

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);            /* to read correct text, skip 2 col's */

    fprintf(file_out, "%s  ", k_value);          /* output the current threshold value */

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);            /* to read correct text, skip 2 col's */

    fscanf(file_mean, "%f", &mean);              /* read the corresponding mean */

    fprintf(file_out, "mean_value = %f  ", mean);

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);            /* adjust the pointer to the beginning
                                                    of the file "c10_detail.dat" */

    /* calculate the boundary of the band */

    up_boundary = mean + j;
    lw_boundary = mean - j;

    /* calculate the number of the small errors */

    for (i = 1; i < 81; i++)
      if( (tau[i-1] >= lw_boundary) && (tau[i-1] <= up_boundary) )
        number++;

    fprintf(file_out, "number = %d\n", number);

  }
}
```

F.2.3 Eighty Extracted τ Values (file name:tau_data.dat)

```
2.12742 2.18777 2.17859 2.12656 2.17465 2.1775 2.19197 2.21798
2.33714 2.15528 2.18736 2.2225 2.22436 2.21208 2.22253 1.45189
2.17705 2.18329 2.20177 2.22969 2.31806 2.29611 2.27188 1.75367
2.20324 2.22871 2.26579 2.29229 2.19074 2.29612 2.20923 2.21387
2.28671 2.35304 1.17145 0.0636748 2.20641 2.29232 2.34982 2.08324
2.07417 2.11242 2.19065 2.13068 2.0772 2.08495 2.08396 2.1062
2.25574 2.09906 2.11657 2.07227 2.12352 1.96091 2.19077 2.18401
2.18293 2.1715 2.22586 2.00958 2.18663 0.132958 2.15652 2.18759
2.20099 2.21515 2.17997 2.19088 2.19255 0.29736 2.15612 2.13826
2.19294 2.24055 2.26081 1.75062 0.0063146 2.21265 0.00329595 1.52677
```

F.2.4 Data File Containing τ Statistics for Different Threshold Values (file name:tau_detail.dat)

```
TAU k=0.01 (k*k=0.0001)  ???   2.18518   ???   2.45378%
TAU k=0.02 (k*k=0.0004)  ???   2.18458   ???   2.32971%
TAU k=0.03 (k*k=0.0009)  ???   2.18312   ???   2.31765%
TAU k=0.04 (k*k=0.0016)  ???   2.18126   ???   2.3834%
TAU k=0.05 (k*k=0.0025)  ???   2.17884   ???   2.51659%
TAU k=0.06 (k*k=0.0036)  ???   2.17683   ???   2.73397%
TAU k=0.07 (k*k=0.0049)   ???    2.17535    ???     2.99408%
TAU k=0.08 (k*k=0.0064)   ???    2.17421    ???     3.20145%
TAU k=0.09 (k*k=0.0081)   ???    2.17305    ???     3.39659%
TAU k=0.1 (k*k=0.01)   ???    2.17231    ???    3.56784%
TAU k=0.11 (k*k=0.0121)   ???    2.1718    ???    3.72553%
TAU k=0.12 (k*k=0.0144)   ???    2.17121    ???     3.89158%
TAU k=0.13 (k*k=0.0169)   ???    2.17015    ???     4.05931%
TAU k=0.14 (k*k=0.0196)   ???    2.16889    ???     4.22188%
TAU k=0.15 (k*k=0.0225)   ???    2.16765    ???     4.37681%
TAU k=0.16 (k*k=0.0256)   ???    2.16632    ???     4.53111%
TAU k=0.17 (k*k=0.0289)   ???    2.16511    ???     4.66948%
TAU k=0.18 (k*k=0.0324)   ???    2.1638    ???    4.80911%
TAU k=0.19 (k*k=0.0361)   ???    2.16242    ???     4.94393%
TAU k=0.2 (k*k=0.04)   ???    2.16083    ???    5.08428%
TAU k=0.21 (k*k=0.0441)   ???    2.1594    ???    5.22785%
TAU k=0.22 (k*k=0.0484)   ???    2.15798    ???     5.37332%
TAU k=0.23 (k*k=0.0529)   ???    2.15655    ???     5.52183%
TAU k=0.24 (k*k=0.0576)   ???    2.15511    ???     5.67251%
TAU k=0.25 (k*k=0.0625)   ???    2.15369    ???     5.82672%
TAU k=0.26 (k*k=0.0676)   ???    2.15226    ???     5.98234%
TAU k=0.27 (k*k=0.0729)   ???    2.15082    ???     6.14113%
TAU k=0.28 (k*k=0.0784)   ???    2.1494    ???    6.30084%
TAU k=0.29 (k*k=0.0841)   ???    2.14798    ???     6.46321%
TAU k=0.3 (k*k=0.09)   ???    2.14655    ???    6.62716%
TAU k=0.31 (k*k=0.0961)   ???    2.14511    ???     6.79297%
TAU k=0.32 (k*k=0.1024)   ???    2.14368    ???     6.9602%
TAU k=0.33 (k*k=0.1089)   ???    2.14226    ???     7.12875%
TAU k=0.34 (k*k=0.1156)   ???    2.14083    ???     7.29832%
TAU k=0.35 (k*k=0.1225)   ???    2.13941    ???     7.46964%
TAU k=0.36 (k*k=0.1296)   ???    2.13797    ???     7.61286%
TAU k=0.37 (k*k=0.1369)   ???    2.13654    ???     7.75046%
TAU k=0.38 (k*k=0.1444)   ???    2.13512    ???     7.88894%
TAU k=0.39 (k*k=0.1521)   ???    2.13393    ???     8.02538%
TAU k=0.4 (k*k=0.16)   ???    2.1328    ???    8.16176%
TAU k=0.41 (k*k=0.1681)   ???    2.1317    ???    8.29997%
TAU k=0.42 (k*k=0.1764)   ???    2.13059    ???     8.43909%
TAU k=0.43 (k*k=0.1849)   ???    2.12948    ???     8.57905%
TAU k=0.44 (k*k=0.1936)   ???    2.12836    ???     8.72069%
TAU k=0.45 (k*k=0.2025)   ???    2.12725    ???     8.86328%
TAU k=0.46 (k*k=0.2116)   ???    2.12614    ???     9.00722%
TAU k=0.47 (k*k=0.2209)   ???    2.12504    ???     9.15143%
TAU k=0.48 (k*k=0.2304)   ???    2.12393    ???     9.29705%
TAU k=0.49 (k*k=0.2401)   ???    2.12282    ???     9.44364%
TAU k=0.5 (k*k=0.25)   ???    2.12169    ???    9.59151%
TAU k=0.51 (k*k=0.2601)   ???    2.12058    ???     9.74018%
TAU k=0.52 (k*k=0.2704)   ???    2.11947    ???     9.88943%
TAU k=0.53 (k*k=0.2809)   ???    2.11837    ???     10.0392%
TAU k=0.54 (k*k=0.2916)   ???    2.11726    ???     10.1902%
TAU k=0.55 (k*k=0.3025)   ???    2.11614    ???     10.3375%
TAU k=0.56 (k*k=0.3136)   ???    2.11503    ???     10.4698%
TAU k=0.57 (k*k=0.3249)   ???    2.11392    ???     10.603%
TAU k=0.58 (k*k=0.3364)   ???    2.1128    ???    10.7373%
TAU k=0.59 (k*k=0.3481)   ???    2.11176    ???     10.8693%
TAU k=0.6 (k*k=0.36)   ???    2.1108    ???    11.0043%
TAU k=0.61 (k*k=0.3721)   ???    2.10986    ???     11.1379%
TAU k=0.62 (k*k=0.3844)   ???    2.1089    ???    11.2602%
TAU k=0.63 (k*k=0.3969)   ???    2.10793    ???     11.3791%
TAU k=0.64 (k*k=0.4096)   ???    2.10697    ???     11.4971%
TAU k=0.65 (k*k=0.4225)   ???    2.10602    ???     11.6124%
TAU k=0.66 (k*k=0.4356)   ???    2.10516    ???     11.7272%
TAU k=0.67 (k*k=0.4489)   ???    2.10435    ???     11.8419%
TAU k=0.68 (k*k=0.4624)   ???    2.10353    ???     11.9574%
TAU k=0.69 (k*k=0.4761)   ???    2.10272    ???     12.076%
TAU k=0.7 (k*k=0.49)   ???    2.10191    ???    12.1903%
TAU k=0.71 (k*k=0.5041)   ???    2.10109    ???     12.3133%
TAU k=0.72 (k*k=0.5184)   ???    2.10029    ???     12.4292%
TAU k=0.73 (k*k=0.3295)   ???    2.09947    ???     12.553%
TAU k=0.74 (k*k=0.5476)   ???   2.09867   ???    12.6736%
TAU k=0.75 (k*k=0.5625)   ???   2.09786   ???    12.7837%
TAU k=0.76 (k*k=0.5776)   ???   2.09705   ???    12.9169%
TAU k=0.77 (k*k=0.5929)   ???   2.09622   ???    13.0257%
TAU k=0.78 (k*k=0.6084)   ???   2.09542   ???    13.1627%
TAU k=0.79 (k*k=0.6241)   ???   2.09462   ???    13.2862%
TAU k=0.8 (k*k=0.64)   ???   2.09379   ???    13.4109%
TAU k=0.81 (k*k=0.6561)   ???   2.09298   ???    13.5354%
TAU k=0.82 (k*k=0.6724)   ???   2.09217   ???    13.6606%
TAU k=0.83 (k*k=0.6889)   ???   2.09137   ???    13.7863%
TAU k=0.84 (k*k=0.7056)   ???   2.09056   ???    13.9124%
TAU k=0.85 (k*k=0.7225)   ???   2.08975   ???    14.0456%
TAU k=0.86 (k*k=0.7396)   ???   2.08894   ???    14.1665%
TAU k=0.87 (k*k=0.7569)   ???   2.08813   ???    14.2879%
TAU k=0.88 (k*k=0.7744)   ???   2.08731   ???    14.3944%
TAU k=0.89 (k*k=0.7921)   ???   2.08651   ???    14.5011%
TAU k=0.9 (k*k=0.81)   ???   2.0857   ???    14.6086%
TAU k=0.91 (k*k=0.8281)   ???   2.08489   ???    14.7167%
TAU k=0.92 (k*k=0.8464)   ???   2.08417   ???    14.8235%
TAU k=0.93 (k*k=0.8649)   ???   2.08349   ???    14.9305%
TAU k=0.94 (k*k=0.8836)   ???   2.08284   ???    15.0374%
TAU k=0.95 (k*k=0.9025)   ???   2.08218   ???    15.1448%
TAU k=0.96 (k*k=0.9216)   ???   2.0815   ???    15.2526%
TAU k=0.97 (k*k=0.9409)   ???   2.08084   ???    15.3609%
TAU k=0.98 (k*k=0.9604)   ???   2.08016   ???    15.47%
TAU k=0.99 (k*k=0.9801)   ???   2.07951   ???    15.579%
TAU k=1 (k*k=1)   ???   2.07882   ???    15.6893%
TAU k=1.01 (k*k=1.0201)   ???   2.07818   ???    15.799%
```

F.2.5 Output File from "number_t.c" Containing the Numbers of Small Errors (file name:number_t.dat)

```
TAU
k=0.01  mean_value = 2.185180  number = 18
k=0.02  mean_value = 2.184580  number = 23
k=0.03  mean_value = 2.183120  number = 30
k=0.04  mean_value = 2.181260  number = 33
k=0.05  mean_value = 2.178840  number = 40
k=0.06  mean_value = 2.176830  number = 44
k=0.07  mean_value = 2.175350  number = 48
k=0.08  mean_value = 2.174210  number = 49
k=0.09  mean_value = 2.173050  number = 54
k=0.1  mean_value = 2.172310  number = 58
k=0.11  mean_value = 2.171800  number = 59
k=0.12  mean_value = 2.171210  number = 60
k=0.13  mean_value = 2.170150  number = 64
k=0.14  mean_value = 2.168890  number = 64
k=0.15  mean_value = 2.167650  number = 64
k=0.16  mean_value = 2.166320  number = 66
k=0.17  mean_value = 2.165110  number = 66
k=0.18  mean_value = 2.163800  number = 67
k=0.19  mean_value = 2.162420  number = 68
k=0.2  mean_value = 2.160830  number = 70
k=0.21  mean_value = 2.159400  number = 70
k=0.22  mean_value = 2.157980  number = 70
k=0.23  mean_value = 2.156550  number = 70
k=0.24  mean_value = 2.155110  number = 70
k=0.25  mean_value = 2.153690  number = 70
k=0.26  mean_value = 2.152260  number = 70
k=0.27  mean_value = 2.150820  number = 70
k=0.28  mean_value = 2.149400  number = 70
k=0.29  mean_value = 2.147980  number = 70
k=0.3  mean_value = 2.146550  number = 70
k=0.31  mean_value = 2.145110  number = 70
k=0.32  mean_value = 2.143680  number = 70
```

```
k=0.33  mean_value = 2.142260  number = 70
k=0.34  mean_value = 2.140830  number = 70
k=0.35  mean_value = 2.139410  number = 70
k=0.36  mean_value = 2.137970  number = 70
k=0.37  mean_value = 2.136540  number = 70
k=0.38  mean_value = 2.135120  number = 70
k=0.39  mean_value = 2.133930  number = 72
k=0.4  mean_value = 2.132800  number = 72
k=0.41  mean_value = 2.131700  number = 72
k=0.42  mean_value = 2.130590  number = 72
k=0.43  mean_value = 2.129480  number = 72
k=0.44  mean_value = 2.128360  number = 72
k=0.45  mean_value = 2.127250  number = 72
k=0.46  mean_value = 2.126140  number = 72
k=0.47  mean_value = 2.125040  number = 72
k=0.48  mean_value = 2.123930  number = 72
k=0.49  mean_value = 2.122820  number = 72
k=0.5  mean_value = 2.121690  number = 72
k=0.51  mean_value = 2.120580  number = 72
k=0.52  mean_value = 2.119470  number = 72
k=0.53  mean_value = 2.118370  number = 72
k=0.54  mean_value = 2.117260  number = 72
k=0.55  mean_value = 2.116140  number = 72
k=0.56  mean_value = 2.115030  number = 72
k=0.57  mean_value = 2.113920  number = 72
k=0.58  mean_value = 2.112800  number = 72
k=0.59  mean_value = 2.111760  number = 73
k=0.6  mean_value = 2.110800  number = 73
k=0.61  mean_value = 2.109860  number = 73
k=0.62  mean_value = 2.108900  number = 73
k=0.63  mean_value = 2.107930  number = 73
k=0.64  mean_value = 2.106970  number = 73
k=0.65  mean_value = 2.106020  number = 73
k=0.66  mean_value = 2.105160  number = 74
k=0.67  mean_value = 2.104350  number = 74
k=0.68  mean_value = 2.103530  number = 74
k=0.69  mean_value = 2.102720  number = 74
k=0.7  mean_value = 2.101910  number = 74
k=0.71  mean_value = 2.101090  number = 74
k=0.72  mean_value = 2.100290  number = 74
k=0.73  mean_value = 2.099470  number = 74
k=0.74  mean_value = 2.098670  number = 74
k=0.75  mean_value = 2.097860  number = 74
k=0.76  mean_value = 2.097050  number = 74
k=0.77  mean_value = 2.096220  number = 74
k=0.78  mean_value = 2.095420  number = 74
k=0.79  mean_value = 2.094620  number = 74
k=0.8  mean_value = 2.093790  number = 74
k=0.81  mean_value = 2.092980  number = 74
k=0.82  mean_value = 2.092170  number = 74
k=0.83  mean_value = 2.091370  number = 74
k=0.84  mean_value = 2.090560  number = 74
k=0.85  mean_value = 2.089750  number = 74
k=0.86  mean_value = 2.088940  number = 74
k=0.87  mean_value = 2.088130  number = 74
k=0.88  mean_value = 2.087310  number = 74
k=0.89  mean_value = 2.086510  number = 74
k=0.9  mean_value = 2.085700  number = 74
k=0.91  mean_value = 2.084890  number = 74
k=0.92  mean_value = 2.084170  number = 75
k=0.93  mean_value = 2.083490  number = 75
k=0.94  mean_value = 2.082840  number = 75
k=0.95  mean_value = 2.082180  number = 75
k=0.96  mean_value = 2.081500  number = 75
k=0.97  mean_value = 2.080840  number = 75
k=0.98  mean_value = 2.080160  number = 75
k=0.99  mean_value = 2.079510  number = 75
```

```
k=1  mean_value = 2.078820  number = 75
k=1.01  mean_value = 2.078180  number = 75
```

F.3 Parameter L_g

F.3.1 The Circuit File

```
!  file lg.ckt

expression

  i = 1;                    ! index
  k = i / 2000;             ! rescale for correct display interval

  !  number of small errors for different threshold
  !  numbers obtained from the file "number_l.dat" which is
  !  an output file of the C program "number_l.c"

  lg_number[100] = [
          7 12 16 20 22 28 34 37 38 44
         49 52 53 54 57 58 59 61 65 67
         68 69 70 70 70 70 71 71 72 72
         73 73 73 73 73 73 73 73 73 73
         74 74 74 74 74 74 74 74 74 74
         74 74 74 74 74 75 75 75 75 75
         75 75 75 75 75 75 75 75 75 75
         75 75 75 75 75 75 75 75 75 75
         75 75 75 75 75 75 75 75 75 75
         75 75 75 75 75 75 75 75 75 75
         ];

  !  calculate the percentage
  lg_percent[100] = 100 * lg_number / 80;
end

sweep
i: from 1 to 100 step 1 k lg_percent[i]
{parametric Title = "percentage of assumed small errors for LG"
            X = k  Y = lg_percent[i]
            Ymin = 0  Ymax = 100  NYticks = 4
            Xmin = 0  Xmax = 0.05 NXticks = 4
            X_title = "k"  Y_title = "percentage"};
end
```

F.3.2 C Program to Calculate the Number of Small Errors

```
/* File: number_l.c */
#include "stdio.h"
#include "string.h"
#include "math.h"

main ()

{
  int i, number;
  float j, mean,up_boundary, lw_boundary, lg[80];
  char k_value[20];
  FILE *file_data = NULL;
  FILE *file_mean = NULL;
  FILE *file_out = NULL;

  file_data = fopen("lg_data.dat", "r");          /* contains the extracted parameters */
  file_mean = fopen("lg_detail.dat", "r");        /* contains the estimated means and */
                                                  /* standard deviations from OSA90/hope */
  file_out = fopen("number_lg.dat", "w");         /* contains the number of small errors */
  for (i = 1; i < 81; i++)
    fscanf(file_data, "%f", &lg[i-1]);          /* reads in the extracted parameters */

  fprintf(file_out, "LG\n");                    /* a reminder */

  for (j = 0.0005; j < 0.0505; j = j + 0.0005) {

    number = 0;

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);           /* to read correct text, skip 2 col's */

    fprintf(file_out, "%s  ", k_value);         /* output the current threshold value */

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);           /* to read correct text, skip 2 col's */

    fscanf(file_mean, "%f", &mean);             /* read the corresponding mean */

    fprintf(file_out, "mean_value = %f  ", mean);

    for (i = 1; i < 3; i++)
    fscanf(file_mean, "%s", k_value);           /* adjust the pointer to the beginning
                                                   of the file "lg_detail.dat" */

    /* calculate the boundary of the band */

    up_boundary = mean + j;
    lw_boundary = mean - j;

    /* calculate the number of the small errors */

    for (i = 1; i < 81; i++)
      if( (lg[i-1] >= lw_boundary) && (lg[i-1] <= up_boundary) )
        number++;

    fprintf(file_out, "number = %d\n", number);
  }
}
```

F.3.3 Eighty Extracted L_g Values (file name:lg_data.dat)

```
0.0249343 0.0382752 0.0244437 0.0305276 0.0320525 0.0351226 0.0384052 0.0404104
0.0203612 0.0295276 0.0338138 0.0378106 0.0401943 0.0439019 0.0400248 0.19135
0.0253422 0.0309073 0.0351278 0.0362827 0.0361359 0.040135 0.0377837 0.00740458
0.0277345 0.0336417 0.0338764 0.0354762 0.0342489 0.0377881 0.0304654 0.0321601
0.0340442 0.0361249 0.198991 0.0291953 0.0262987 0.026106 0.024076 0.0352524
0.0397872 0.0311892 0.0342306 0.0215204 0.0439407 0.0443833 0.0302681 0.0349599
0.0146805 0.0396286 0.0420346 0.0426165 0.0406111 0.0411889 0.0255615 0.0235038
0.0326257 0.037303 0.0387938 0.0378664 0.0419138 0.23049 0.02803 0.0333612
0.035629 0.03844 0.0375648 0.0342395 0.0396309 0.0257349 0.0290948 0.0318604
0.0368525 0.0378124 0.0357473 0.192996 0.189776 0.0312339 0.0367897 0.0195684
```

F.3.4 Data File Containing L_g Statistics for Different Threshold Values (file name:lg_detail.dat)

```
LG k=0.0005 (k*k=2.5e-7)   ???    0.035257    ???     41.0532%
LG k=0.001 (k*k=1e-6)   ???    0.0351909    ???     36.5016%
LG k=0.0015 (k*k=2.25e-6)   ???    0.0352069    ???     15.359%
LG k=0.002 (k*k=x.xxxe-4)   ???    0.0352478    ???     42.5559%
LG k=0.0025 (k*k=6.25e-6)   ???    0.0352647    ???     12.9723%
LG k=0.003 (k*k=9e-6)   ???    0.0351938    ???     12.7343%
LG k=0.0035 (k*k=1.225e-5)   ???    0.0351363    ???     12.7979%
LG k=0.004 (k*k=1.6e-5)  ???   0.0350772   ???    13.0777%
LG k=0.0045 (k*k=2.025e-5)  ???   0.0350465   ???    13.4091%
LG k=0.005 (k*k=2.5e-5)  ???   0.0350246   ???    14.0233%
LG k=0.0055 (k*k=3.025e-5)  ???   0.0349771   ???    14.6756%
LG k=0.006 (k*k=3.6e-5)  ???   0.0349285   ???    15.3166%
LG k=0.0065 (k*k=4.225e-5)  ???   0.0348863   ???    15.9721%
LG k=0.007 (k*k=4.9e-5)  ???   0.0348427   ???    16.5859%
LG k=0.0075 (k*k=5.625e-5)  ???   0.0348016   ???    17.0682%
LG k=0.008 (k*k=6.4e-5)  ???   0.0347553   ???    17.4792%
LG k=0.0085 (k*k=7.225e-5)  ???   0.0347052   ???    17.8692%
LG k=0.009 (k*k=8.1e-5)  ???   0.0346713   ???    18.2168%
LG k=0.0095 (k*k=9.025e-5)  ???   0.0346492   ???    18.5394%
LG k=0.01 (k*k=1e-4)  ???   0.0346271   ???    18.8613%
LG k=0.0105 (k*k=1.1025e-4)  ???   0.0346097   ???    19.1918%
LG k=0.011 (k*k=1.21e-4)  ???   0.034602   ???    19.5273%
LG k=0.0115 (k*k=1.3225e-4)  ???   0.0346006   ???    19.8438%
LG k=0.012 (k*k=1.44e-4)  ???   0.0346006   ???    20.1548%
LG k=0.0125 (k*k=1.5625e-4)  ???   0.0346006   ???    20.4635%
LG k=0.013 (k*k=1.69e-4)  ???   0.0346006   ???    20.7507%
LG k=0.0135 (k*k=1.8225e-4)  ???   0.0346065   ???    21.0172%
LG k=0.014 (k*k=1.96e-4)  ???   0.0346136   ???    21.2748%
LG k=0.0145 (k*k=2.1025e-4)  ???   0.034624   ???    21.5371%
LG k=0.015 (k*k=2.25e-4)  ???   0.0346379   ???    21.8037%
LG k=0.0155 (k*k=2.4025e-4)  ???   0.0346575   ???    22.0736%
LG k=0.016 (k*k=2.56e-4)  ???   0.034678   ???   22.3483%
LG k=0.0165 (k*k=2.7225e-4)  ???   0.0346986   ???    22.6279%
LG k=0.017 (k*k=2.89e-4)  ???   0.0347191   ???    22.9123%
LG k=0.0175 (k*k=3.0625e-4)  ???   0.0347396   ???    23.2013%
LG k=0.018 (k*k=3.24e-4)  ???   0.0347602   ???    23.4948%
LG k=0.0185 (k*k=3.4225e-4)  ???   0.0347807   ???    23.7763%
LG k=0.019 (k*k=3.61e-4)  ???   0.0348013   ???    24.0322%
LG k=0.0195 (k*k=3.8025e-4)  ???   0.0348218   ???    24.2919%
LG k=0.0205 (k*k=4.2025e-4)  ???   0.0348672   ???    24.8202%
LG k=0.021 (k*k=4.41e-4)  ???   0.0348942   ???    25.0879%
LG k=0.0215 (k*k=4.6225e-4)  ???   0.0349213   ???    25.3587%
LG k=0.022 (k*k=4.84e-4)  ???   0.0349483   ???    25.6326%
LG k=0.0225 (k*k=5.0625e-4)  ???   0.0349753   ???    25.9093%
LG k=0.023 (k*k=5.29e-4)  ???   0.0350023   ???    26.1888%
LG k=0.0235 (k*k=5.5225e-4)  ???   0.0350294   ???    26.4709%
LG k=0.024 (k*k=x.xxxe-4)  ???   0.0350564   ???    42.7882%
LG k=0.0245 (k*k=x.xxxe-4)  ???   0.0350834   ???    42.7552%
LG k=0.025 (k*k=x.xxxe-4)  ???   0.0351104   ???    42.7223%
LG k=0.0255 (k*k=x.xxxe-4)  ???   0.0351375   ???    42.6895%
LG k=0.026 (k*k=x.xxxe-4)  ???   0.0351645   ???    42.6567%
LG k=0.0265 (k*k=x.xxxe-4)  ???   0.0351915   ???    42.6239%
LG k=0.027 (k*k=x.xxxe-4)  ???   0.0352185   ???    42.5912%
LG k=0.0275 (k*k=x.xxxe-4)  ???   0.0352455   ???    42.5586%
LG k=0.028 (k*k=x.xxxe-4)  ???   0.0352744   ???    42.5238%
LG k=0.0285 (k*k=x.xxxe-4)  ???   0.0353077   ???    42.4837%
LG k=0.029 (k*k=x.xxxe-4)  ???   0.035341   ???    42.4436%
LG k=0.0295 (k*k=x.xxxe-4)  ???   0.0353743   ???    42.4036%
LG k=0.03 (k*k=x.xxxe-4)  ???   0.0354077   ???    42.3637%
LG k=0.0305 (k*k=x.xxxe-4)  ???   0.035441   ???    42.3239%
LG k=0.031 (k*k=x.xxxe-4)  ???   0.0354743   ???    42.2841%
LG k=0.0315 (k*k=x.xxxe-4)  ???   0.0355077   ???    42.2444%
LG k=0.0320 (k*k=x.xxxe-4)  ???   0.035541   ???    42.2048%
LG k=0.0325 (k*k=x.xxxe-4)  ???   0.0355743   ???    42.1652%
LG k=0.0330 (k*k=x.xxxe-4)  ???   0.0356077   ???    42.1258%
LG k=0.0335 (k*k=x.xxxe-4)  ???   0.035641   ???    42.0864%
LG k=0.0340 (k*k=x.xxxe-4)  ???   0.0356743   ???    42.0471%
LG k=0.0345 (k*k=x.xxxe-4)  ???   0.0357076   ???    42.0078%
LG k=0.0350 (k*k=x.xxxe-4)  ???   0.035741   ???    41.9686%
LG k=0.0355 (k*k=x.xxxe-4)  ???   0.0357743   ???    41.9295%
LG k=0.0360 (k*k=x.xxxe-4)  ???   0.0358076   ???    41.8905%
LG k=0.0365 (k*k=x.xxxe-4)  ???   0.035841   ???    41.8515%
LG k=0.0370 (k*k=x.xxxe-4)  ???   0.0358743   ???    41.8127%
LG k=0.0375 (k*k=x.xxxe-4)  ???   0.0359076   ???    41.7739%
LG k=0.0380 (k*k=x.xxxe-4)  ???  0.035941  ???  41.7351%
LG k=0.0385 (k*k=x.xxxe-4)  ???  0.0359743  ???  41.6964%
LG k=0.0390 (k*k=x.xxxe-4)  ???  0.0360076  ???  41.6578%
LG k=0.0395 (k*k=x.xxxe-4)  ???  0.036041  ???  41.6193%
LG k=0.0400 (k*k=x.xxxe-4)  ???  0.0360743  ???  41.5809%
LG k=0.0405 (k*k=x.xxxe-4)  ???  0.0361076  ???  41.5425%
LG k=0.0410 (k*k=x.xxxe-4)  ???  0.0361409  ???  41.5042%
LG k=0.0415 (k*k=x.xxxe-4)  ???  0.0361743  ???  41.4659%
LG k=0.0420 (k*k=x.xxxe-4)  ???  0.0362076  ???  41.4278%
LG k=0.0425 (k*k=x.xxxe-4)  ???  0.0362409  ???  41.3897%
LG k=0.0430 (k*k=x.xxxe-4)  ???  0.0362743  ???  41.3516%
LG k=0.0435 (k*k=x.xxxe-4)  ???  0.0363076  ???  41.3137%
LG k=0.0440 (k*k=x.xxxe-4)  ???  0.0363409  ???  41.2758%
LG k=0.0445 (k*k=x.xxxe-4)  ???  0.0364409  ???  41.1625%
LG k=0.0450 (k*k=x.xxxe-4)  ???  0.0364076  ???  41.2002%
LG k=0.0455 (k*k=x.xxxe-4)  ???  0.0364409  ???  41.1625%
LG k=0.0460 (k*k=x.xxxe-4)  ???  0.0364743  ???  41.1249%
LG k=0.0465 (k*k=x.xxxe-4)  ???  0.0365076  ???  41.0874%
LG k=0.0470 (k*k=x.xxxe-4)  ???  0.0365409  ???  41.0499%
LG k=0.0475 (k*k=x.xxxe-4)  ???  0.0365742  ???  41.0125%
LG k=0.0480 (k*k=x.xxxe-4)  ???  0.0366076  ???  40.9751%
LG k=0.0485 (k*k=x.xxxe-4)  ???  0.0366409  ???  40.9379%
LG k=0.0490 (k*k=x.xxxe-4)  ???  0.0366742  ???  40.9007%
LG k=0.0495 (k*k=x.xxxe-4)  ???  0.0367076  ???  40.8635%
LG k=0.0500 (k*k=x.xxxe-4)  ???  0.0367409  ???  40.8264%
```

F.3.5 Output File from "number_l.c" Containing the Numbers of Small Errors (file name:number_l.dat)

```
LG
k=0.0005  mean_value = 0.035257  number = 7
k=0.001  mean_value = 0.035191  number = 12
k=0.0015  mean_value = 0.035207  number = 16
k=0.002  mean_value = 0.035248  number = 20
k=0.0025  mean_value = 0.035265  number = 22
k=0.003  mean_value = 0.035194  number = 28
k=0.0035  mean_value = 0.035136  number = 34
k=0.004  mean_value = 0.035077  number = 37
k=0.0045  mean_value = 0.035046  number = 38
k=0.005  mean_value = 0.035025  number = 44
k=0.0055  mean_value = 0.034977  number = 49
k=0.006  mean_value = 0.034929  number = 52
k=0.0065  mean_value = 0.034886  number = 53
k=0.007  mean_value = 0.034843  number = 54
k=0.0075  mean_value = 0.034802  number = 57
k=0.008  mean_value = 0.034755  number = 58
k=0.0085  mean_value = 0.034705  number = 59
k=0.009  mean_value = 0.034671  number = 61
k=0.0095  mean_value = 0.034649  number = 65
k=0.01  mean_value = 0.034627  number = 67
k=0.0105  mean_value = 0.034610  number = 68
k=0.011  mean_value = 0.034602  number = 69
k=0.0115  mean_value = 0.034601  number = 70
k=0.012  mean_value = 0.034601  number = 70
k=0.0125  mean_value = 0.034601  number = 70
k=0.013  mean_value = 0.034601  number = 70
k=0.0135  mean_value = 0.034607  number = 71
k=0.014  mean_value = 0.034614  number = 71
k=0.0145  mean_value = 0.034624  number = 72
k=0.015  mean_value = 0.034638  number = 72
k=0.0155  mean_value = 0.034658  number = 73
k=0.016  mean_value = 0.034678  number = 73
k=0.0165  mean_value = 0.034699  number = 73
k=0.017  mean_value = 0.034719  number = 73
k=0.0175  mean_value = 0.034740  number = 73
```

```
k=0.018  mean_value = 0.034760  number = 73
k=0.0185  mean_value = 0.034781  number = 73
k=0.019  mean_value = 0.034801  number = 73
k=0.0195  mean_value = 0.034822  number = 73
k=0.0205  mean_value = 0.034867  number = 73
k=0.021  mean_value = 0.034894  number = 74
k=0.0215  mean_value = 0.034921  number = 74
k=0.022  mean_value = 0.034948  number = 74
k=0.0225  mean_value = 0.034975  number = 74
k=0.023  mean_value = 0.035002  number = 74
k=0.0235  mean_value = 0.035029  number = 74
k=0.024  mean_value = 0.035056  number = 74
k=0.0245  mean_value = 0.035083  number = 74
k=0.025  mean_value = 0.035110  number = 74
k=0.0255  mean_value = 0.035138  number = 74
k=0.026  mean_value = 0.035165  number = 74
k=0.0265  mean_value = 0.035191  number = 74
k=0.027  mean_value = 0.035218  number = 74
k=0.0275  mean_value = 0.035246  number = 74
k=0.028  mean_value = 0.035274  number = 74
k=0.0285  mean_value = 0.035308  number = 75
k=0.029  mean_value = 0.035341  number = 75
k=0.0295  mean_value = 0.035374  number = 75
k=0.03  mean_value = 0.035408  number = 75
k=0.0305  mean_value = 0.035441  number = 75
k=0.031  mean_value = 0.035474  number = 75
k=0.0315  mean_value = 0.035508  number = 75
k=0.0320  mean_value = 0.035541  number = 75
k=0.0325  mean_value = 0.035574  number = 75
k=0.0330  mean_value = 0.035608  number = 75
k=0.0335  mean_value = 0.035641  number = 75
k=0.0340  mean_value = 0.035674  number = 75
k=0.0345  mean_value = 0.035708  number = 75
k=0.0350  mean_value = 0.035741  number = 75
k=0.0355  mean_value = 0.035774  number = 75
k=0.0360  mean_value = 0.035808  number = 75
k=0.0365  mean_value = 0.035841  number = 75
k=0.0370  mean_value = 0.035874  number = 75
k=0.0375  mean_value = 0.035908  number = 75
k=0.0380  mean_value = 0.035941  number = 75
k=0.0385  mean_value = 0.035974  number = 75
k=0.0390  mean_value = 0.036008  number = 75
k=0.0395  mean_value = 0.036041  number = 75
k=0.0400  mean_value = 0.036074  number = 75
k=0.0405  mean_value = 0.036108  number = 75
k=0.0410  mean_value = 0.036141  number = 75
k=0.0415  mean_value = 0.036174  number = 75
k=0.0420  mean_value = 0.036208  number = 75
k=0.0425  mean_value = 0.036241  number = 75
k=0.0430  mean_value = 0.036274  number = 75
k=0.0435  mean_value = 0.036308  number = 75
k=0.0440  mean_value = 0.036341  number = 75
k=0.0445  mean_value = 0.036441  number = 75
k=0.0450  mean_value = 0.036408  number = 75
k=0.0455  mean_value = 0.036441  number = 75
k=0.0460  mean_value = 0.036474  number = 75
k=0.0465  mean_value = 0.036508  number = 75
k=0.0470  mean_value = 0.036541  number = 75
k=0.0475  mean_value = 0.036574  number = 75
k=0.0480  mean_value = 0.036608  number = 75
k=0.0485  mean_value = 0.036641  number = 75
k=0.0490  mean_value = 0.036674  number = 75
k=0.0495  mean_value = 0.036708  number = 75
k=0.0500  mean_value = 0.036741  number = 75
```

BIBLIOGRAPHY

[1] J.W. Bandler and S.H. Chen, "Circuit optimization: the state of the art," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-36, 1988, pp. 424-443.

[2] G.C. Temes and D.A. Calahan, "Computer-aided network optimization the state-of-the-art," *Proc. IEEE*, vol. 55, 1967, pp. 1832-1863.

[3] J.W. Bandler and H.L. Abdel-Malek, "Optimal centering, tolerancing, and yield determination via updated approximations and cuts," *IEEE Trans. Circuits Syst.*, vol. CAS-25, 1978, pp. 853-871.

[4] H.L. Abdel-Malek and J.W. Bandler, "Yield optimization for arbitrary statistical distributions, Part I: Theory," *IEEE Trans. Circuits Syst.*, vol. CAS-27, 1980, pp. 245-253.

[5] H.L. Abdel-Malek and J.W. Bandler, "Yield optimization for arbitrary statistical distributions, Part II: Implementation," *IEEE Trans. Circuits Syst.*, vol. CAS-27, 1980, pp. 253-262.

[6] J.W. Bandler, M.A. El-Kady, W. Kellermann and W.M. Zuberek, "An optimization approach to the best alignment of manufactured and operating systems," in *Proc. IEEE Int. Symp. Circuits Syst.* (Newport Beach, CA), 1983, pp. 542-545.

[7] H. Tromp, "The generalized tolerance problem and worst case search," in *Proc. Conf. Computer-Aided Design of Electronic and Microwave Circuits Syst.* (Hull, England), 1977, pp. 72-77.

[8] H. Tromp, "Generalized worst case design, with applications to microwave networks," Doctoral thesis (in Dutch), Faculty of Engineering, University of Ghent, Ghent, Belgium, 1978.

[9] D.D. Morrison, "Optimization by least squares," *SIAM J. Numer. Anal.*, vol. 5, 1968, pp. 83-88.

[10] J.W. Bandler, S.H. Chen and S. Daijavad, "Microwave device modeling using efficient ℓ_1 optimization: a novel approach," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-34, 1986, pp. 1282-1293.

[11] J.W. Bandler, W. Kellermann and K. Madsen, "A nonlinear ℓ_1 optimization algorithm for design, modeling and diagnosis of networks," *IEEE Trans. Circuits Syst.*, vol. CAS-34, 1987, pp. 174-181.

[12] K. Madsen, H. Schjaer-Jacobsen, and J. Voldby, "Automated minimax design of networks," *IEEE Trans. Circuits Syst.*, vol. CAS-22, 1975, pp. 791-796.

[13] J.W. Bandler, S.H. Chen, R.M. Biernacki, K. Madsen, L. Gao* and H. Yu, "Robustizing circuit optimization using Huber functions," in *IEEE Int. Microwave Symp. Dig.* (Atlanta, GA), 1993, pp. 1009-1012.

[14] P.J. Huber, "Robust estimation of a location parameter," *Ann. Math. Statist.*, vol. 35, 1964, pp. 73-101.

[15] H. Ekblom and K. Madsen, "Algorithms for non-linear Huber estimation," *BIT*, vol. 29, 1989, pp. 60-76.

[16] G. Li* and K. Madsen, "Robust nonlinear data fitting," in *Numerical Analysis 1987*, D.F. Griffiths and G.A. Watson, Eds., Pitman Research Notes in Mathematics Series 170. Harlow, Essex, UK: Longman, 1988, pp. 176-191.

[17] P. Huber, *Robust Statistics*. New York: Wiley, 1981.

[18] J.W. Bandler, S.H. Chen, R.M. Biernacki, L. Gao*, K. Madsen and H. Yu, "Huber optimization of circuits: a robust approach," *IEEE Trans. Microwave Theory Tech.*, vol. 41, 1993.

[19] *OSA90/hope*™, Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, 1993.

[20] *HarPE*™, Optimization Systems Associates Inc., P.O. Box 8083, Dundas, Ontario, Canada L9H 5E7, 1993.

[21] R. Dutter and P.J. Huber, "Numerical methods for the nonlinear robust regression problem," *J. Stat. Comp. Sim.*, vol. 13, 1981, pp. 79-113.

[22] D.F. Shanno and D.M. Rocke, "Numerical methods for robust regression: linear models," *SIAM J. Sci. Stat. Comput.*, vol. 7, 1986, pp. 86-97.

[23] A.E. Beaton and J.W. Tukey, "The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data," *Technometrics,* vol. 16, 1974, pp. 147-185.

[24] J.W. Bandler, "Computer-aided circuit optimization," in *Modern Filter Theory and Design,* G.C. Temes and S.K. Mitra, Eds. New York: Wiley-Interscience, 1973, pp. 211-271.

[25] J.W. Bandler and M.R.M. Rizk, "Optimization of electrical circuits," *Mathematical Programming Study,* vol. 11, 1979, pp. 1-64.

[26] G.C. Temes and D.Y.F. Zai, "Least *p*th approximation," *IEEE Trans. Circuit Theory,* vol. CT-16, 1969, pp.235-237.

[27] B.D. Bunday, *Basic Optimisation Methods.* London: Edward Arnold, 1984.

[28] K. Madsen, "Minimization of non-linear approximation functions," Doctoral Thesis, Institute for Numerical Analysis, Technical University of Denmark, Lyngby, Denmark, 1985.

[29] J.J. Moré, "Recent developments in algorithms and software for trust region methods," in *Mathematical Programming, The State of the Art (Bonn 1982).* Bonn, West Germany: Springer Verlag, 1982, pp. 258-287.

[30] D.G. Luenberger, *Linear and Nonlinear Programming.* Reading, MA: Addison-Wesley, 1984.

[31] K. Madsen and L. Gao*, "A one-sided Huber approach for design problems," private communications, 1992.

[32] J.W. Bandler, R.M. Biernacki, Q. Cai, S.H. Chen, S. Ye, Q.-J. Zhang, "Integrated physics-oriented statistical modeling, simulation and optimization," *IEEE Trans. Microwave Theory Tech.,* vol. MTT-40, 1992, pp. 1374-1400.

[33] J. Purviance, D. Criss and D. Monteith, "FET model statistics and their effects on design centering and yield prediction for microwave amplifiers," in *IEEE Int. Microwave Symp. Dig.* (New York), 1988, pp. 315-318.

[34] Q. Cai, "Physics-based microwave device modeling and circuit optimization," Ph.D. Thesis, McMaster University, 1992.

[35] R.W. Dutton, D.A. Divekar, A.G. Gonzalez, S.E. Hansen and D.A. Antoniadis, "Correlation of fabrication process and electrical device parameter variations," *IEEE J. Solid-State Circuits,* vol. SC-12, 1977, pp. 349-355.

[36] D.A. Divekar, R.W. Dutton and W.J. McCalla, "Experimental study of Gummel-Poon model parameter correlations for bipolar junction transistors," *IEEE J. Solid-State Circuits,* vol. SC-12, 1977, pp. 552-559.

[37] P.J. Rankin, "Statistical modeling for integrated circuits," *IEE Proc.,* vol. 129, Pt. G, No. 4, 1982, pp. 186-191.

[38] N. Herr and J.J. Barnes, "Statistical circuit simulation modeling of CMOS VLSI," *IEEE Trans. Computer-Aided Design,* vol. CAD-5, 1986, pp. 15-22.

[39] C.J.B. Spanos and S.W. Director, "Parameter extraction for statistical IC process characterization," *IEEE Trans. Computer-Aided Design,* vol. CAD-5, 1986, pp. 66-78.

[40] J.W. Bandler, R.M. Biernacki, S.H. Chen, J. Loman, M. Renault and Q.J. Zhang, "Combined discrete/normal statistical modeling of microwave devices," in *Proc. European Microwave Conf.* (Wembley, England), 1989, pp. 205-210.

[41] J.W. Bandler, R.M. Biernacki, S.H. Chen, J. Song, S. Ye and Q.J. Zhang, "Statistical modeling of GaAs MESFETs," in *IEEE Int. Microwave Symp. Dig.* (Boston, MA), 1991, pp. 87-90.

[42] P.H. Ladbrooke, *MMIC Design: GaAs FETs and HEMTs.* Norwood, MA: Artech House, 1989.

[43] A. Materka and T. Kacprzak, "Computer calculation of large-signal GaAs FET amplifier characteristics," *IEEE Trans. Microwave Theory Tech.,* vol. MTT-33, 1985, pp. 129-135.

[44] M.A. Khatibzadeh and R.J. Trew, "A large-signal, analytic model for the GaAs MESFET," *IEEE Trans. Microwave Theory Tech.,* vol. MTT-36, 1988, pp. 231-238.

[45] G. Blom, *Probability and Statistics: Theory and Applications.* New York: Springer-Verlag, 1989.

[46] W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes: The Art of Scientific Computing (FORTRAN Version).* Cambridge University Press, Cambridge, England, 1989.

[47] Measurement data provided by Plessey Research Caswell Ltd., Caswell, Towcester, Northamptonshire, England NN12 8EQ, 1990.

[48] R.S. Berkowitz, "Conditions for network-element-value solvability," *IRE Trans. Circuit Theory,* vol. CT-9, 1962, pp. 24-29.

[49] S.D. Bedrosian and R.S. Berkowitz, "Solution procedure for single element-kind networks," *IRE Int. Conv. Rec.*, pt. 2, 1962, pp. 16-24.

[50] P.M. Lin and Y.S. Elcherif, "Computational approaches to fault dictionary," in *Analog Methods for Computer-aided Circuit Analysis and Diagnosis,* T. Ozawa, Ed., New York: Marcel Dekker, 1988, pp. 325-364.

[51] J.W. Bandler and A.E. Salama, "Fault diagnosis of analog circuits," *Proc. IEEE,* vol. 73, 1985, pp. 1279-1325.

[52] J.W. Bandler and Q.-J. Zhang, "Optimization techniques for modeling, diagnosis, and tuning," in *Analog Methods for Computer-aided Circuit Analysis and Diagnosis,* T. Ozawa, Ed., New York: Marcel Dekker, 1988, pp. 381-416.

[53] H.M. Merrill, "Failure diagnosis using quadratic programming," *IEEE Trans. Reliability,* vol. R-22, 1973, pp. 207-213.

[54] R. DeCarlo and R. Saeks, *Interconnected Dynamical Systems.* New York: Marcel Dekker, 1981.

[55] M.N. Ransom and R. Saeks, "The connection function — theory and application," *Int. J. Circuit Theory Appl.*, vol. 3, 1975, pp. 5-21.

[56] M.N. Ransom and R. Saeks, "Fault isolation with insufficient measurements," *IEEE Trans. Circuit Theory,* vol. CT-20, 1973, pp. 416-417.

[57] J.W. Bandler, R.M. Biernacki, A.E. Salama and J.A. Starzyk, "Fault isolation in linear analog circuits using the ℓ_1 norm," in *Proc. IEEE Int. Symp. Circuits Syst.* (Rome, Italy), 1982, pp. 1140-1143.

[58] J.W. Bandler and Q.-J. Zhang, "An automatic decomposition approach to optimization of large microwave systems," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-35, 1987, pp. 1231-1239.

[59] D.M. Himmelblau, Ed., *Decomposition of Large-Scale Problems.* Amsterdan: North-Holland, 1973.

[60] A. Sangiovanni-Vincentelli, L.K. Chen and L.O. Chua, "An efficient heuristic cluster algorithm for tearing large-scale networks," *IEEE Trans. Circuits Syst.*, vol. CAS-24, 1977, pp. 709-717.

[61] A.E. Atia, "Computer-aided design of waveguide multiplexers," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-22, 1974, pp. 332-336.

[62] C.M. Kudsia, "Manifestations and limits of dual-mode filter configurations for communications satellite multiplexers," in *AIAA 9th Communications Satellite Systems Conference* (San Diego, CA), 1982, pp. 294-303.

[63] E.G. Cristal and G.L. Matthaei, "A technique for the design of multiplexers having contiguous channels," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-12, 1964, pp. 88-93.

[64] J.W. Bandler, S. Daijavad and Q.-J. Zhang, "Exact simulation and sensitivity analysis of multiplexing networks," *IEEE Trans. Microwave Theory Tech.*, vol. MTT-34, 1986, pp. 93-102.

[65] J.W. Bandler, S. Daijavad and Q.-J. Zhang, "Theory of computer-aided design of microwave multiplexers," Simulation Optimization Systems Research Laboratory, McMaster University, Hamilton, Canada, Report SOS-84-8-R, 1984.

[66] R.M. Biernacki, J.W. Bandler, J. Song and Q.J. Zhang, "Efficient quadratic approximation for statistical design," *IEEE Trans. Circuits Syst.*, vol. 36, 1989, pp. 1449-1454.

[67] W. Kellermann, "Advances in optimization of circuits and systems using recent minimax and ℓ_1 algorithms," Ph.D. Thesis, McMaster University, 1986.

*The names Li Gao and Gao Li in this thesis represent the same person, whose actual name should be Li Gao.

SUBJECT INDEX