POLITECNICO DI TORINO

Master of Science in Electronic Engineering

Master Degree Thesis

Greedy Algorithms for Black-Box Parameterized Modeling of Electromagnetic Structures

Supervisor:
prof. Stefano Grivet-Talocia

Co-Supervisor:
prof. Piergiorgio L.E. Uslenghi

Candidate
Elisa Fevola

Academic year 2017 – 2018

Summary

This work provides a software tool for the generation of parameterized macromodels of electromagnetic structures. The novelty of this work is the implementation of a class of greedy and adaptive algorithms for the optimal choice of those points in the parameter space that are used to construct and self-validate the model, and which are provided by a commercial EM solver. The developed tool provides a seamless integration with the adopted field solver, allowing a fully-automated model generation.

Acknowledgements

I would like to express my deepest gratitude to my advisor, Professor Stefano Grivet-Talocia, for his extensive and invaluable guidance throughout this work. His passion and his patience have been a source of inspiration through these years.
This thesis has been partially developed at the University of Illinois at Chicago, thus I am extremely thankful to Professor Piergiorgio Uslenghi for giving me the opportunity to undertake this research work at UIC, and for his continuous support during my experience in the US.
Finally, I would like to thank my parents and my sister for their unconditional support and love.
A special thank to Antonio: even when you were far away, you were always there.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The use of parameterized macromodels has become more and more popular in a wide range of applications, from electromagnetic compatibility to microwave engineering, where they can be employed for the analysis, design and optimization of microwave structures. It is quite common, in fact, that for design optimization purposes or sensitivity analysis some geometric properties and physical quantities are left undefined and become variables of the system.
A macromodel can be defined as an approximate reduced-order behavioral model, which provides in a compact form a state-space representation of the object response. The success of macromodels is due mainly to the fact that they are easy and fast to simulate both in time and in frequency domain, still guaranteeing a high level of accuracy.
A typical workflow for macromodel generation is to start from scattering responses obtained from commercial full-wave solvers, which are then post-processed to fit or identify a compact model. This implies that, in order to simulate all configurations that are necessary to build the model, an extremely large amount of full-wave analyses must be performed. With the increase in the number of dimensions, the quantity of data samples increases exponentially, making the construction of the macromodel too expensive in terms of computational time and resources.
One of the possible solutions to overcome this problem is the use of adaptive sampling algorithms targeting the identification of a quasi-minimal distribution of data samples sufficient to characterize the system, and thus to build a reasonably accurate model. This thesis presents a class of sampling algorithms for a fully-automated generation of parameterized macromodels, based on a given electromagnetic solver. The presented framework not only implements a combination of greedy and adaptive algorithms for the optimal choice of points in the parameter space, but it also connects one of the most diffused commercial EM solvers (Keysight EMPro) to a MATLAB tool for the extraction of parametric macromodels in a fully automated way.
The developed framework implements an iterative adaptive algorithm which selects, at each step, the optimal points in the parameter space for the generation of the model. The choice of such points is based both on the analysis of data already available in the parameters space from previous iterations (data-driven technique) and on the accuracy of intermediate macromodels (model-driven technique). More specifically, the optimality of a new point More specifically, the optimality of a new point is established by a joint analysis

of the distribution of points already present in the parameters space, of the variability of data in such space and on the accuracy of the model created during the previous iteration. Due to the model-driven approach used during this selection, an intermediate parameterized macromodel is built at each iteration. The choice, among all available data, of those which will be used to fit the model, and those left for validation is made by a Vectorial Kernel Orthogonal Greedy Algorithm (VKOGA). VKOGA operates a sparse approximation of nonlinear vectorial functions by simultaneously approximating all vector components in a greedy way. The iterative nature of the algorithm allows to obtain a compact, robust model with a minimum number of points, and so of solver simulations, reducing drastically computational time. The tool developed for this thesis, moreover, provides a seamless integration between field solvers and the macromodeling engine, thus further increasing time efficiency and offering to the user the possibility to interact exclusively with such tool, being the interaction among field solver and model generator completely automated.
The algorithm has been trained and tested on a number of electromagnetic structures, namely a distributed filter, an integrated low pass filter and a microstrip patch antenna, and its effectiveness has been demonstrated in all the aforementioned cases.

1.1 Outline

The thesis is organized as follows: in **Chapter 2** a general overview on macromodeling theory is given, with a particular emphasis on parameterized macromodels, which are relevant to this work.
Chapter 3 is devoted to a detailed description of the adaptive algorithm implemented for the selection of optimal points in the design space. A brief reference to examples is given, which however is the main topic of **Chapter 4**, fully focused on the experimental results of the implemented tool. Here, all the examples and practical cases in which the framework has been tested are described and results are provided and analysed, as a proof of the effectiveness of the presented class of algorithms. In **Chapter 5**, instead, a summary on available commercial EM solvers is provided, together with an analysis on their working principles, their target applications and similarities/differences among them. The rest of the Chapter is then devoted to a more technical description of the integration process between the MATLAB tool and the EM solver, followed by conclusions and future work ideas in **Chapter 6**.

Chapter 2

Passive Parameterized Macromodeling

2.1 What is a macromodel?

The need of macromodeling arises from the impossibility to apply direct approaches on a whole electrical system: in order to perform numerical modeling it would be necessary to analyse an entire system (which is often extremely complex and detailed, like a PCB) characterized by first-principle models (like Maxwell's equations) and simulate it on a computer. The resources needed for this type of direct simulation in terms of runtime and memory requirements are simply prohibitive, thus making it extremely difficult for engineers to model and simulate directly a complex system.
A possible solution to overcome this difficulty is to divide the system into sub-parts and characterize them independently by means of "macromodels". Macromodels are approximate behavioral models obtained from the terminal responses of the structure, which allow to obtain a state-space representation of the behavior of the object [1].
Since sub-blocks represent specific parts or devices of the whole structure, they can be properly connected to perform system-level analysis, but this time with reasonable processing time.

2.2 Black-box macromodeling

The most diffused type of macromodeling, and the one which is used in this thesis, is called **black-box modeling**: the name refers to the similarities between the physical object under analysis and the final macromodel, and it can be clarified by describing how black-box macromodels are obtained in the first place. The behaviour of the structure is analysed by identifying some terminals and retrieving the corresponding input and output responses of the object. These responses will be used to build the actual macromodel, whose responses will approximate the true responses with a certain level of accuracy. This means that, at the end of the macromodeling process, there will be no direct correspondence between the model and the original system in terms of topology: the model will reproduce the behaviour of the object, not its physical structure [1].

The most common situations in which black-box macromodeling comes into play are basically two:

- the system under analysis is available as a physical object (an antenna, for example). In this case real measurements can be performed on it, either in time domain or in frequency domain, in order to collect data upon which a model can be built;
- the object is available as a CAD model (but not as a real hardware). This means that the measurements on it will be performed by means of commercial field solvers (like Keysight EMPro, the EM solver employed in this thesis), which usually provide only the results of the simulations and not the intermediate steps of the computation (this is mainly due to "protect" the algorithm from unwanted disclosure). Thus, at the end of the simulation the object is already a black-box when it comes to its behaviour.

This second situation is the case for this work, where the models are built upon the responses of commercial EM solvers.
The whole discipline of macromodeling deals with the numerical techniques used to identify the model of a system, starting only from its input-output characteristics. It is important to clarify that the formulation of a sufficiently accurate model is the first aim of the procedure: there is no interest in discovering the internal physical structure of the black-box, this is not what macromodeling is for. Once the model is created, the object will be represented by a system of equations that approximate its characteristics, with no link to its geometry.

2.3 Rational curve fitting

The core step in the formulation of behavioral macromodels consists in fitting the model to a set of tabulated input-output responses. In order to do this, a model structure which will represent the object under analysis must be chosen in the first place. Generally, since we deal with *Linear Time-Invariant* (LTI) systems, the natural choice is to create a model which will mimic their transfer function $H(s)$. Again, a natural choice in this case is to represent $H(s)$ as a rational function with the complex frequency s as the independent variable.
A rational function $H(s)$ corresponds, by definition, to an algebraic fraction of two polynomials as numerator and denominator:

$$H(s) = \frac{N(s)}{D(s)} = \frac{b_0 + b_1 s + b_2 s^2 + \cdots + b_m s^m}{1 + a_1 s + a_2 s^2 + \cdots + a_n s^n}. \tag{2.1}$$

where m is the numerator degree and n is the denominator degree. This general expression can be rewritten in equivalent forms, such as the **pole-zero form**:

$$H(s) = \alpha \frac{(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - v_1)(s - v_2) \cdots (s - v_n)} \tag{2.2}$$

where v_i are the poles and z_i are the zeros, and as the **pole-residue form**:

$$H(s) = r_0 + \frac{r_1}{s - v_1} + \frac{r_2}{s - v_2} + \cdots + \frac{r_n}{s - v_n} \tag{2.3}$$

where r_i are the residues, taking $m = n$.

Each of these representations can be a starting point for the creation of a macromodel, but generally the most employed is the pole-residue form of 2.3, mainly because algorithms computing poles and residues turn out to be more robust [1].
Once the model structure has been defined, the following steps consist in determining the model parameters by fitting the model characteristic to the input-output responses collected from direct measurements or solvers' simulations. If we define the available responses as $\widetilde{H_t}$ and the frequency at which they are calculated as $s_t = \mathrm{j}\omega_t$, we can restate the problem as

$$\widetilde{H_t} \approx H(\mathrm{j}w_t) = r_0 + \frac{r_1}{\mathrm{j}\omega_t - v_1} + \frac{r_2}{\mathrm{j}\omega_t - v_2} + \cdots + \frac{r_n}{\mathrm{j}\omega_t - v_n} \tag{2.4}$$

This corresponds to a curve fitting problem, where the coefficients r_i and v_i must satisfy 2.4 for all frequencies s_t.
All the three representations of the rational function reported above lead to cost functions which are **non-linear** and **non-convex** in the decision variables, thus potentially presenting many local minima. The problem of the minimization of the cost function then becomes very challenging, since a local optimization scheme like the Newton's method is not guaranteed to converge. The need for new rational curve fitting schemes comes from the non-linearity and non-convexity of the cost function, leading to new and more robust numerical methods.
Many methods have been formulated and are available, but one, the **Vector Fitting (VF)** algorithm, will be analysed in this thesis, as it is the method employed for the creation of the parametric macromodels we deal with.

2.4 Vector Fitting

The Vector Fitting method, as anticipated before, is an algorithm for fitting a model, expressed in the form of a rational function, to input-output responses, expressed either in frequency domain or in time domain. It has been introduced by [2], and an open-source MATLAB code implementing it is available online [3].
VF is probably the most employed fitting method, at least for electrical and electronic applications, thanks to some features that make it simple and versatile to use, but at the same time sufficiently accurate and efficient.
Before diving into the details of Vector Fitting algorithm, it is helpful to introduce some precursors of the VF scheme, the so-called Sanathanan-Koerner (SK) and Generalized Sanathanan-Koerner (GSK) iteration. This step is important because these two formulations introduce some key aspects that solve the two crucial problems characterizing other fitting algorithms, in particular a bias in the fit and the ill-conditioning of the model.

2.4.1 The Sanathanan-Koerner iteration

Sanathanan Koerner iteration, presented in [4], has the merit of having solved a major concern of the linearization approach via weighting, that is the introduction of a bias in the fit due to the frequency dependence of the weight [1].

The rational form has always the form reported in 2.1. If we group all the coefficients into a vector of the form

$$x = (b_0, b_1, ..., b_m, a_1 ... a_n)^T \tag{2.5}$$

the model takes the form

$$H(s;x) = \frac{N(s;x)}{D(s;x)} \tag{2.6}$$

Since the final goal is to obtain a model which approximates the original responses with the highest accuracy achievable, it is useful to define a residual vector as

$$r_t(x) = \widetilde{H_t} - \frac{N(s_t;x)}{D(s_t;x)} \tag{2.7}$$

which can be rewritten as

$$r_t(x) = \frac{D(s_t;x)\widetilde{H_t} - N(s_t;x)}{D(s_t;x)} \tag{2.8}$$

The solution is obtained by iteration, so the residual can be expressed in terms of the iteration index ν

$$r_t^{\nu}(x_\nu) = \frac{D(s_t;x_\nu)\widetilde{H_t} - N(s_t;x_\nu)}{D(s_t;x_{\nu-1})} \tag{2.9}$$

What must be noticed in 2.9 is the denominator, which corresponds to the denominator D calculated in the previous iteration $\nu - 1$. Due to the fact that this term is already available when the iteration ν is performed, the problem reduces to a linear least squares problem.

2.4.2 The Generalized Sanathanan-Koerner iteration

The second problem highlighted before was the ill-conditioning of the model, due to the use of polynomials as numerator and denominator in the rational function $H(s;x)$.
The solution introduced by the GSK iteration is the use of more general basis functions in place of the previous monomial ones, which can be expressed as follows:

$$H(s;x) = \frac{N(s;x)}{D(s;x)} = \frac{\sum_{j=0}^{m} c_j \varphi_j(s)}{\sum_{j=0}^{n} d_j \varphi_j(s)} \tag{2.10}$$

where

$$\boldsymbol{x} = (c_0, ..., c_n, d_0, ...d_n)^{\intercal}. \tag{2.11}$$

The basis functions $\varphi(s)$ represent a general *rational* basis function, and this is perfectly acceptable since $N(s;x)$ and $D(s;x)$ become linear combinations of rational functions, thus rational functions themselves, and consequently $H(s;x)$, being their ratio, is a rational function as well.

A first natural choice for $\varphi_j(s)$ is the **partial fraction basis**, expressed as

$$\varphi_0(s) = 1, \quad \text{and} \quad \varphi_j(s) = \frac{1}{s - q_j}, \quad j = 1, ..., n. \tag{2.12}$$

These basis functions are linearly independent, provided that the associated poles q_j are different one from the other.

2.4.3 From GSK to Vector Fitting

Using 2.12 the model can be represented as

$$H(s;x) = \frac{N(s;x)}{D(s;x)} = \frac{c_0 + \sum_{j=1}^{n} \frac{c_j}{s-q_j}}{1 + \sum_{j=1}^{n} \frac{d_j}{s-q_j}} \tag{2.13}$$

where we set $d_0 = 1$ to guarantee uniqueness in the model representation. This can be rewritten as

$$H(s;x) = c_0 \frac{\prod_{j=1}^{n} \frac{s - z_j}{s - q_j}}{\prod_{j=1}^{n} \frac{s - v_j}{s - q_j}} = c_0 \frac{\prod_{j=1}^{n} (s - z_j)}{\prod_{j=1}^{n} (s - v_j)} \tag{2.14}$$

From this passage it is clear why the Vector Fitting is defined as a *pole relocation* process: if we start from a set of poles $\{q_j\}$ we end up with a set of estimated poles $\{v_j\}$ for the model, retrieved through a refinement process [5].

2.4.4 The Vector Fitting scheme

In order to present the VF algorithm, it is necessary to start from a group of initial arbitrary poles, the starting poles. From this set $\{q_j^\nu, j = 1, ..., n\}$ we can define the basis

$$\varphi_0^\nu(s) = 1, \quad \text{and} \quad \varphi_j^\nu(s) = \frac{1}{s - q_j^\nu}, \quad j = 1, ..., n. \tag{2.15}$$

and the *weighting function*

$$\xi^\nu(s) = \varphi_0^\nu(s) + \sum_{j=1}^{n} d_j^\nu \varphi_j^\nu(s) = 1 + \sum_{j=1}^{n} \frac{d_j^\nu}{s - q_j^\nu} \tag{2.16}$$

This expression can be used to approximate $\widetilde{H}_t$ in this way:

$$\xi^\nu(s_t)\widetilde{H}_t = \left(1 + \sum_{j=1}^{n} \frac{d_j^\nu}{s_t - q_j^\nu}\right) \widetilde{H}_t \approx c_0^\nu + \sum_{j=1}^{n} \frac{c_j^\nu}{s_t - q_j^\nu} \tag{2.17}$$

by solving the corresponding linear least squares problem to determine the coefficient c_j^ν and d_j^ν. The weighting function can be transformed into its zero-pole form

$$\xi^\nu(s) = 1 + \sum_{j=1}^{n} \frac{d_j^\nu}{s - q_j^\nu} = \frac{\prod_{j=1}^{n} (s - z_j^\nu)}{\prod_{j=1}^{n} (s - q_j^\nu)} \tag{2.18}$$

The zeros z_j^ν will be used to calculate the set of starting poles $q_j^{\nu+1}$ for the following iteration. This pole relocation process is repeated until convergence, which coincides with the weight function $\xi^\nu(s)$ approaching 1 for all frequencies.

2.5 Parameterized Macromodeling

What has been discussed up to now does not contemplate the presence of a parameter: the macromodel is always expressed as a function of just the complex frequency s, which is the only independent variable of the system.
However, there are many practical cases in which the dependence upon one or more parameters must be included in the model. It can happen, for example, that the macromodeling technique is used to represent an object under design: in this case, it is possible that some physical parameters are left undefined, and their values will be chosen such that the overall object performance is optimized.
Another practical case is the modeling of an object illuminated by an external electromagnetic field (a recurrent situation in EMC/EMI problems). If the behavior of the system in presence of an external excitation must be modeled, the impinging direction of the wave can be left as a parameter, so that the model can predict the system response for any possible direction.
The presence of parameters in the formulation of the problem makes the subject shift from the so called *univariate macromodeling* (with no parameter dependence) to *multivariate macromodeling*, better known as *parameterized macromodeling*, as explained in [1].
Examples illustrated above give an idea of the wide range of applications in which parameterized macromodels can be employed, from optimization problems to EMI/EMC analyses.
In the remaining part of the chapter, the main results presented before will be reformulated for the multivariate case.
First of all, a parameterized macromodel can be defined as $\mathbf{H}(s;\theta)$, implying that the model response depends both on frequency s and on a parameter θ. Obviously, the parameters can be more than one, so that the macromodel representation becomes $\mathbf{H}(s;\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ represents a vector of parameters, but here for simplicity the case with one parameter will be analysed.
Construction of the model starts from the evaluation of the true response for fixed frequencies and fixed parameter values. True responses, in fact, are usually obtained from real life measurements, or as outputs of EM solvers. In both cases, if we define a frequency range $[\omega_{min}, \omega_{max}]$ and a parameter range $[\theta_{min}, \theta_{max}]$, the available responses will be of the form

$$\widetilde{\mathbf{H}}_{t,\mu} = \widetilde{\mathbf{H}}(j\omega_t;\theta_\mu) \quad t = 1, ..., T, \mu = 1, ..., M. \tag{2.19}$$

meaning that they refer to fixed sampled frequencies $\omega_t, t = 1, ..., T$, with $\omega_1 = \omega_{min}$ and $\omega_T = \omega_{max}$, and to fixed parameter values $\theta_m, m = 1, ..., M$, with $\theta_1 = \theta_{min}$ and $\theta_M = \theta_{max}$.
The model will be constructed starting from these data, by enforcing the fitting condition:

$$\mathbf{H}(j\omega_t;\theta_\mu) \approx \widetilde{\mathbf{H}}_{t;\mu} \quad t = 1, ..., T, \mu = 1, ..., M. \tag{2.20}$$

implying that the obtained model will approximate the true responses over the defined parameter and frequency range.

As illustrated in [1], there are many formulations for parameterized macromodels: the simplest one is the case with fixed poles, where it is assumed that the parameter dependency does not affect the position of the poles. In other words, only the residues depend on the parameter, while the poles remain always the same. This case is obviously really specific, and it cannot be applied in the majority of situations where a parameterization of the model is necessary and where, for instance, the system resonances that are represented by the poles depend on the parameter(s).
A more general form is the fully parameterized model, where a parameter dependency of the poles is taken into account. It must be underlined, however, that the parameterization of the poles $v(\theta)$ and the zeros $z(\theta)$ needs to be implicit, since they generally vary non-smoothly with respect to the parameter, as a variation of the parameter θ could lead to bifurcations of zeros and poles [6].
This leads to an implicitly parameterized macromodel, which can be expressed as

$$\mathbf{H}(s;\theta) = \frac{N(s;\theta)}{D(s;\theta)} = \frac{\sum_{j=0}^{n}\sum_{\chi=1}^{X}\mathbf{A}_{j,\chi}\varphi_j(s)\xi_\chi(\theta)}{\sum_{j=0}^{n}\sum_{\chi=1}^{X}B_{j,\chi}\varphi_j(s)\xi_\chi(\theta)}, \tag{2.21}$$

The basis functions $\varphi_j(s)$ are frequency dependent and they are defined as partial fractions

$$\varphi_0(s) = 0, \quad \varphi_j(s) = \frac{1}{s - q_j}, j = 1, ..., n, \tag{2.22}$$

where $\xi_\chi(\theta)$, instead, denote a set of parameter dependent basis functions, the Fourier basis for example, as presented in [6], or the Chebychev polynomial up to a given maximum degree.
The fitting problem then reduces to the identification of the coefficients $\mathbf{A}_{j,\chi}$ and $B_{j,\chi}$. The parameterized form of the generalized Sanathanan-Koerner iteration (GSK), also called PSK, is used, which can be expressed in the form:

$$\frac{\sum_{j=0}^{n}\sum_{\chi=1}^{X}\left[d_{j,l}^{\nu}\widetilde{\mathbf{H}}_{t,\mu} - \mathbf{A}_{j,\chi}^{\nu}\right]\varphi_j(j\omega_t)\xi_\chi(\theta_\mu)}{\sum_{j=0}^{n}\sum_{\chi=1}^{X}B_{j,\chi}^{\nu-1}\varphi_j(j\omega_t)\xi_\chi(\theta_\mu)} \approx 0 \tag{2.23}$$

2.23 can be recognized as a linear least squares problem where the unknowns to be identified are $\mathbf{A}_{j,\chi}^{\nu}$ and $B_{j,\chi}^{\nu}$. The index ν suggests that the solution is obtained through an iterative procedure where $\nu = 1,2, ...$ until the solution converges.

2.6 A Parameterized macromodeling tool

A direct implementation of the concepts introduced before regarding macromodeling of parametric structures is provided by **IdEMPar**, a MATLAB tool developed at Politecnico di Torino [7]. The tool, based on [8] and [9], is an innovative CAD tool for the creation of parametric models of passive components and interconnects.
The tool generates models which are functions of complex frequency $j\omega$ and one or more design parameters, allowing for fast optimizations and what-if analyses. Models, in fact, are

created from a set of simulated scattering responses of the device under analysis, provided in the form of Touchstone files. Fitting of such responses allows for the creation of a model which interpolates the device response with respect to both frequency and design parameters. Moreover, the generated model can be exported as a SPICE netlist or as a set of Touchstone file, thus giving the possibility to use it in the most diffused EDA tools.

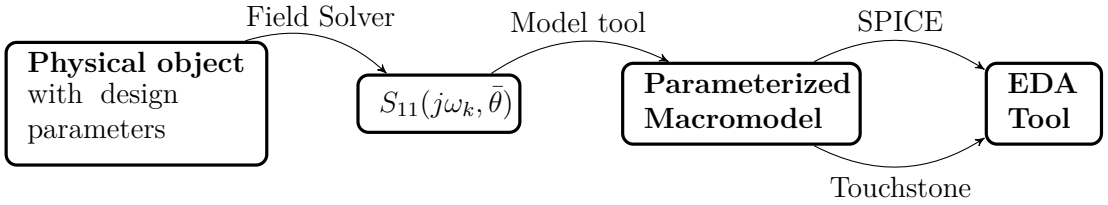


Figure 2.1: Block diagram of the workflow for the parameterized macromodeling tool

As already stated previously, the tool is based on the theory presented before, which led to the formulation of a parameterized model as:

$$S_{iq}(j\omega;\theta) = \frac{\sum_{\chi,n} A_{\chi n}\xi_{\chi}(\theta)\varphi_n(j\omega)}{\sum_{\chi,n} B_{\chi n}\xi_{\chi}(\theta)\varphi_n(j\omega)}, \quad i = 1,2,3..., q = 1,2,3... \tag{2.24}$$

where

- $\xi_{\chi}(\theta)$ is a set of basis functions defined in the parameter space;
- $\varphi_n(j\omega)$ are the partial fraction basis function defined by the automatically selected basis poles;
- $A_{\chi n}$, $B_{\chi n}$ are the unknown model coefficients which must be identified for model generation.

If more than one parameter is present, the model can be formulated as:

$$S_{iq}(j\omega,\theta_1,...,\theta_M) = \frac{\sum_{\chi_1,...,\chi_M,n} A_{\chi_1,...,\chi_M n}\xi_{\chi_1}(\theta_1)...\xi_{\chi_M}(\theta_M)\varphi_n(j\omega)}{\sum_{\chi_1,...,\chi_M,n} B_{\chi_1,...,\chi_M n}\xi_{\chi_1}(\theta_1)...\xi_{\chi_M}(\theta_M)\varphi_n(j\omega)} \tag{2.25}$$

Since the macromodel is a rational function of frequency, the synthesis of a SPICE netlist is straightforward, making this tool a powerful instrument for the creation of models employable in EDA processes.
As the purpose of this thesis is to develop a class of adaptive algorithms selecting data points for model generation, the proposed tool is not involved in the actual creation of the model. This duty is left to IdEMPar, which in this work has been employed as a black-box software.
Even if the code for the generation of parameterized macromodel has not been touched, some modifications on IdEMPar have been necessary, in order to integrate it with the developed framework.

Chapter 3

The main algorithm

As already expressed in the introduction, this research thesis mainly deals with the implementation of an adaptive algorithm for an optimal selection of data points necessary to build an accurate parameterized macromodel. The focus of this chapter is an extensive analysis of the implemented algorithm: after a brief summary on the context and background of this work, a detailed presentation of the algorithm will follow.

3.1 Why an adaptive algorithm?

Parameterized macromodels, as they have been presented in Chapter 2, have become more and more diffused in a wide variety of applications. The fact that they reproduce the behaviour of a system maintaining the dependency on one or more layout variables makes them particularly suitable during the design, optimization and analysis processes.
It should be clear by now that models are built by fitting a set of system responses, which can come directly from real-life measurements, or they can be the output of first-principle solvers. Such solvers, which will be analysed more in depth in Chapter 5, are generally commercial tools that solve Maxwell's equations at a system level by dividing the object under analysis and the surrounding space into small elements.
In order to build a parametric macromodel, ideally an EM solver simulation should be performed for each value that the parameter can assume. If more than one parameter is present, the number of necessary simulations increases exponentially. A common approach, in this case, is to divide the design space into a fine grid: in such a way, points are distributed uniformly and they cover homogeneously the entire space (3.1, plot on the left). This approach, however, becomes soon unfeasible: the large number of simulations that must be performed (especially if more than one parameter is present), combined with the fact that EM simulations tend to be time-consuming, makes the generation of a parameterized macromodel extremely expensive, both in terms of time and computational resources. In addition, the tool for model creation is a stand-alone software, and no type of integration is provided with commercial field solvers: this translates into a significant amount of additional time that the user must spend to drive manually the solver, repeat the required simulations, collect data and use them to feed the modeling tool. If, at the end of the process, the model results to be not accurate enough, the procedure must be

repeated from the beginning.

A possible solution to this problem is the use of an adaptive algorithm which automatically selects a quasi-minimal distribution of data points able to characterize the overall behavior of the response, as it is explained in [10].

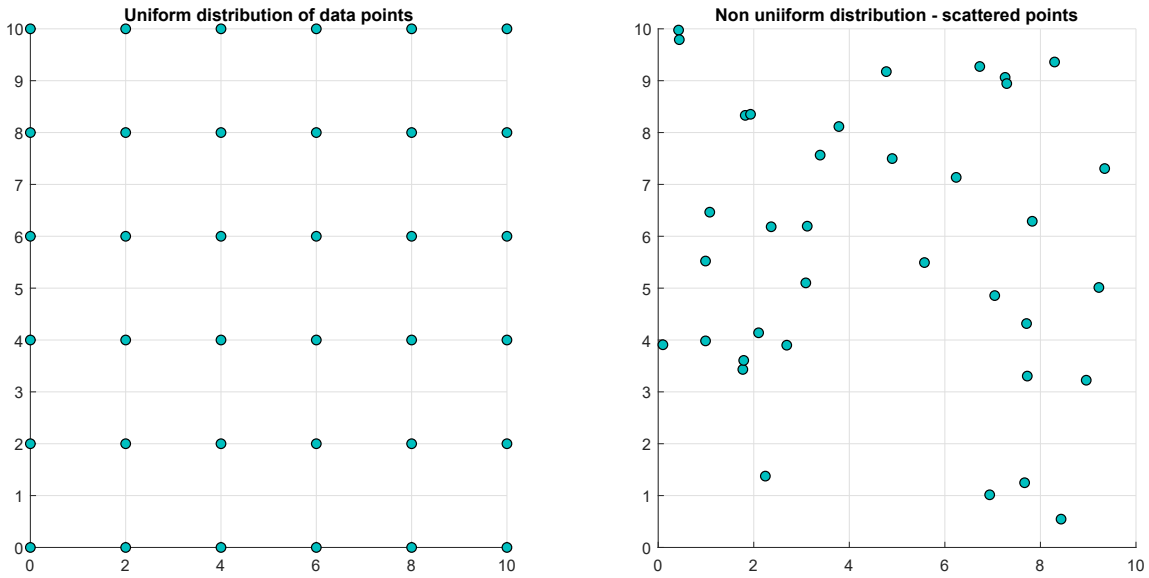


Figure 3.1: Comparison between uniform and non-uniform distribution of data points in space.

The algorithm works in an iterative way: it starts from an initial set of data points distributed over the parameter space, and it is able to identify at each step the best position for the data points to be added at the initial set, which eventually will be used to construct the model.

The final distribution of points, differently from what happens with the standard approach described before, is no more homogeneous, but it will be more dense in the regions that require a finer sampling (3.1, plot on the right).

The algorithm implemented for this research work, moreover, provides a complete integration with the field solver: during each iteration, after the identification of new data points, the algorithm automatically calls the solver to run the new simulations, collects the necessary results and runs the model generator tool. The accuracy of the model will act both as termination criterion for the algorithm and as a selection criterion for the new data points in the next iteration.

In the next section, the adaptive algorithm for the selection of new data points will be analysed in detail.

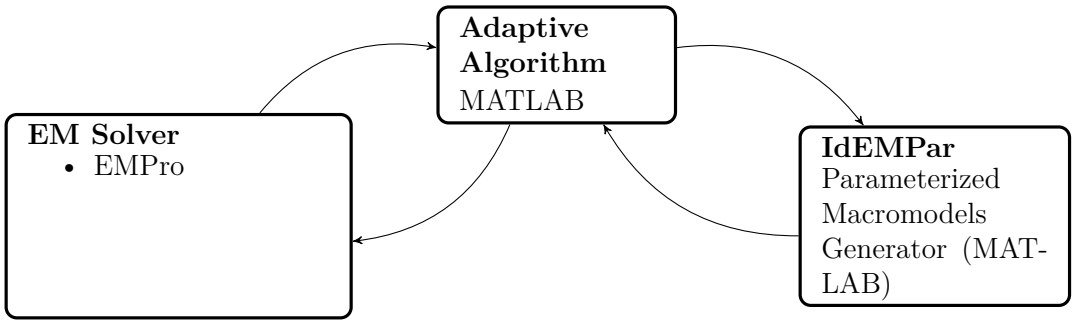


Figure 3.2: Block diagram of the framework in which the realized tool is placed

3.2 An adaptive algorithm for selection of new data points

3.2.1 Parameterized macromodels and the design space

Parameterized macromodels are created by fitting a set of responses $\widetilde{H}(s, \boldsymbol{\theta})$, where s is the complex frequency $s = j\omega$ and $\boldsymbol{\theta}$ is a vector containing all the design parameters $\boldsymbol{\theta} = \{\theta^{(m)}\}_{m=1}^{M}$. Since responses come from an Electromagnetic solver, frequency actually corresponds to a vector containing a set of discrete sampled frequencies $\{s_t\}_{t=1}^{T}$.
Each EM simulation corresponds to a specific *data point* in the design space, and consequently to a specific set of scalar values that parameters $\boldsymbol{\theta}$ assume in that simulation. Thus, if the set of data points is defined as $P = \{\boldsymbol{p}_k\}_{k=1}^{K}$, each point $\boldsymbol{p}_k$ consists of $p_k^{(1)}, ..., p_k^{(M)}$ scalar values associated to the parameters $\boldsymbol{\theta}$.
The adaptive algorithm is expected to choose the best position for data points in the parameter space, so that the best model accuracy for the minimum number of data points is obtained.

3.2.2 Criteria for the choice of new data points

The adaptivity of the algorithm lies in the choice of new data points at each iteration: if this choice is conveniently made, a small amount of points will be sufficient to characterize the complete behavior of the system.
In this case, three main criteria have been identified for point selection; these criteria are often in conflict, so a tradeoff among them guarantees the robustness of the algorithm.
The criteria are:

- *Exploration* - a first intuitive rule is to select new points in the regions of the space which do not contain points yet, and so which have not been yet characterized. This criterion guarantees that, at the end, the entire design space has been explored, and data points are spread in a uniform way. It is clear that this criterion does not take into consideration the response associated to data points, but just their location in space.

- *Exploitation* - this criterion ensures that points are placed in those regions of design

space that are characterized by a large variation in the response. Thus, if the behaviour of the system changes significantly is a specific region, there the sampling density will be more refined.

- *Model-Data relative error* - if the two previous criteria were based on the position of points and on the true response, this one takes into consideration the accuracy of the model constructed at the previous iteration. It relies on the idea that more points need to be added in the regions where the model is not sufficiently accurate, so where more information is necessary to approximate the system response.

The procedure is to adopt a metric for each selection criterion, which will help identifying the regions that are more suitable for the new points according to each rule. All metrics will be merged into a combined metric providing an ultimate ranking of data points. In the following paragraphs each criterion will be analysed more in depth.

3.2.3 Exploration

The metric associated with the exploration criterion must assess the distribution of data points. As suggested in [10], this is here achieved by computing the **Voronoi tessellation** of the parameters space.
The Voronoi tessellation is defined as the partitioning of a space based on the distance of each point belonging to the space from a set of predefined points, also called *seeds*. The plane is divided into regions containing one seed plus all those points which are closer to that seed than to any other present in the space. These regions are the so called **Voronoi cells**.
In the algorithm, the idea is to identify the Voronoi cells around each point already present in the design space. An estimation of the volume of each cell (or the area, if the parameter space is $\mathbb{R}^2$), will give an indication of the regions sampled more sparsely.
It can be demonstrated that the Voronoi diagram is the dual of the Delaunay triangulation, and usually the former is obtained by first calculating the latter, as it is depicted in 3.3.
The Delaunay triangulation, named after Boris Delaunay for his research on this topic back in 1934 [11], is a triangulation $DT(\Delta)$ on a given set of discrete points Δ such that no point in Δ is inside the circumcircle of any triangle in $DT(\Delta)$ [12].
In this algorithm as well, initially the Delaunay triangulation is calculated, then the corresponding Voronoi tessellation is retrieved: this is not the definitive tessellation yet, as the cells close to the edges of the space are unbounded, but in order to estimate their area/volume they must be bounded. The code that was implemented in this work, then, uses the edges of the design space as the polytope bounding the Voronoi diagram.
Once all the cells are bounded, their area is estimated by means of Monte Carlo method [13]. This method consists of the following steps:

- Include the area to be estimated inside a rectangle of known area;
- place a specific number of points inside the rectangle at random locations;
- count the number of points (over the total) which fall inside the unknown area;

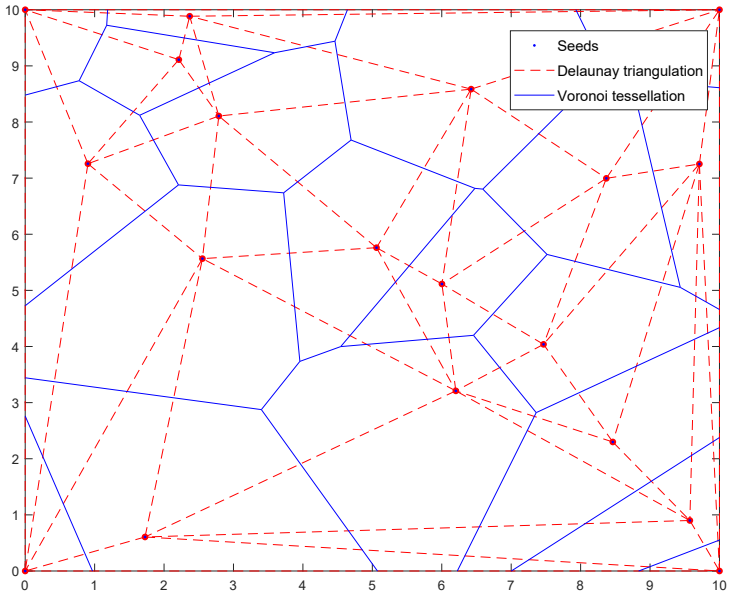


Figure 3.3: Superimposed plots of Delaunay triangulation and Voronoi tessellation.

- estimate the unknown area by means of this formula:

$$\text{unknown area} = \text{area of the rectangle x } \frac{\text{points inside the unknown area}}{\text{total number of points}} \tag{3.1}$$

Once all areas around data points have been estimated, a normalized metric $A(\boldsymbol{p}_k)$ is used:

$$A(\boldsymbol{p}_k) = \frac{Area(C_k)}{\sum_{k=1}^{k=K} Area(C_k)} \tag{3.2}$$

In this way $A(\boldsymbol{p}_k)$ gives a measure of the portion of space inside the Voronoi cell C_k around point $\boldsymbol{p}_k$.

3.2.4 Exploitation

This criterion identifies the regions in the design space where the system response varies more rapidly, and it does so by calculating a local linear approximation at each data point. As a first step, for each data point $\boldsymbol{p}_k$ in P a set of L neighbouring points is identified. This set can be labelled as $N(\boldsymbol{p}_k)$, where

$$N(\boldsymbol{p}_k) = \{\boldsymbol{p}_{kl}\}_{l=1}^{L}, \quad N(\boldsymbol{p}_k) \subset P \backslash \{\boldsymbol{p}_k\} \tag{3.3}$$

For each point $\boldsymbol{p}_k$, the neighbours are used for an estimation of the gradient $\nabla H(s_t, \boldsymbol{\theta})$ for each frequency s_t. The gradient allows to determine the local linear approximation at point $\boldsymbol{p}_k$ $\breve{H}(s_t, \boldsymbol{\theta})$.
A comparison of this approximation with the true response $H(s_t, \boldsymbol{\theta})$ at the neighbouring points $N(\boldsymbol{p}_k)$ gives a measure of the variability of the response in the area around $\boldsymbol{p}_k$.

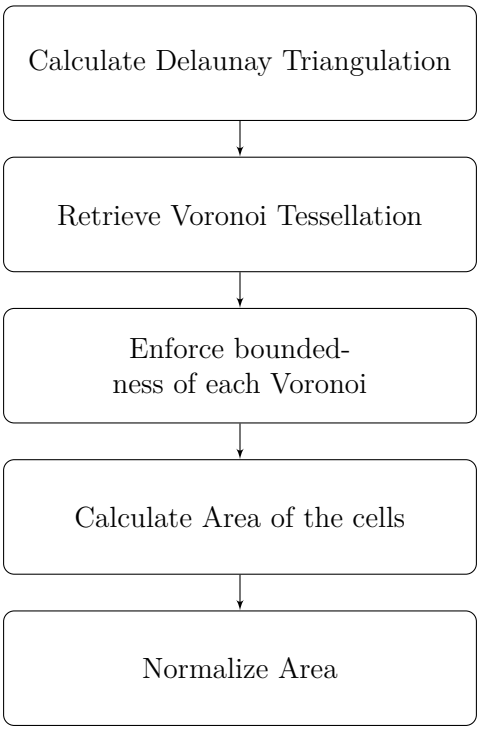


Figure 3.4: Flow chart of exploration based metric

Choice of neighbouring points

An important step in this procedure involves the choice of the neighbouring points $N(\boldsymbol{p}_k)$. In fact, they must lie as close as possible to the data point $\boldsymbol{p}_k$, but at the same time they must be placed as far as possible one from the other: in this way it is guaranteed that they cover the entire design space around point $\boldsymbol{p}_k$.
As indicated in [10], these two requirements are checked for each possible set of neighbouring points (having fixed the central data point $\boldsymbol{p}_{ki}$), and they are translated into two properties, namely *cohesion* and *adhesion*.

- Cohesion implies the neighbouring points must lie as close as possible to the point $\boldsymbol{p}_k$. In other words, the distance of each neighbour $\boldsymbol{p}_{kl}$ from the central point $\boldsymbol{p}_k$ must be the smallest possible, such that $C(N(\boldsymbol{p}_k))$ is minimized

$$C(N(\boldsymbol{p}_k)) = \frac{1}{L}\sum_{l=1}^{L} ||\boldsymbol{p}_{kl} - \boldsymbol{p}_k|| \tag{3.4}$$

- Adhesion implies that the neighbouring points must lie as far away from each other as possible, maximizing $D(N(\boldsymbol{p}_k))$

$$D(N(\boldsymbol{p}_k)) = \frac{1}{L}\sum_{l=1}^{L} \min\{||\boldsymbol{p}_{kl} - \boldsymbol{p}_{kw}||, \forall w \neq l\} \tag{3.5}$$

It is clear that cohesion and adhesion are in contrast, so the goal is to find the best trade-off maximizing both.
It is demonstrated in [14] that if we choose a set of neighbouring points containing $L = 2M$ elements, the best configuration will be the M-dimensional cross-polytope. A **polytope** is a geometric object with "flat" sides, and it is a generalisation in any number of dimensions of the three-dimensional polyhedron [15]. For example, a polygon is a 2-polytope.
A **cross-polytope**, instead, is a regular, convex polytope that exists in n-dimensions. A square, for example, is a 2-cross-polytope [16].
The idea, then, is to check which, among all possible sets of neighboring points, is closer to the cross-polytope configuration: this is done by defining a neighbourhood score

$$S(N(\boldsymbol{p}_k)) = \frac{R(N(\boldsymbol{p}_k))}{C(N(\boldsymbol{p}_k))} \tag{3.6}$$

where

$$R(N(\boldsymbol{p}_k)) = \frac{D(N(\boldsymbol{p}_k))}{\sqrt{2}C(N(\boldsymbol{p}_k))} \tag{3.7}$$

is called cross-polytope ratio, since it gives a measure of how much a neighbouring set is close to a cross-polytope. Looking at 3.7, it is intuitive that, if $R(N(\boldsymbol{p}_k))$ is 0, it means that also adhesion $D(N(\boldsymbol{p}_k)) = 0$, thus all neighbouring points lie in the same spot (the distance among them is 0). If, instead, $R(N(\boldsymbol{p}_k)) = 1$, it means that the neighbouring points form a perfect cross-polytope. This can be demonstrated in the simple two-dimensional case. Consider 3.5, where the black dot is the central point, for which a neighbourhood is defined, and the red dots are such neighbours.
The distance between each neighbour and the central point is 1, so the Cohesion factor, as expressed in 3.4, will be equal to 1.
The minimum distance among neighbouring points, instead, is equal to $\sqrt{2}$, so also Adhesion factor in 3.5 will be equal to $\sqrt{2}$. If we substitute these values in 3.7, it will result that the cross-polytope ratio is 1. The correctness of the results is confirmed by the fact that a 2-dimensional cross-polytope is a square (as the one formed by the red dots in 3.5).

Computing the gradient

The response gradient can be expressed as

$$\nabla H(s_t, \boldsymbol{\theta}) = \left(\frac{\partial H(s_t, \boldsymbol{\theta})}{\partial \theta^{(1)}}, \frac{\partial H(s_t, \boldsymbol{\theta})}{\partial \theta^{(2)}}, ..., \frac{\partial H(s_t, \boldsymbol{\theta})}{\partial \theta^{(M)}} \right). \tag{3.8}$$

From this, it is possible to compute the best local linear approximation around point $\mathbf{p}_k$:

$$\breve{H}(s_t, \boldsymbol{\theta}) = H(s_t, \boldsymbol{p}_k) + \nabla H(s_t, \boldsymbol{\theta})|_{p_k}(\boldsymbol{\theta} - \boldsymbol{p}_k). \tag{3.9}$$

The gradient is estimated by solving the system $A \cdot \nabla H(s_t, \boldsymbol{\theta})|_{p_k} = b$ where

$$A = \begin{bmatrix} p_{k1}^{(1)} - p_k^{(1)} & p_{k1}^{(2)} - p_k^{(2)} & \dots & p_{k1}^{(M)} - p_k^{(M)} \\ p_{k2}^{(1)} - p_k^{(1)} & p_{k2}^{(2)} - p_k^{(2)} & \dots & p_{k2}^{(M)} - p_k^{(M)} \\ \dots & \dots & \dots & \dots \\ p_{kL}^{(1)} - p_k^{(1)} & p_{kL}^{(2)} - p_k^{(2)} & \dots & p_{kL}^{(M)} - p_k^{(M)} \end{bmatrix} \tag{3.10}$$

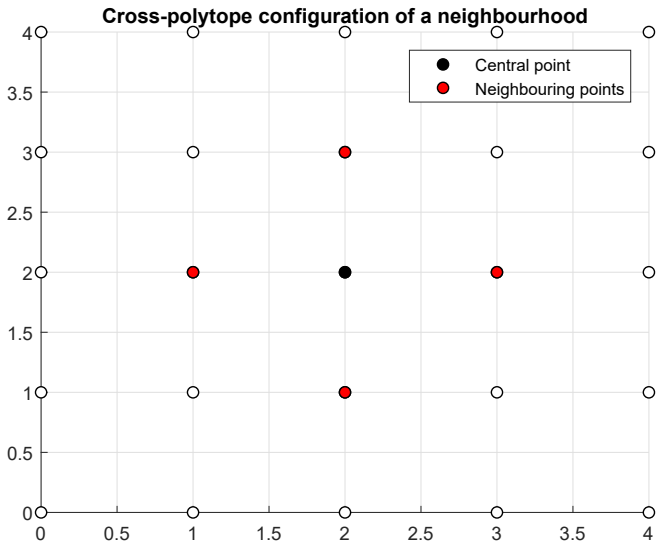


Figure 3.5: Cross-polytope configuration of neighbouring points

$$b = \begin{bmatrix} H(s_t, \boldsymbol{p}_{k1}) \\ H(s_t, \boldsymbol{p}_{k2}) \\ ... \\ H(s_t, \boldsymbol{p}_{kL}) \end{bmatrix} \tag{3.11}$$

where $\boldsymbol{p}_{k1}, \boldsymbol{p}_{k2}, ..., \boldsymbol{p}_{kL}$ are the neighbouring points of point $\boldsymbol{p}_k$. At this point, the best local linear approximation is compared the the original response at the neighbouring points for each frequency s_t. The exploitation metric is then obtained by estimating the gradient between the response of $\breve{H}(s_t, p_{kl})$ and the real response $H(s_t, p_{kl})$ at the neighbouring points, and maximizing the gradient among all frequency points.

$$Expl(\boldsymbol{p}_k) = max_t(\sum_{l=1}^{L} |\breve{H}(s_t, \boldsymbol{p}_{kl}) - H(s_t, \boldsymbol{p}_{kl})|) \tag{3.12}$$

In the algorithm the normalized version is used:

$$E(\boldsymbol{p}_k) = \frac{Expl(\boldsymbol{p}_k)}{\sum_{k=1}^{K} Expl(\boldsymbol{p}_k)} \tag{3.13}$$

The workflow for the estimation of the Exploitation metric is summarized in 3.6.

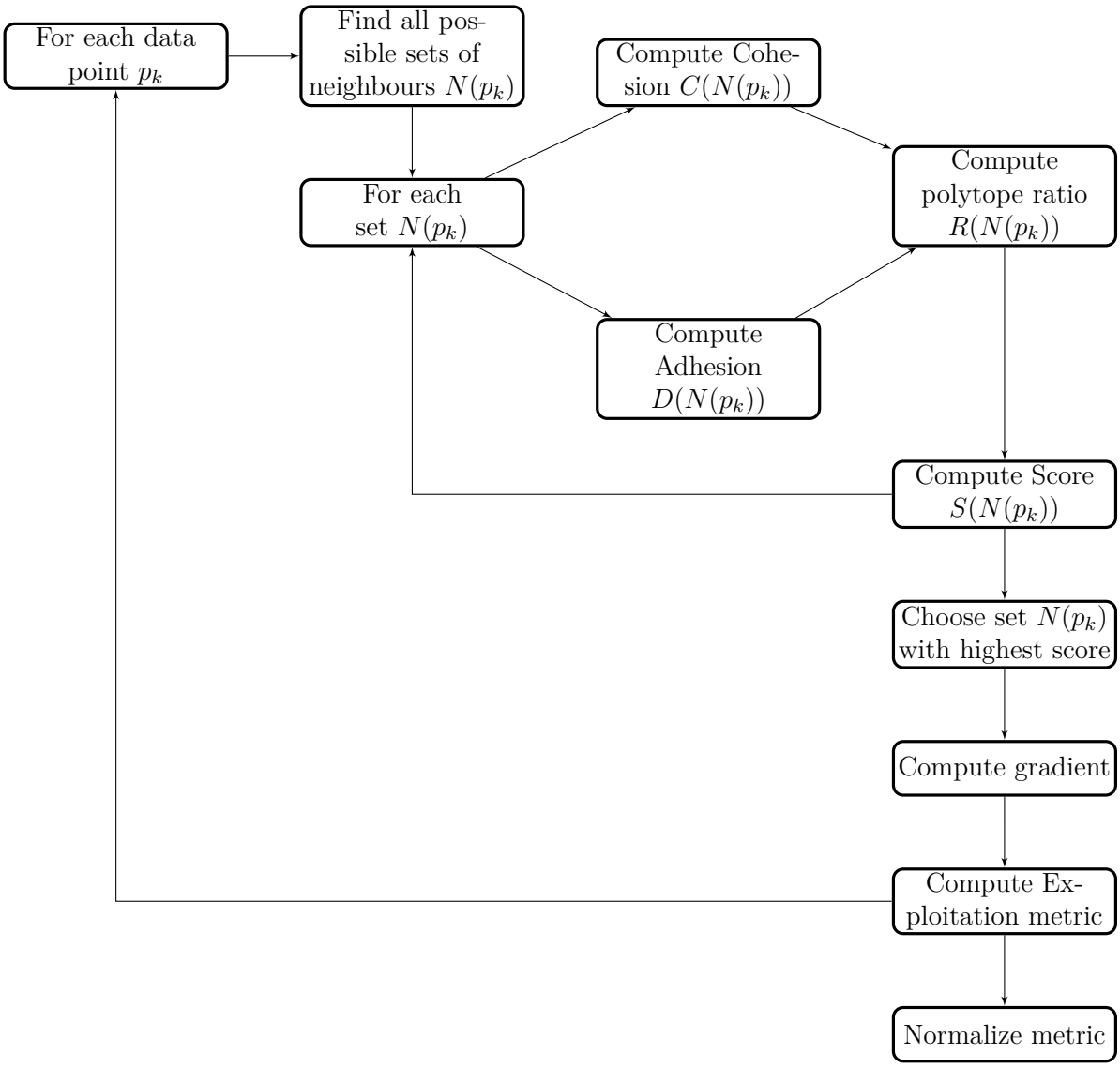


Figure 3.6: Flow chart of exploitation based metric

3.2.5 Data-Model relative error

The first two metrics guarantee a robust procedure for an optimal choice of data points. However, their value is established before any type of modeling has been performed, consequently not taking at all into consideration the quality of the final result. This type of approach is the so-called *data-driven* approach.
There are a number of reasons why this type of strategy can be preferred to the *model-driven* one: it is for sure faster, as no intermediate macromodel needs to be computed, and it is not dependent nor influenced by the chosen model type.
However, an accurate management of the orders of intermediate macromodels, together with some wise choices that allow to reduce the number of models to be generated at each iteration, allow the use of a model-driven approach where the additional computational time is compensated by an increase in the efficiency of the algorithm.
Creating the model at each iteration and identifying the regions where it is less accurate to drive the choice of the next points allows for a sort of feedback mechanism to be implemented.
For this reason, a third metric is used, namely the relative error between original data and the generated model in the chosen data points of the design space.
If we define the available data as $\widetilde{H}(j\omega;\boldsymbol{\theta})$ since they have been obtained from an EM solver, they will be sampled in frequency and they will refer to a particular set of parameters $\boldsymbol{\theta}$, corresponding to a particular point $\boldsymbol{p}_k$ in the parameter space. Thus, they can be rewritten as

$$\widetilde{H}(j\omega_t;\boldsymbol{p}_k). \tag{3.14}$$

If we call the model $H(j\omega;\boldsymbol{\theta})$, the relative error on each point $\boldsymbol{p}_k$ can be defined as

$$Error(\boldsymbol{p}_k) = \frac{\sqrt{\sum_t ||H(j\omega_t;\boldsymbol{p}_k) - \widetilde{H}(j\omega_t;\boldsymbol{p}_k)||^2}}{\sqrt{\sum_t ||\widetilde{H}(j\omega_t;\boldsymbol{p}_k)||^2}} \tag{3.15}$$

Also this metric, as the other two, is normalized, so that all metrics contribute in the same way to the final choice of points.

$$Err(\boldsymbol{p}_k) = \frac{Error(\boldsymbol{p}_k)}{\sum_{k=1}^{K} Error(\boldsymbol{p}_k)} \tag{3.16}$$

3.2.6 The final metric

The three presented metrics must be combined into an overall metric, which can be employed to rank data points. The global metric is

$$G(\boldsymbol{p}_k) = (1 + \alpha A(\boldsymbol{p}_k)) \cdot (1 + \beta E(\boldsymbol{p}_k)) \cdot (1 + \gamma Err(\boldsymbol{p}_k)) \tag{3.17}$$

where the coefficients α, β, γ allow for a parameterization of the metric, in the sense that the three individual criteria can be assigned different weights depending on their importance

in the choice of the next points. Up to now, the three parameters have been set to 1, so that each criterion contributes in the same way to the final metric. Data points with higher score will correspond to regions that need further sampling, while points with lower score are placed in regions already well represented by existing data points.
In this way, the algorithm selects the points with a larger G and requests from the solver the additional simulations at the center of the corresponding Voronoi cells.

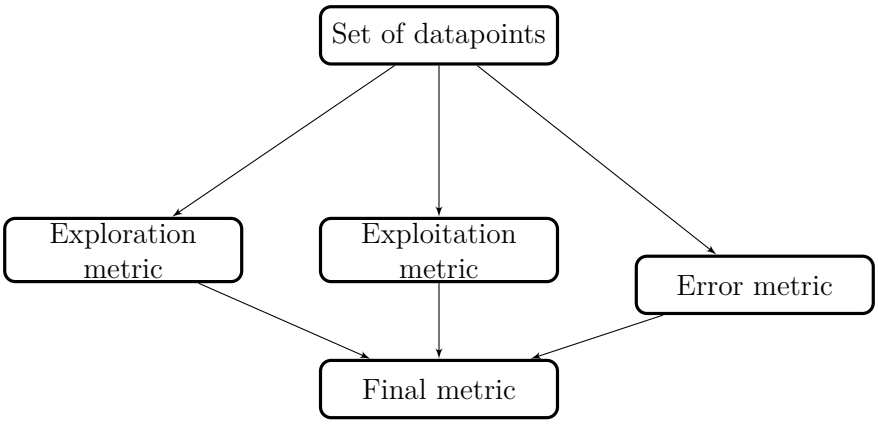


Figure 3.7: Flow chart for generation of final metric

3.3 A greedy algorithm for the selection of fitting points

In the previous section the adaptive algorithm for the selection of new data points has been presented. It has been clarified, as well, that the type of approach used in this work is *model-driven*, such that during intermediate stages of the process the construction of multiple parameterized macromodels must be managed.
A fundamental step in the generation of a model is the choice of fitting and validation points. If, in fact, a given number of responses is available for model construction, just a number of them is usually employed to fit the model: some responses are left out on purpose, as they will be used after the fitting process to "validate" the model, so to verify its quality.
Since the choice of the correct responses for the fitting process results to be often crucial, in the presented work this task is performed by a greedy algorithm, more specifically a **Vectorial Kernel Orthogonal Greedy Algorithm** (VKOGA) [17].
The primary purpose of VKOGA is approximating non-linear vectorial functions. Usually, in order to represent sparsely a non-linear function, an m-term approximation is employed, which expresses how well a linear combination of m functions belonging to a given set from a certain function space can approximate a function belonging to the same space.
It has been demonstrated that it is impossible to find in practice the best m-term approximation, thus greedy algorithms are generally used to identify quasi-optimal m-term approximations. Greedy algorithms, in fact, are iterative processes that, in order to find a global optimum, find the local optimum at each step.

In this algorithm, the space used is a particular type of approximation Hilbert space, the so called *reproducing kernel Hilbert spaces* (RKHS) induced by *kernels* [17]. A function $K : \Omega \text{ x } \Omega \to \mathbb{R}$ is a **positive definite kernel** if $\forall N \in \mathbb{N}$, $X = \{\boldsymbol{x}_1, ..., \boldsymbol{x}_N\} \subset \Omega$ and $\boldsymbol{\alpha} \in \mathbb{R}^N \backslash \{0\}$ we have

$$\sum_{i,j=1}^{N} \alpha_i \alpha_j K(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0. \tag{3.18}$$

From this definition of kernel it is possible to introduce RKHS.
Let $\Omega \subset \mathbb{R}^d$, $K : \Omega \times \Omega \to \mathbb{R}$ be a symmetric, positive definite kernel and $X \subset \Omega$. We denote by

$$B^X := \langle \{K(\boldsymbol{x}, \cdot) | \boldsymbol{x} \in X\} \rangle \tag{3.19}$$

the $\mathbb{R}$-vector space spanned by all functions $K(\boldsymbol{x}, \cdot)$, where $\langle \cdot \rangle$ denotes the span operation. We then denote by

$$B = \overline{B^\Omega} = \overline{\langle \{K(\boldsymbol{x}, \cdot) | \boldsymbol{x} \in \Omega\} \rangle} \tag{3.20}$$

the Hilbert space induced by K over Ω. In fact, for each symmetric, positive definite kernel K there is a unique such space with the reproducing property

$$\langle f, K(\boldsymbol{x}, \cdot) \rangle_B = f(\boldsymbol{x}) \forall f \in B, \boldsymbol{x} \in \Omega \tag{3.21}$$

which is the reason for the name **Reproducing Kernel Hilbert Spaces**.
The estimation of the m-term approximant is computed so that it guarantees the best possible approximation, thus by employing orthogonal projection with respect to the RKHS scalar product.
If $S \subseteq B$ is a linear subspace of B, then the orthogonal projection operator is denoted by

$$\begin{aligned} P_S &: B \to S \\ f &\mapsto P_S[\ f] \end{aligned} \tag{3.22}$$

such that

$$\langle f - P_S[\ f]\ , g \rangle_B = 0 \quad \forall g \in S. \tag{3.23}$$

3.3.1 A brief introduction on Reproducing Kernel Hilbert Spaces

In order to have a better understanding of the theory behind VKOGA, it is important to devote a few words to Reproducing Kernel Hilbert Spaces. RKHS are particularly popular in machine learning, but they have been employed in a variety of applications, from statistics to signal theory. Their main property, which is also the reason for their name, has been introduced in the previous section and it is the **reproducing property** (3.21). To give a less formal, but more intuitive definition of Reproducing Kernels, we can say that a reproducing kernel behaves as the Dirac delta, in the sense that it can evaluate a function in a point ($\delta_x[\ f]\ = f(x)$). Differently from the Dirac delta, however, it is a Hilbert space function.
This property is extremely useful when we want to find a correspondence between continuous functions and pointwise samples. We can restate the property of a reproducing kernel as

$$\langle k_\lambda(x), f(x) \rangle = f(\lambda) = \delta_\lambda[\ f] \tag{3.24}$$

where $k_\lambda(x)$ is the reproducing kernel. Now it is clear how the point evaluation of the Dirac delta is realized by the reproducing kernel by means of the inner product. The procedure is the same if we consider $f(x)$ as built from an orthonormal basis: what the reproducing kernel does is to project the signal onto the basis functions and reconstruct it from the basis function coefficient, thus confirming its reproducing property.
As an example, we can consider the space H^4_{Leg} formed by the first five Legendre polynomials, depicted in 3.8. This space is defined as the linear span of its five elements, as they form an orthogonal basis for the space.

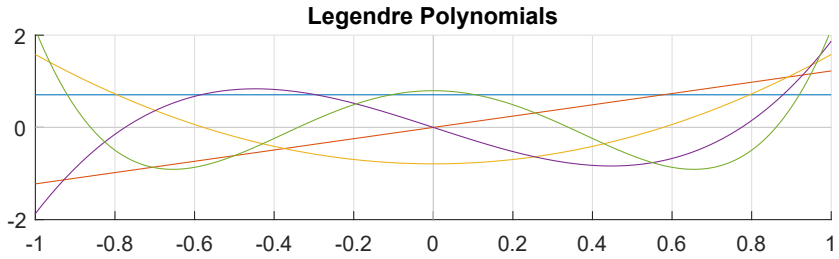


Figure 3.8: Plot of the first five Legendre Polynomials.

If three reproducing points are chosen (crosses in 3.9), it is possible to identify the corresponding reproducing kernels, depicted in 3.9. It is possible to see how the shape of the kernel depends on the position of the reproducing point.
If five reproducing points λ_i are well chosen (in the sense that they are evenly distributed), the corresponding five kernel functions will form a reproducing kernel basis, so that any function $f \in H^4_{leg}$ can be reproduced by knowing just the values $f(\lambda_i)$ in the reproducing points (see 3.10.

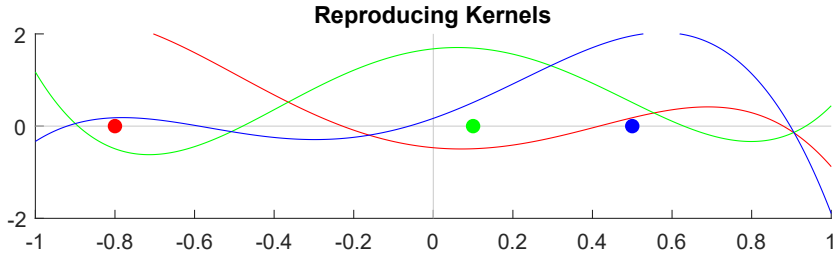


Figure 3.9: Example of three kernel functions for three different reproducing points.

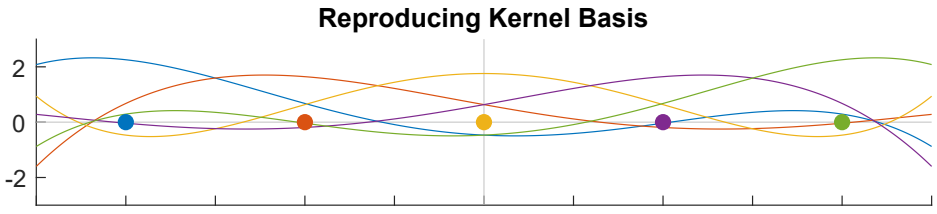


Figure 3.10: Example of kernel functions for five reproducing points.

This brief overview of Reproducing Kernel Hilbert Spaces and Reproducing Kernel Basis should give an idea of how they allow to deal with continuous functions in Hilbert space requiring only numerical samples as input [18].
In the following section the workflow of VKOGA, which is based on RKHS, will be analysed in more detail.

3.3.2 How VKOGA works

Many algorithms used in machine-learning, provided a set of input data, are able to create a mathematical model from them, so that they can learn from data and make data-driven predictions and decisions. The approach they use, common also to VKOGA, is to divide input data into three datasets:

- **Training dataset**: it includes a set of data used to train the model and fit its parameters. Once the model is created, it is run with the training dataset and its response is compared to the target, so that the parameters can be adjusted.
- **Validation dataset**: it includes another set of data which are not used for model training. Once the model has been generated, it is used to predict the responses of the validation dataset. This becomes particularly useful for overfitting detection.
- **Test dataset**: this last set of data is used to provide an unbiased evaluation of the final model.

What VKOGA does is to first split data into these three datasets. It then defines the Radial Basis Function (RBF) kernel, which can be Gaussian or of Wendland type.
Wendland functions are a family of compactly supported radial functions that start from a truncated power function [19]. They are always positive definite up to space dimension d_{max} and have smoothness C^k [20], [21]. An example of Wendland functions is reported in 3.1.

Table 3.1: WENDLAND RADIAL BASIS FUNCTIONS

$\phi(r)$	k	d_{max}
$(1-r)_+^2$	0	3
$(1-r)_+^4(4r+1)$	2	3
$(1-r)_+^6(35r^2+18r+3)$	4	3

The parameters that VKOGA is going to fit at each iteration (*greedy step*) are ϵ, the shape parameter, which determines the flatness and the width of the basis function, and λ, the regularization parameter, which helps avoiding overfitting.
The actual VKOGA consists in the choice of n selected points, among the entire training dataset, which are capable of approximating the non-linear target function.
The selection criterion used by the algorithm can be the *f-greedy* rule, if the target function is known, or the *P-greedy* one, if the target is unknown.
An example of how VKOGA works for the approximation of a function is given in 3.11.

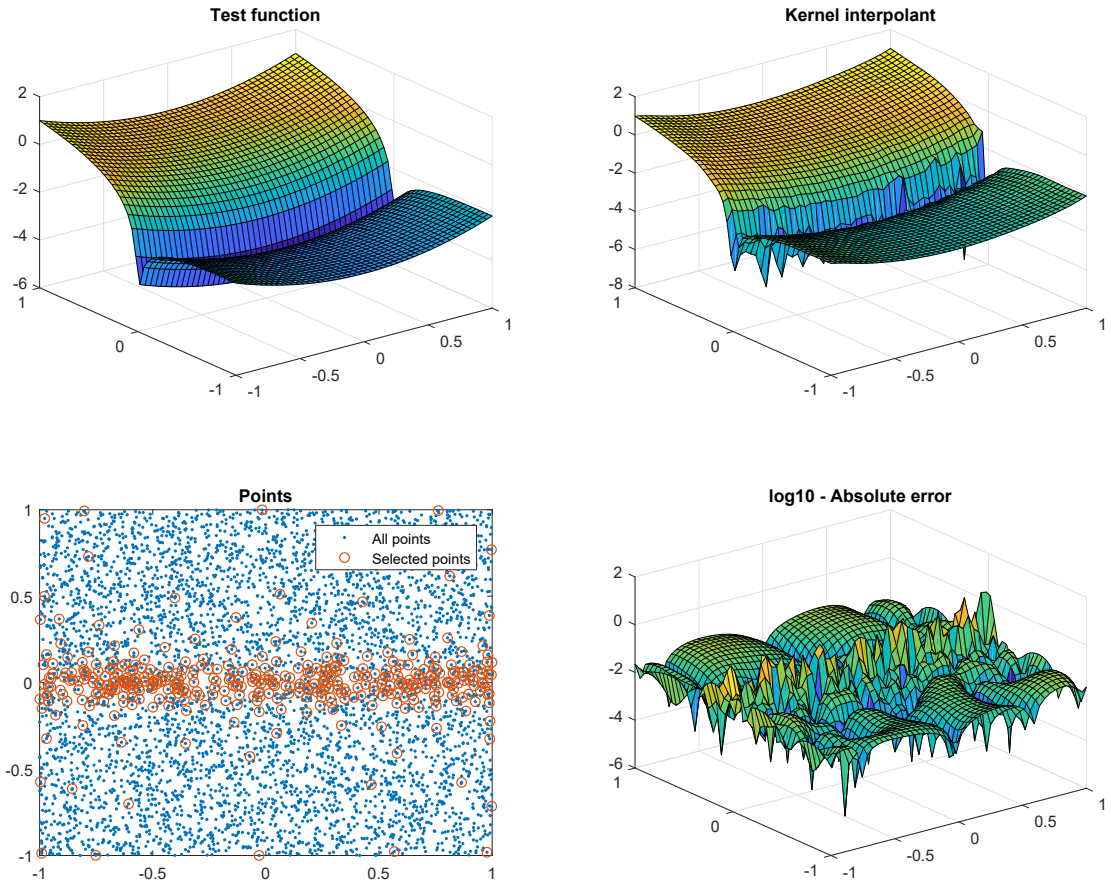


Figure 3.11: Example of VKOGA results for the approximation of a non-linear function.

As anticipated previously, VKOGA is used in the presented framework for the choice of points to be used for fitting in the parameterized model construction. The function used as target is the data-model error.

3.4 The complete algorithm workflow

Now that all the separate parts of the algorithm have been introduced, it is possible to present the complete workflow, step by step.
Such workflow can be divided into a preliminary part, executed only at the beginning of the algorithm, and an iterative part, which is repeated until the termination criterion is met.
The preliminary part consists of the following steps:

- it defines the number of parameters, their names and initial and final value for each parameter. From this it is possible to create the design space;
- it automatically selects n scattered points randomly placed in the design space;
- it calls the EM solver to run the simulations for the n initial points, and then it retrieves the results;
- it uses VKOGA to choose, among the n points, which are going to be used for fitting and which for validation;
- it automatically selects the correct dynamic order for parametric model construction. This is achieved by identifying the dataset with maximum phase variation and selecting the model with the smallest error among a range of models with possible dynamic orders. This order will remain fixed for the rest of calculations, as the number of points doesn't change during iterations;
- once the dynamic order has been identified, it generates all the models with all the possible orders of the parameters. The selection of the right orders is based on the errors of the generated models.

In the iterative part, instead:

- it chooses the best points in the parameter space that must be added, according to the global metric introduced before. The number of added points corresponds to $\frac{1}{3}$ of the points already present in the design space;
- it employs VKOGA to select fitting points for model construction;
- it creates the model with all possible combinations of parameters orders;
- according to the relative error on validation points, it selects the model with the right orders;
- it checks the final model error as a termination criterion for the algorithm. If the target accuracy has not been reached, the model errors will be used, together with the other metrics, to choose the new data points in the next iteration.

Regarding the first point in the iterative part of the algorithm, it must be clarified that the choice of adding $\frac{1}{3}$ new points is heuristic. The number of added points, in fact, must be a good trade-off: if it is too small, the number of iterations required to obtain a good model increases significantly; if it is too large, the adaptivity in the choice of new points is lost, making the use of the algorithm completely useless. Adding $\frac{1}{3}$ of the already present points has turned out experimentally to be a good trade-off, which allows to obtain an accurate model with a reasonable number of iterations, but still keeping the adaptivity of the choice intact.

3.4.1 Selection of model orders

As the realized framework is fully automated, the choice of the model orders (both dynamic order and parameters orders), which is generally done by the user, often with a trial-and-error procedure, needs to be done autonomously by the system. The adopted procedure, then, requires a further explanation.
The correct dynamic order is estimated at the beginning of process, and since it depends on the number of poles of the system, so on its frequency response, it doesn't change with the amount of responses considered, so it can remain fixed for the remaining calculations. For the order of the parameters, instead, the situation is more complicated, as its value changes with the number of available data points. However, some considerations can be made a priori.
Let us consider the final model expression as reported in 2.21 and 2.24:

$$\mathbf{H}(s;\theta) = \frac{N(s;\theta)}{D(s;\theta)} = \frac{\sum_{j=0}^{n}\sum_{\chi=1}^{X}\mathbf{A}_{j,\chi}\varphi_j(s)\xi_\chi(\theta)}{\sum_{j=0}^{n}\sum_{\chi=1}^{X}B_{j,\chi}\varphi_j(s)\xi_\chi(\theta)}, \tag{3.25}$$

If we assume to be in a case where two parameters are present, we can define their orders as χ_1 and χ_2, where

$$\begin{aligned}\chi_1 &= 1, ..., \overline{\chi_1}\\ \chi_2 &= 1, ..., \overline{\chi_2}\end{aligned} \tag{3.26}$$

and where $\overline{\chi}_1$ and $\overline{\chi}_2$ are the maximum allowed orders for each parameter. The variable χ can be introduced, equivalent to $\chi = \chi_1 \cdot \chi_2$. The following condition must be imposed:

$$\chi < \rho \cdot K = \overline{\chi} \tag{3.27}$$

where K is the number of data points in the design space, and ρ is a variable that must be strictly less than one, e.g. $\frac{1}{2}$ or $\frac{1}{3}$, in order to avoid overfitting by ensuring that the main least squares system 2.23 is sufficiently overdetermined. Both ρ and $\overline{\chi_1}, \overline{\chi_2}$ are parameters that are predefined by the user based on some a priori knowledge about the system to be modeled. Then, all the models with all the possible combinations of χ_1 and χ_2 complying with 3.27 are generated, and the one with the smallest overall error is chosen.
Since, by increasing K, also the parameter orders are likely to increase, the values of χ_1 and χ_2 for the current models will be the starting ones for the models in the next iteration, so that no time is wasted in computing models which will have for sure the wrong orders.

Chapter 4

Examples

After the analysis of the algorithm in Chapter 3, the next crucial point, which will be treated in this chapter, is its testing and validation.
The presented framework has been tested on a variety of applications, and its effectiveness demonstrated in all the analysed cases.

As already mentioned, the target of this work is the analysis of Electromagnetic structures, which is carried out by means of electromagnetic solvers. This process indeed requires a further step, since it needs the integration with such solver.
This, together with the large time usually necessary to run an electromagnetic simulation, made it convenient to test and validate the tool first on an example which, although consisting of a microwave structure, did not require the use of field solvers.

The structure taken into consideration is the **Transmission Line filter** of 4.1. This object is composed of single transmission line segments, whose length can be parameterized, and three parallel stubs, each one connected to a load, whose reflection coefficient Γ is left as a parameter as well.
The nominal parameters of this structure are presented in 4.1. This structure should be considered as an academic example, specifically designed to test the main algorithm. Each transmission line segment is characterized by its per-unit-length parameters, here computed using an integral method taking into account skin and proximity effect [22], [23] . Per unit length resistance and inductance are therefore frequency-dependent, leading to a challenging example for rational approximation (which cannot be exact).

It must be noticed that the structure reported in 4.1 is the same as the one reported in 6.1: the same structure, in fact, has been tested multiple times by changing the varying parameters. Each combination of parameters, in fact, leads to a completely different set of responses, thus generating examples which are completely independent one from the other, each of them worth analysing.

The left-end part and the right-end part of the circuit are used as the two ports, taken as references to calculate S-parameters.

Table 4.1: TABLE OF TRANSMISSION LINE FILTER PARAMETERS

Parameter	**Value**
L_1	7 mm
L_2	7 mm
L_3	7 mm
L_4	7 mm
L_5	1 mm
L_6	1 mm
L_7	1 mm
Γ_1	0.5
Γ_2	0.5
Γ_3	0.5

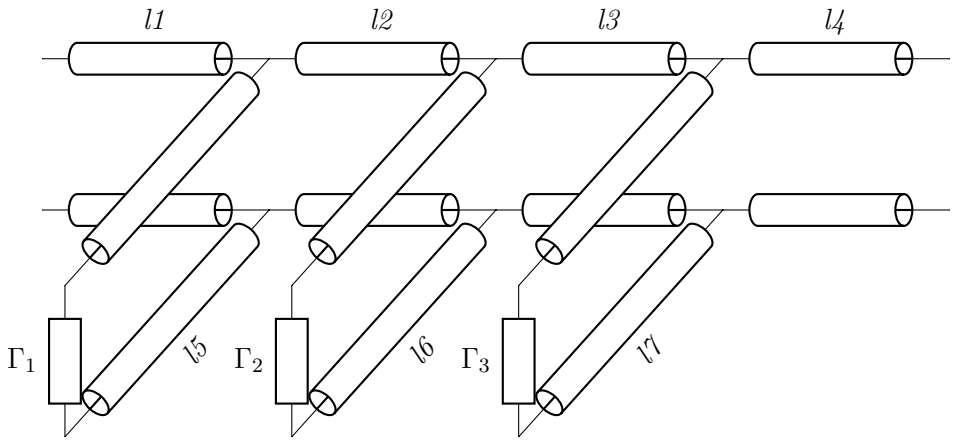


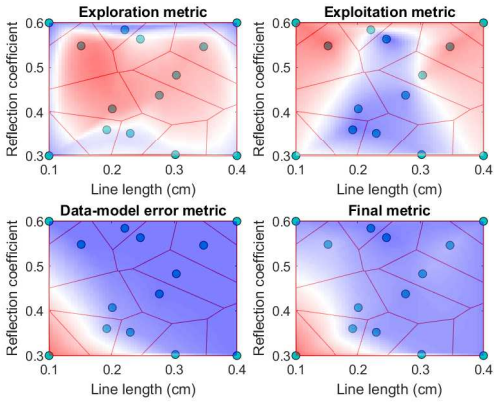
Figure 4.1: Schematic of transmission line filter

Obviously, in this example there are many possibilities for the choice of the two design parameters (namely, the length of internal lines, the length of the stubs and the impedance of the loads), so different combinations of them have been taken into consideration and analysed. This allows to see the behavior of the algorithm in different situations, for example when the two parameters have different variability (so the parameter orders are widely different).
In the following sections the results of each case are reported.

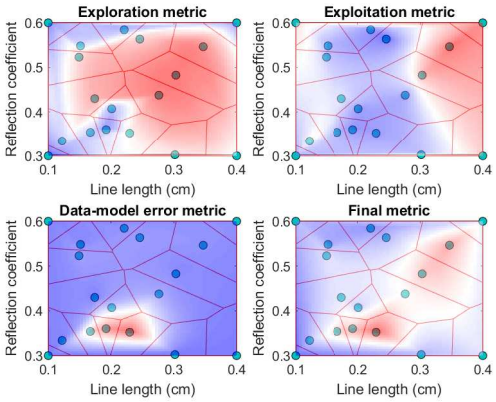
4.1 TL filter with varying stub length and load

In this first case the first parameter is the length of the central stub $l6$ and its load, Γ_2. Stub length ranges from 1 mm to 4 mm, while the reflection coefficient Γ_2 varies as $sin(\alpha)$, with α going from 0.3 to 0.6.

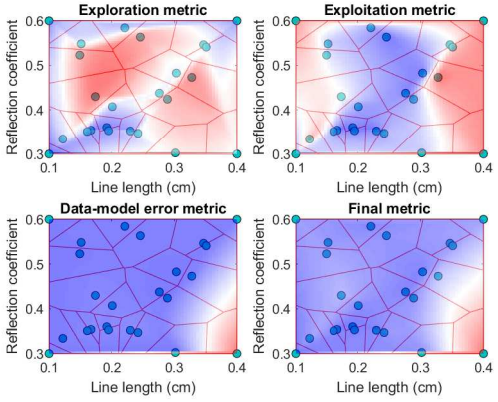
The algorithm performs 3 iterations before the necessary accuracy is reached. In the following pages the results at the end of each iteration are reported in 4.2, 4.3 and in 4.4, and they are summarized in 4.2. A final plot of the model responses compared to original data is reported in 4.5, separating fitting and validation responses.



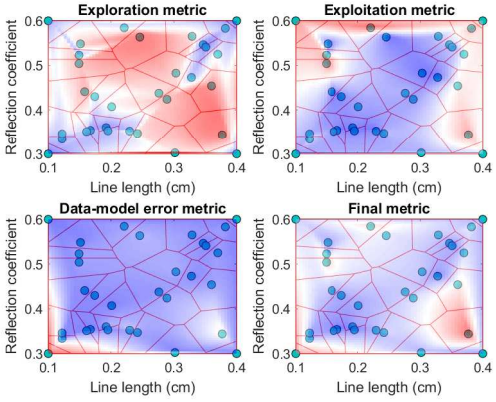
(a) First iteration



(b) Second iteration

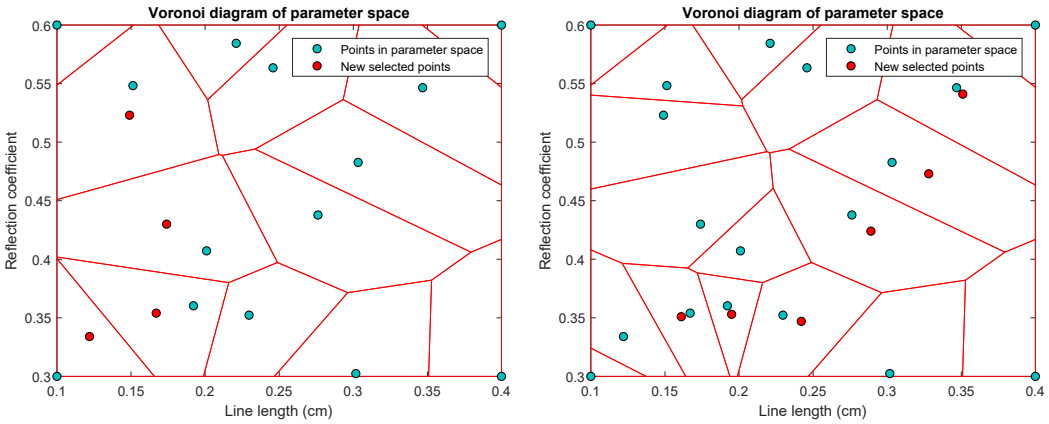


(c) Third iteration



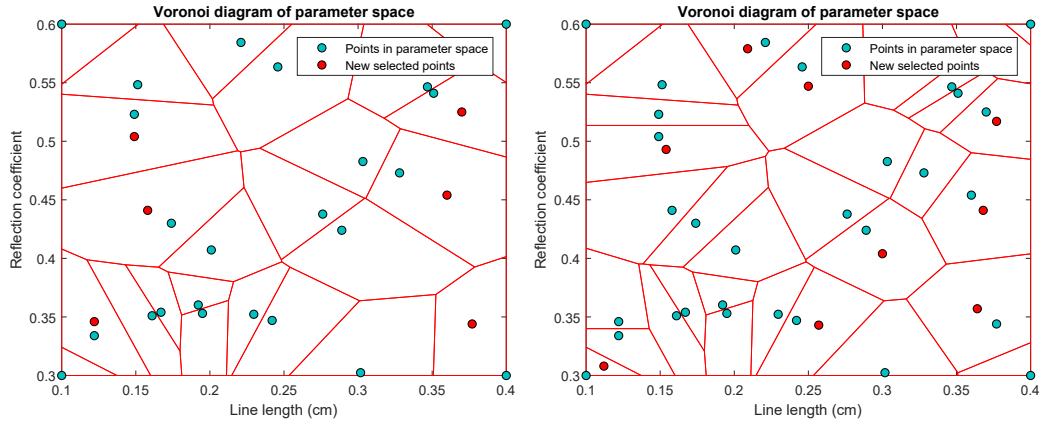
(d) Fourth iteration

Figure 4.2: Plots of the metrics (single and combined) at each iteration.



(a) First iteration

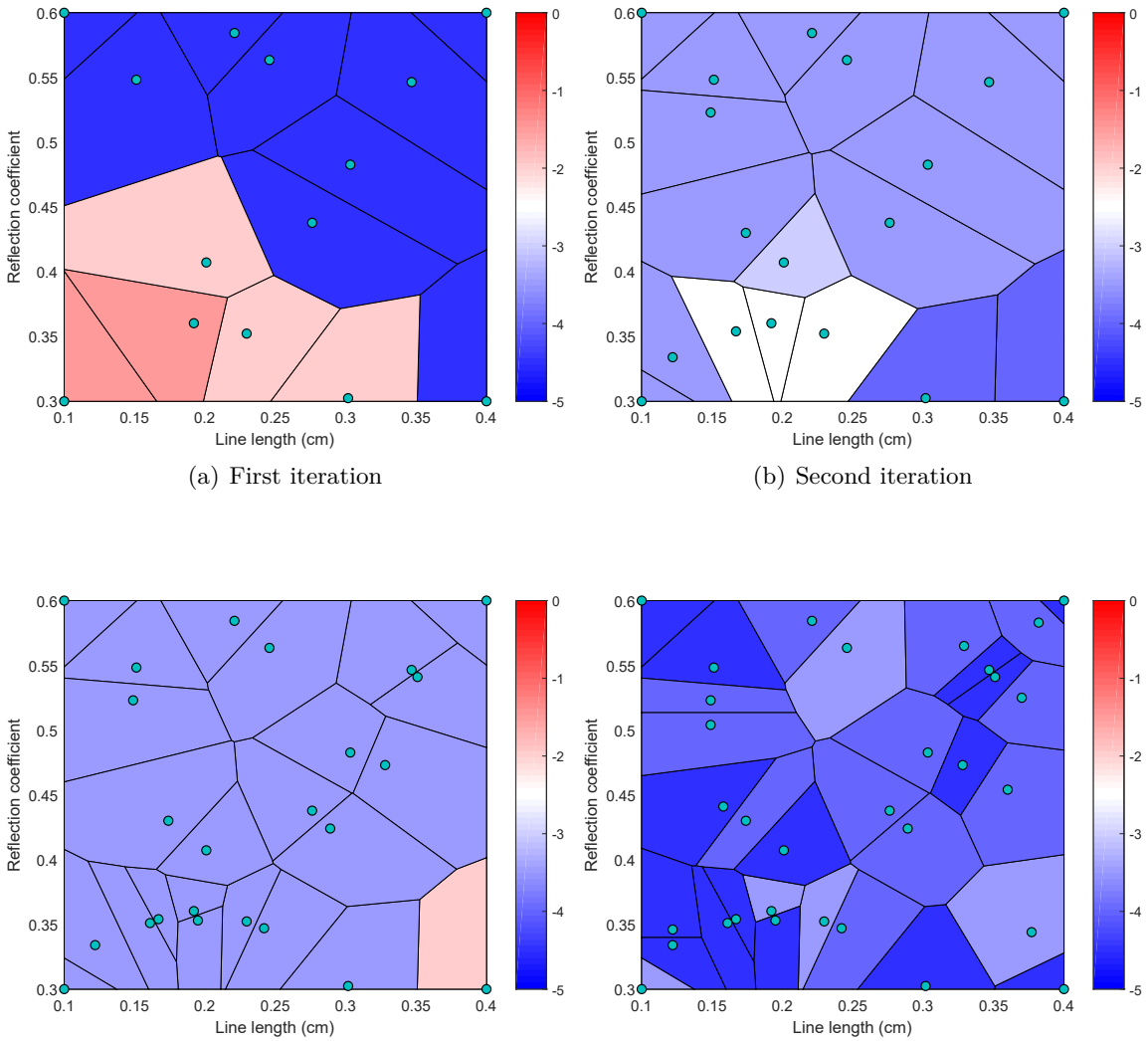
(b) Second iteration



(c) Third iteration

(d) Fourth iteration

Figure 4.3: Plots of the distribution of data points and selection of the new ones at each iteration.



(a) First iteration

(b) Second iteration

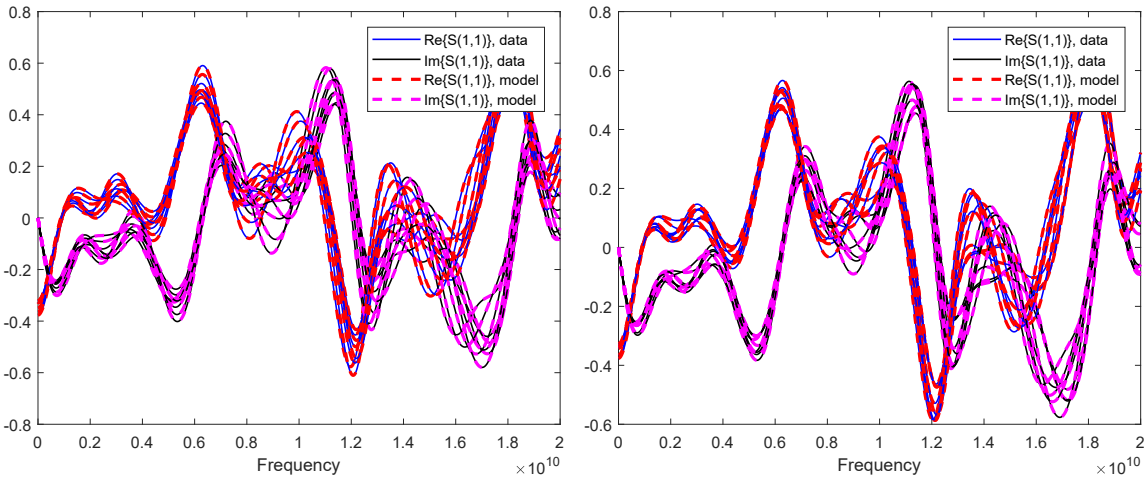
(c) Third iteration

(d) Fourth iteration

Figure 4.4: Plots of the model error at data points for each iteration.

Table 4.2: TABLE OF RESULTS FOR STUB LENGTH AND LOAD CASE

Iteration	**N. of points**	**Par.1 order**	**Par.2 order**	**Model error**
1	14	2	1	0.0499
2	18	2	1	0.0018
3	24	2	1	0.0052
4	32	3	1	4.279e-4



(a) Final model, selection of fitting responses

(b) Final model, validation responses

Figure 4.5: Final model

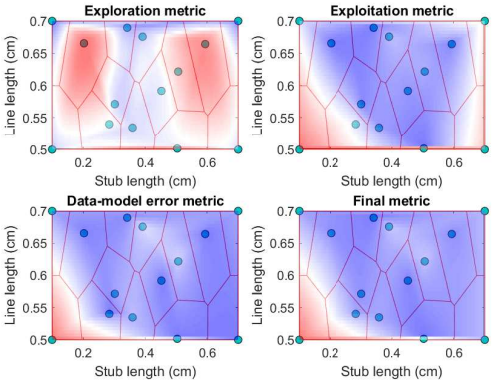
4.2 TL filter with varying internal line length and stub length

The second example is still related to the Transmission Line filter, but this time the two parameters are the length of the internal line l_2 and the length of the central stub l_6. As it can be seen in 4.3, the parameter orders are higher than in the previous example, and the final accuracy reached is worse. A number of other tests have been conducted on this example, changing the parameters and their range of variation. Even if, obviously, each case is different, all of them are similar in terms of quality of the results, confirming the effectiveness of the algorithm. For this reason, it has been considered unnecessary to include all of them in this thesis.

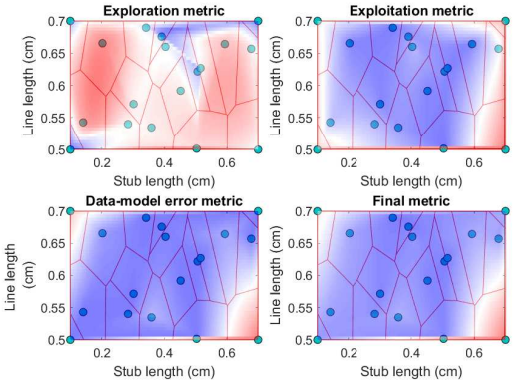
A final representation of model responses compared to original data can be found in 4.9.

Table 4.3: TABLE OF RESULTS FOR STUB LENGTH AND LINE LENGTH CASE

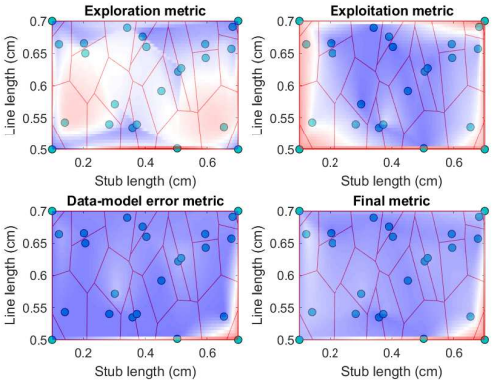
Iteration	**N. of points**	**Par.1 order**	**Par.2 order**	**Model error**
1	14	1	2	0.4744
2	18	2	2	0.0806
3	24	2	2	0.0646
4	32	2	3	0.0083
5	42	3	3	0.0047



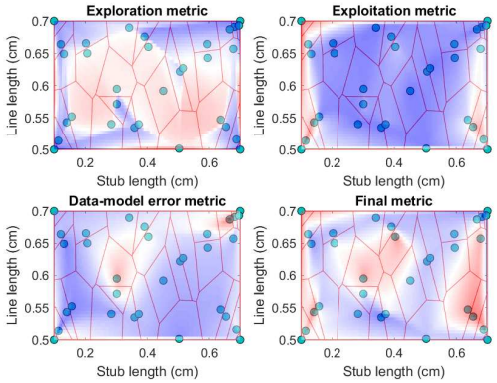
(a) First iteration



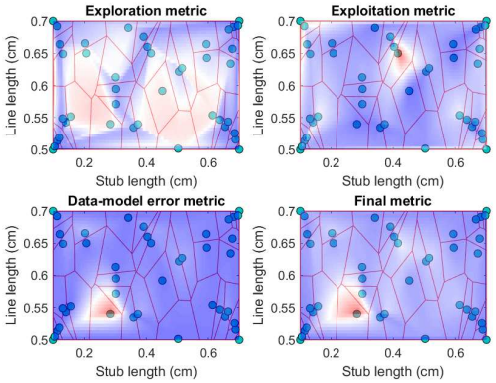
(b) Second iteration



(c) Third iteration

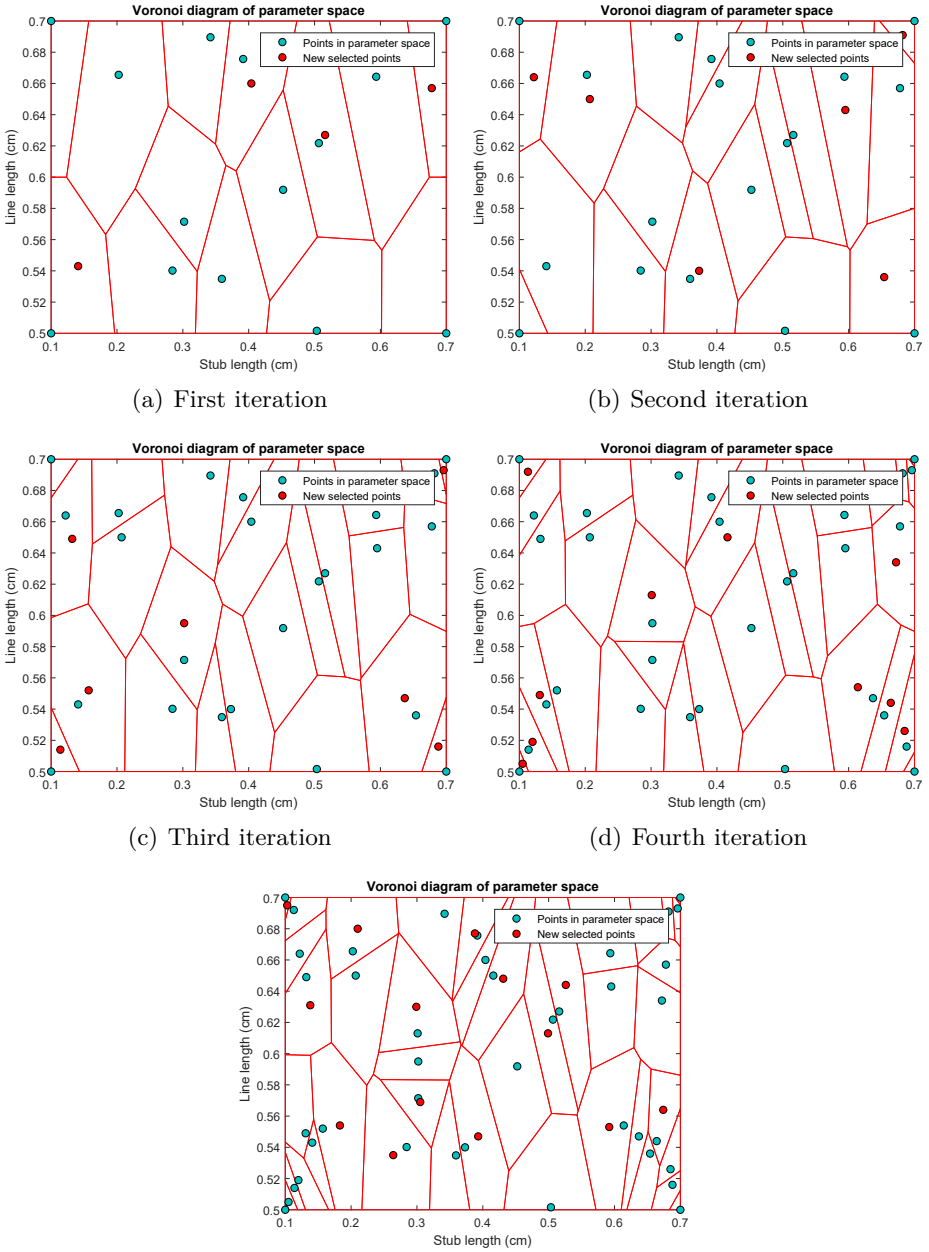


(d) Fourth iteration



(e) Fifth iteration

Figure 4.6: Plots of the metrics (single and combined) at each iteration, stub-line case.



(a) First iteration

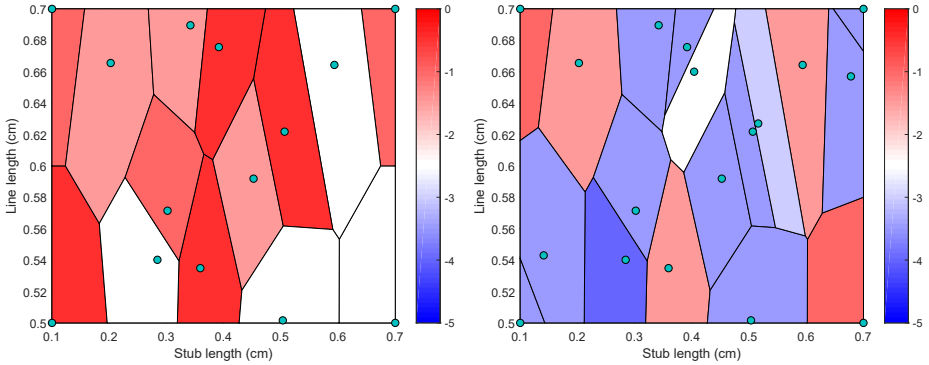
(b) Second iteration

(c) Third iteration

(d) Fourth iteration

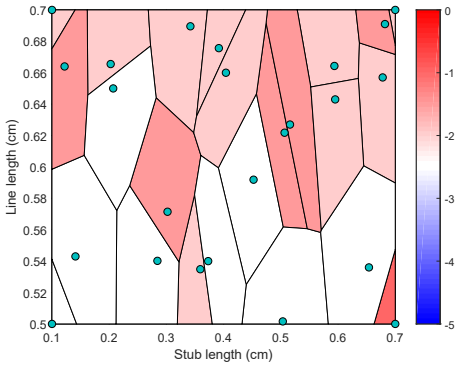
(e) Fifth iteration

Figure 4.7: Plots of the distribution of data points and selection of the new ones at each iteration, stub-line case.

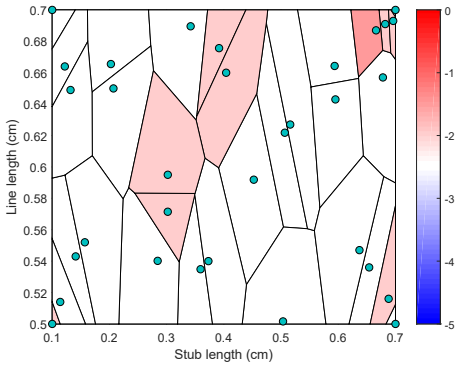


(a) First iteration

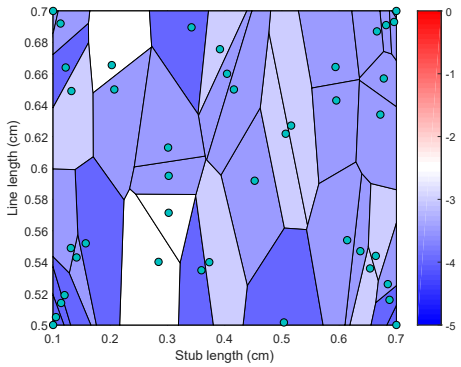
(b) Second iteration



(c) Third iteration

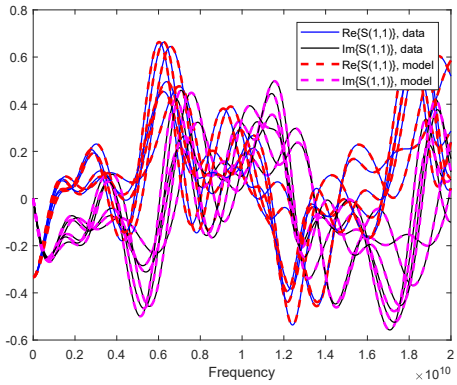


(d) Fourth iteration

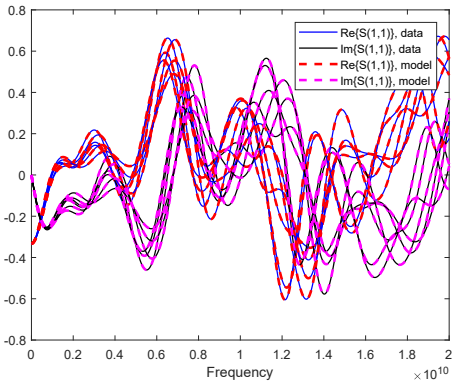


(e) Fifth iteration

Figure 4.8: Plots of the metrics (single and combined) at each iteration, line-stub case



(a) Final model, selection of fitting responses



(b) Final model, validation responses

Figure 4.9: Final model

4.3 Microstrip low-pass filter

This example, differently from the previous ones, involves the use of a field solver for the simulation of the structure, specifically Keysight EMPro. The object under analysis is a planar microstrip passive filter. The stepped impedance low-pass filter is displayed in 4.10. The structure is composed of a rectangular substrate block (relative permittivity=3, Conductivity = 0 S/m), with dimensions 12mm x 10mm x 0.64mm. The Strip Line, instead, is made of PEC and it has been modeled as a Sheet Body coming from the union of single rectangles. On the left-end side and right-end side of the structure, two ports have been inserted for the measurement of S-parameters. The simulation is run in time domain (FDTD engine), so the cell size for the FDTD grid has been fixed to 0.1mm in the three dimensions, giving Mesh priority to the PEC objects of the structure.
Absorbing boundary conditions have been set in X and Y direction, while in Z the boundary is PEC (Lower Boundary). Each simulation has a fixed duration of 10000*timestep, where the timestep value is automatically chosen by the software, depending on object geometry. It has been verified in time domain that the duration is sufficient for the signals to reach complete convergence (all energy has been dissipated). The structure has been designed to be completely parameterized. For this test, the two parameters chosen are the width of the two squares, called l_1 and l_2. Both of them range from 2 mm to 3.5 mm. The algorithm required 6 iterations to obtain an accurate model. A final representation of the model can be found in 4.14.

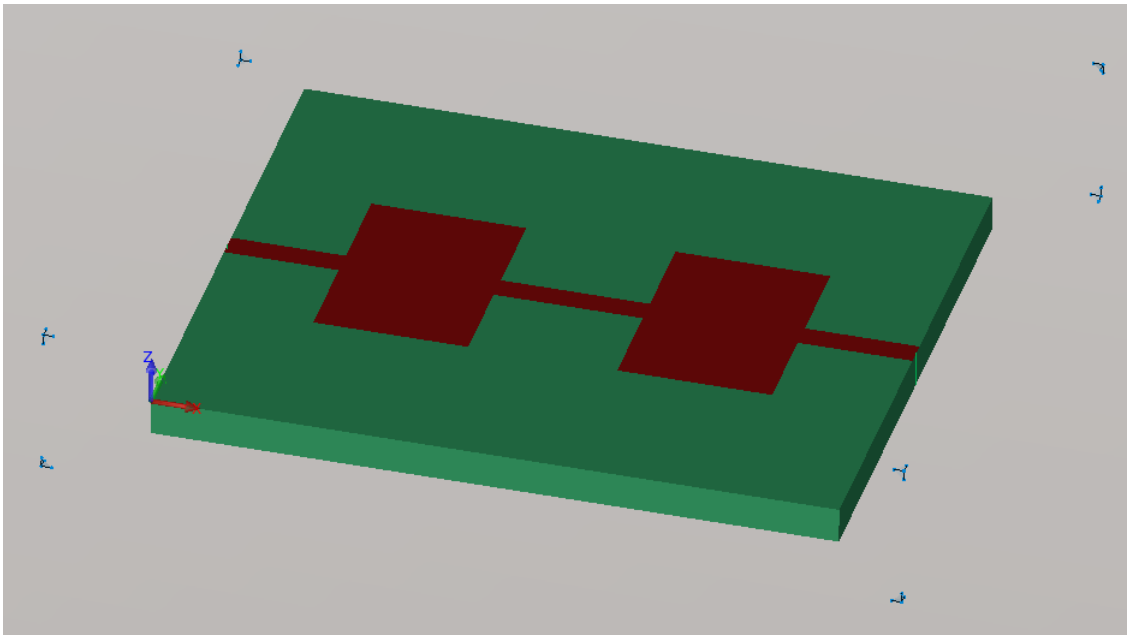
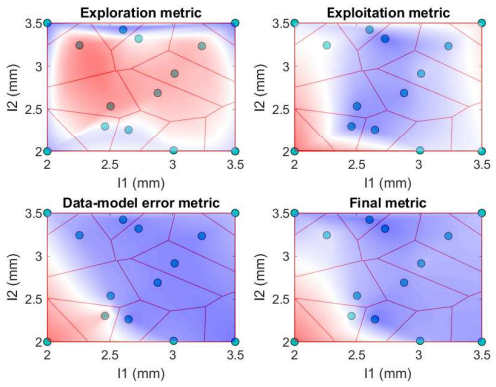


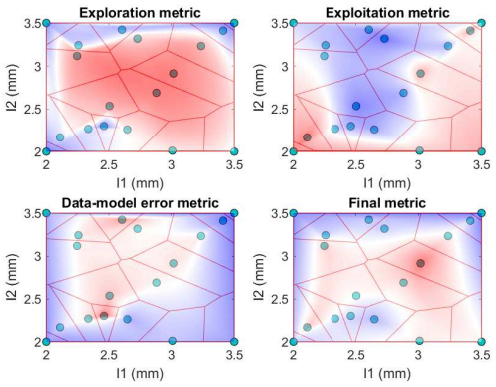
Figure 4.10: 3D representation of the microstrip filter

Table 4.4: TABLE OF RESULTS FOR MICROSTRIP LOW PASS FILTER

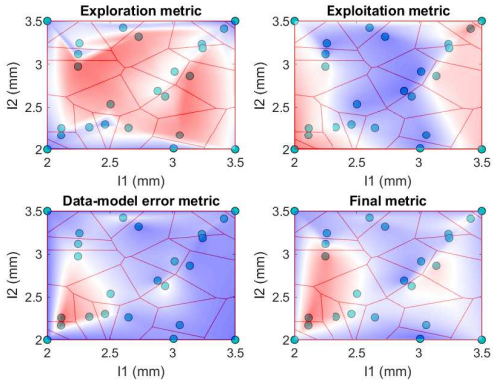
Iteration	**N. of points**	**Par.1 order**	**Par.2 order**	**Model error**
1	14	2	1	0.4443
2	18	2	2	0.0682
3	24	2	2	0.1040
4	32	2	2	0.0532
5	42	2	2	0.0047
6	56	2	2	0.0009



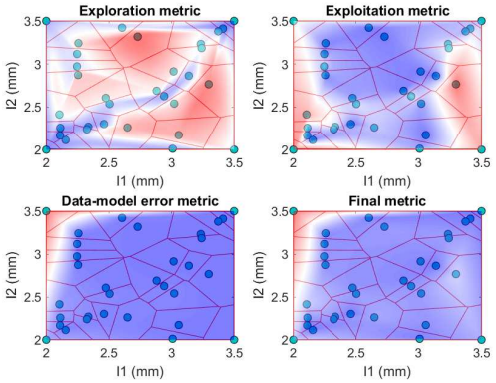
(a) First iteration



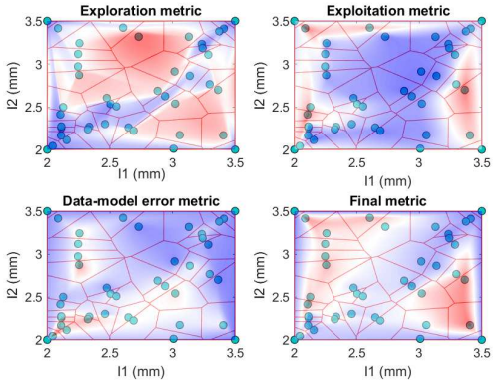
(b) Second iteration



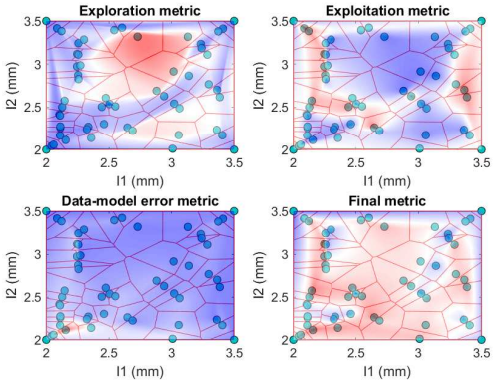
(c) Third iteration



(d) Fourth iteration

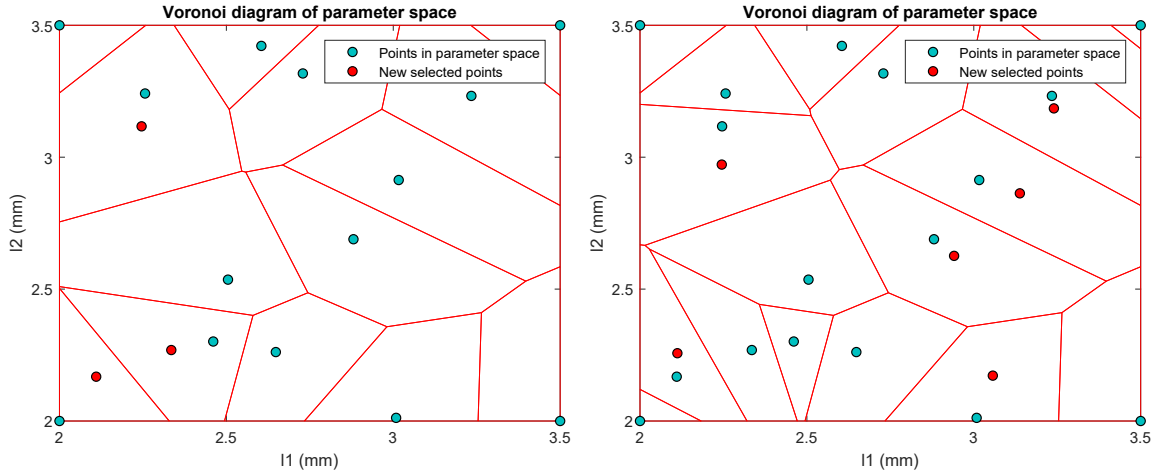


(e) Fifth iteration



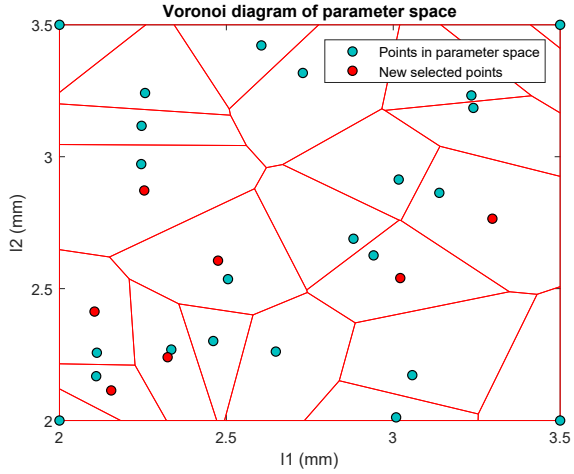
(f) Sixth iteration

Figure 4.11: Plots of the metrics (single and combined) at each iteration, low-pass filter case

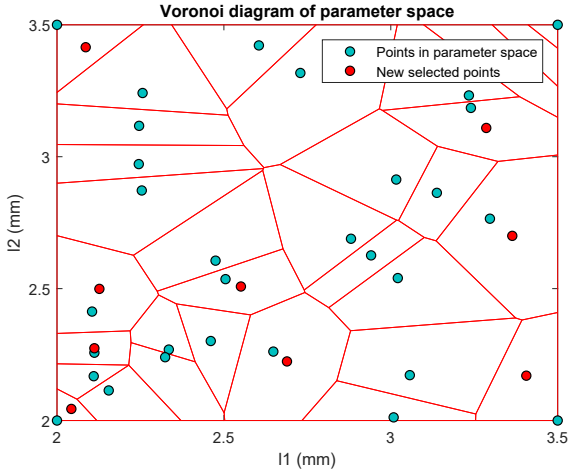


(a) First iteration

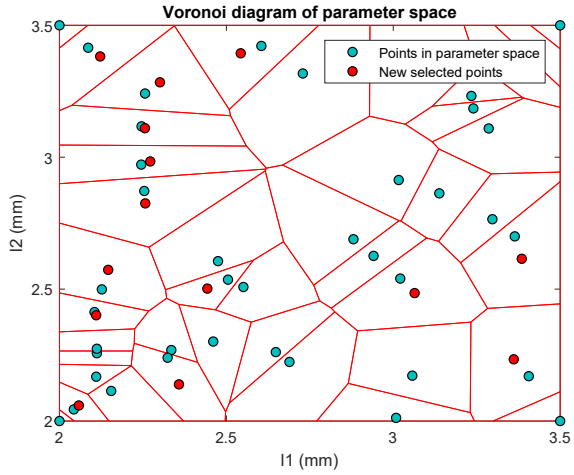
(b) Second iteration



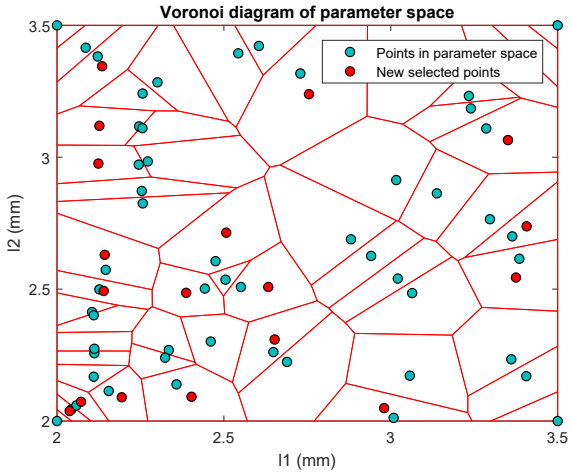
(c) Third iteration



(d) Fourth iteration

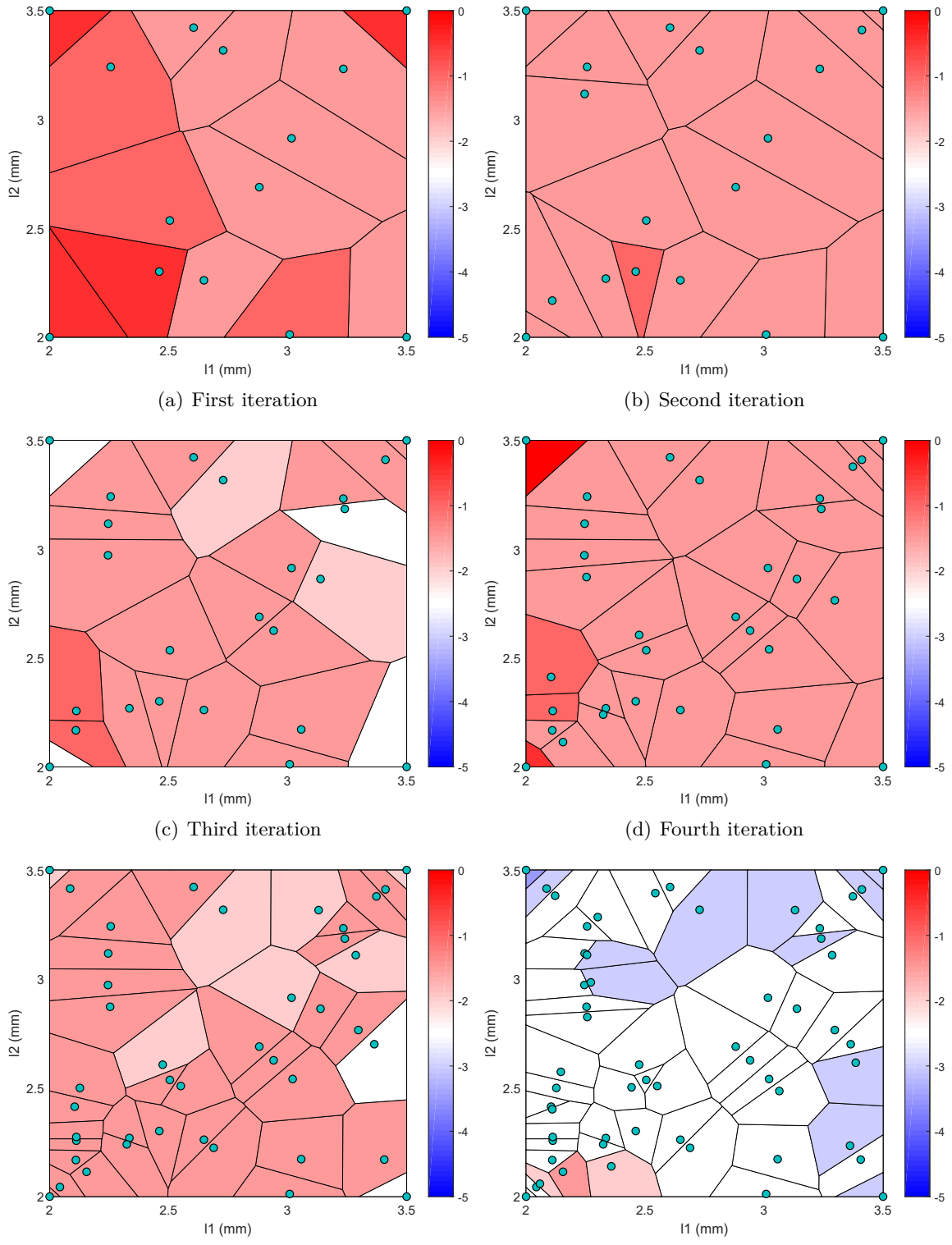


(e) Fifth iteration



(f) Sixth iteration

Figure 4.12: Plots of the selection of new points at each iteration, low-pass filter case



(a) First iteration

(b) Second iteration

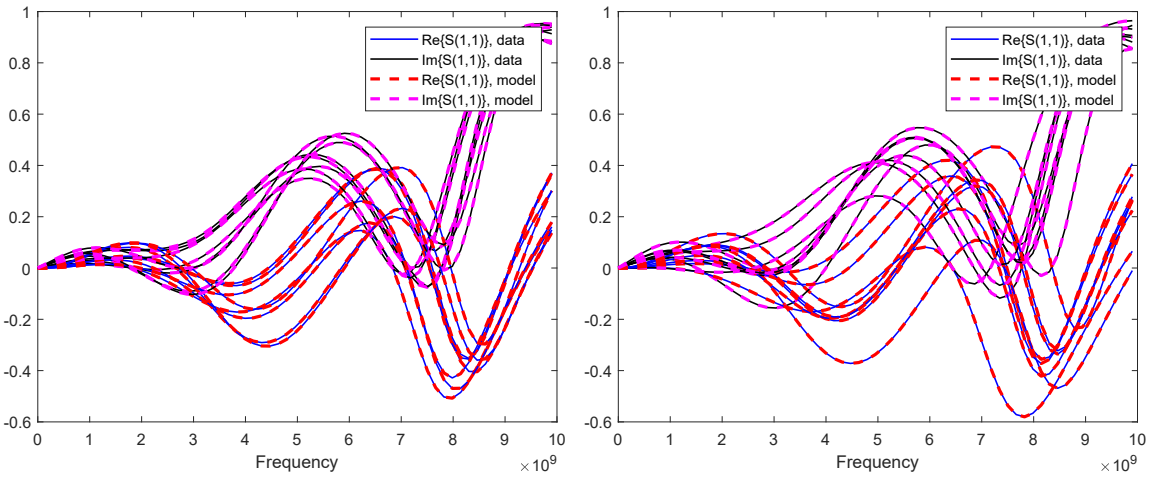
(c) Third iteration

(d) Fourth iteration

(e) Fifth iteration

(f) Sixth iteration

Figure 4.13: Plots of model error at each iteration, low-pass filter case



(a) Final model, selection of fitting responses

(b) Final model, validation responses

Figure 4.14: Final model

4.4 Microstrip patch antenna

The last example presented in this thesis is a microstrip patch antenna. The structure is made of a rectangular substrate block 29.6mm × 23mm. The substrate has relative permittivity =2.2 and conductivity = 0 S/m. On top of the substrate, two rectangles made of PEC are modelled as Sheet Bodies. One rectangle is 16mm × 12.45mm, while the stub is 4mm × 2.5mm. The width of the stub is parameterized, and it ranges from 2mm to 5mm, as well as the position of the stub, which can slide in the y-direction (from 7mm to 8mm measured from the top side of the object). The port of the structure is placed on one side of the stub.
The structure has been simulated in Keysight EMPro with FEM Engine.

The results provided by the algorithm are reported in the figures below. 4.16 contains the evaluation of the metrics for the 4 iterations that the algorithm performed to obtain an accurate model, 4.17 the selection of new points for each iteration, and 4.18 the model errors.
After 4 iterations, the final model error is reported in 4.19, together with a logarithmic plot of the model error at each iteration.
A summary of numerical results can be found in 4.5 and a final representation of the model responses compared to original data is provided in 4.20.

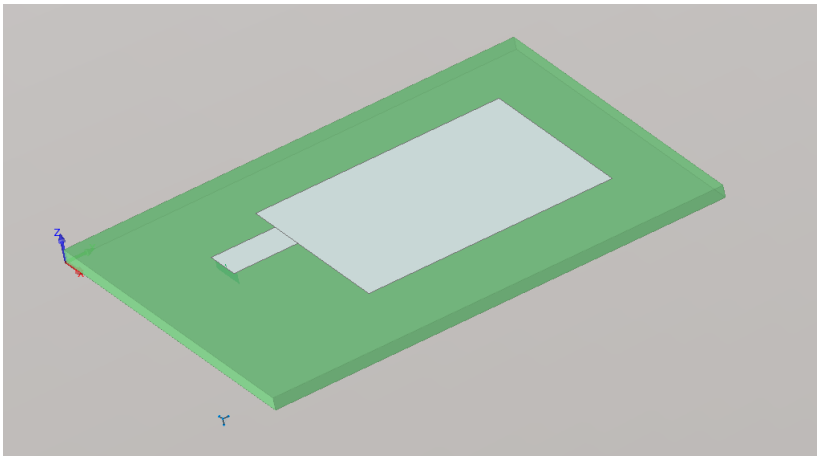
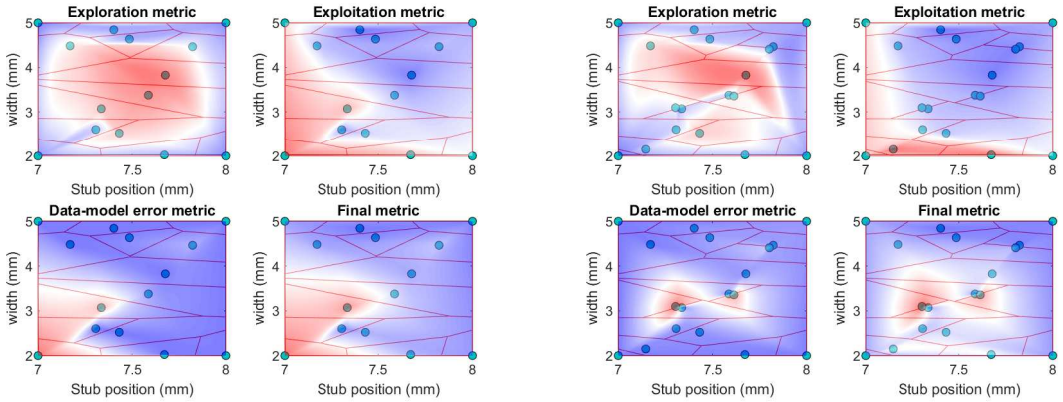
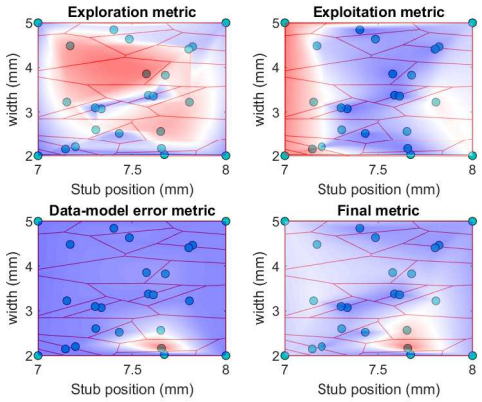


Figure 4.15: 3D representation of the microstrip patch antenna

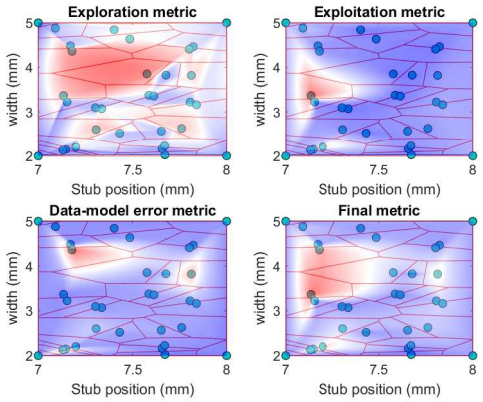


(a) First iteration

(b) Second iteration

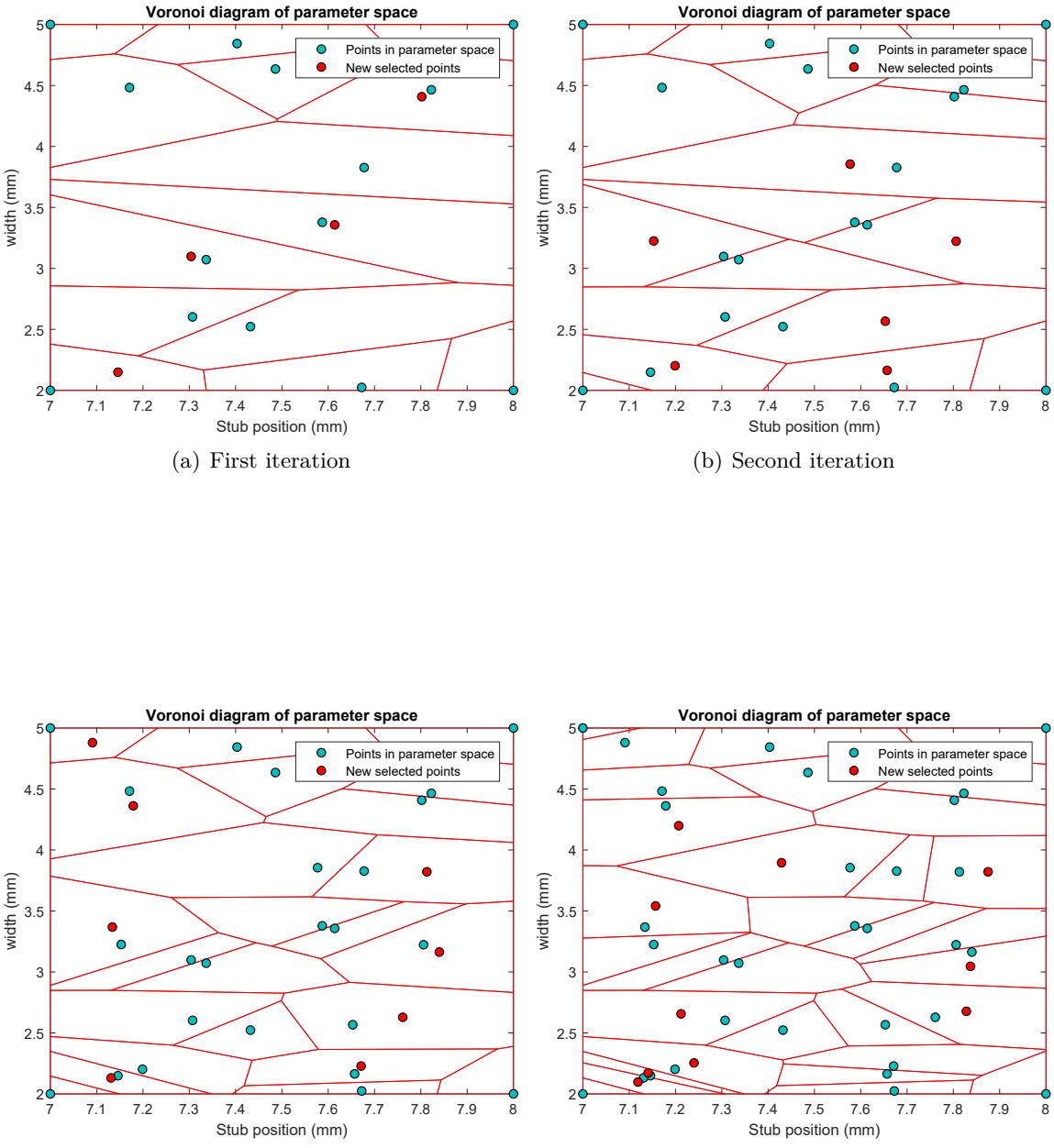


(c) Third iteration



(d) Fourth iteration

Figure 4.16: Plots of the metrics (single and combined) at each iteration, patch antenna case



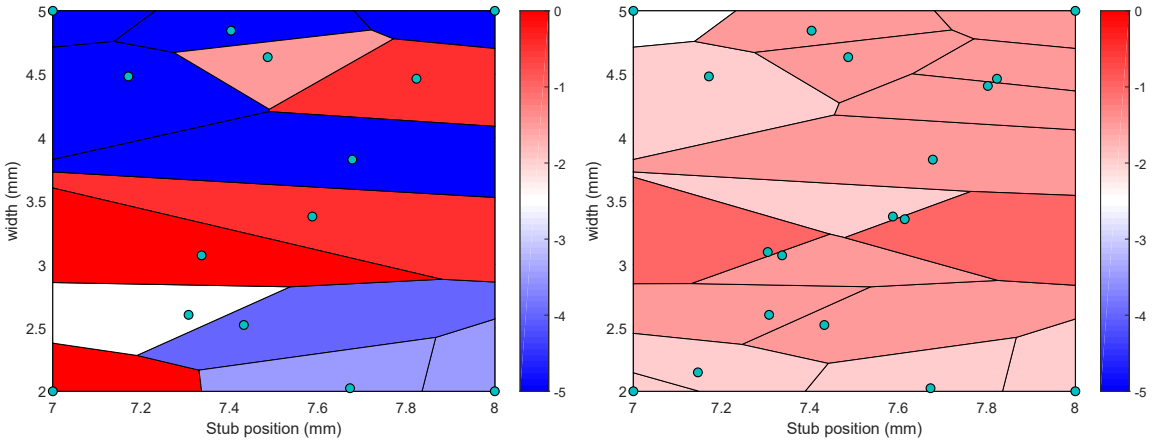
(a) First iteration

(b) Second iteration

(c) Third iteration

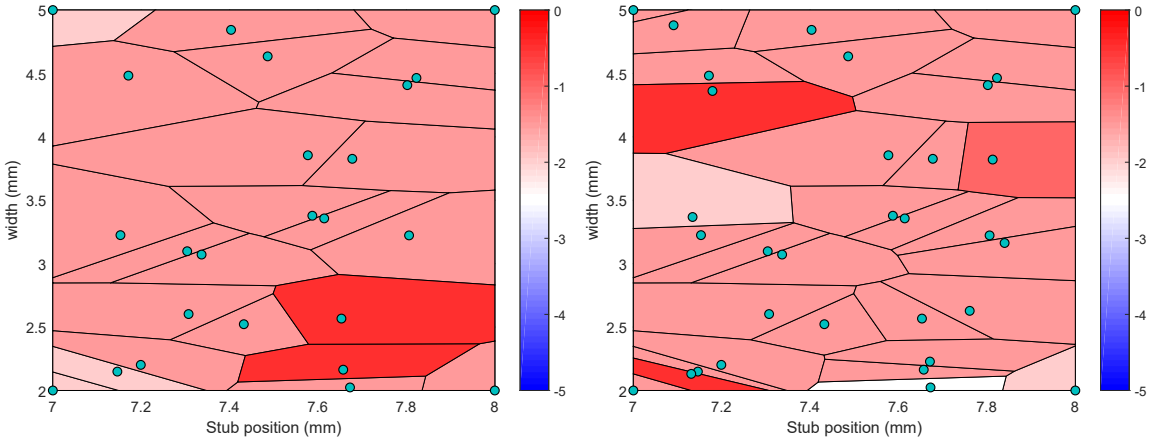
(d) Fourth iteration

Figure 4.17: Plots of the selection of new points at each iteration, patch antenna case



(a) First iteration

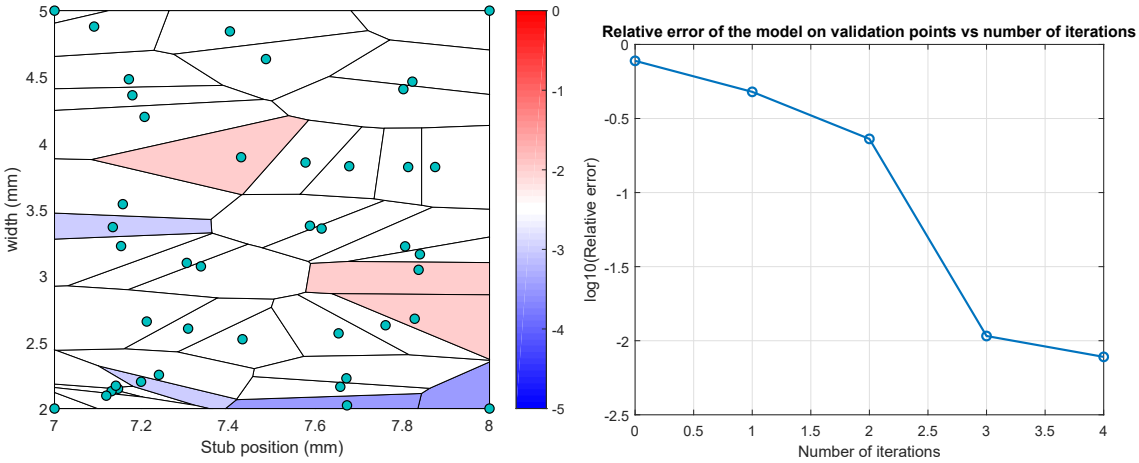
(b) Second iteration



(c) Third iteration

(d) Fourth iteration

Figure 4.18: Plots of the model error at each iteration, patch antenna case



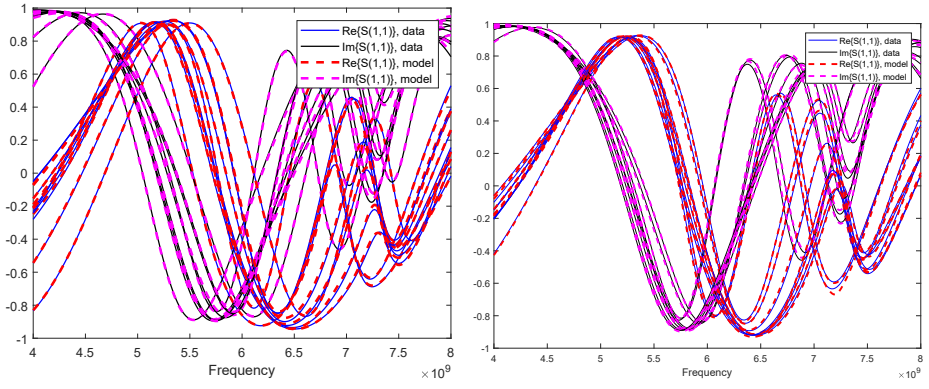
(a) Final model error

(b) Model error at each iteration

Figure 4.19: Final results, patch antenna case

Table 4.5: TABLE OF RESULTS FOR MICROSTRIP PATCH ANTENNA

Iteration	N. of points	Par.1 order	Par.2 order	Model error
1	14	2	2	0.7749
2	18	2	2	0.4785
3	24	2	2	0.1303
4	32	2	4	0.0107
5	42	3	4	0.0077



(a) Final model, selection of fitting responses

(b) Final model, validation responses

Figure 4.20: Final model

Chapter 5

EM Solvers

Beside the implementation of a class of adaptive algorithms, the other main goal of this thesis is the integration between commercial field solvers and the macromodeling tool. The final target is for sure a general integration with any type of EM solver, an ambitious yet unrealistic goal.
The integration of a solver, in fact, requires the knowledge (so the availability for the end user) of the API (Application Programming Interface) of the software, which is not always distributed by the producer. Moreover, the programming/scripting language used varies from tool to tool, increasing further the necessary knowledge/skills to perform this task.
The (chronologically) first part of this research work was devoted to the analysis of all EM solvers available, taking into considerations the features that would indicate among them the most suitable for this specific application. This type of analysis requires as a first step a direct comparison of the solvers, which here consists in the design and simulation of the same structure in all the environments, with a subsequent cross-validation of the results obtained.
This premise motivates the presence of this chapter, where a brief overview of EM solvers precedes a description of the validation process which has been conducted. Then, the rest of the chapter is devoted to the illustration of the steps necessary to obtain full integration of the solver with the MATLAB tool. Although this is clearly not scientifically relevant to the main topic of this work (namely, a class of adaptive algorithms for parameterized macromodeling), this is indeed the part of the process which required more time and the largest number of "transversal skills" to be developed. Furthermore, it led to a full framework automation, a key feature of the work presented in this thesis.

5.1 Classification of EM Solvers

Electromagnetic field solvers (EM solvers) are software tools devoted to solve directly Maxwell's equations, thus applying a first-principle numerical method. They are a subset of the wider field of EDA (Electronic Design Automation), and they are largely employed for the design of PCBs and Integrated Circuits. As explained in [24], field solvers can be inserted into the larger framework of Computational Electromagnetics (CEM), which can be defined as the technique that employs computers to solve electromagnetic problems,

obtaining as an output numerical results.
A possible classification of field solvers can be done by considering the type of numerical technique employed. As suggested by [25], there are two big groups of numerical methods for the solution of Maxwell's Equations:

- The first group solves Maxwell's Equations directly in their differential form, so it refers to their PDE (Partial Differential Equations) formulation. The most famous techniques belonging to this family are the **FDTD** (Finite-Difference Time-Domain) method and the **FEM** (Finite Element Method). In both cases the object under analysis and the surrounding space are divided into small elements (they could be segments, patches or cells), which interact with nearby units. The most favorable situation for the employment of this type of approach is when small 3D objects with many complex details have to be analysed. In particular, since the space under consideration must be discretized, this technique is more suited for closed problems, where the space is finite, while for open problems it would require the discretization of an infinite space. This problem is overcome thanks to the introduction of techniques like PML (Perfectly Matched Layers) and Absorbing Boundary Conditions.

- The second group solves Maxwell's Equations by dealing with their integral form. the problem is solved by dealing with the unknown currents flowing on the object under analysis. The integral equation which is used is formed by the solution to a point source, also known as Green's function. The peculiarity of this approach lies in the fact that the solution to the problem is valid not only on the object, but also in the entire surrounding space, without any type of approximation. The **Method of Moments** (MoM) is the most diffused method belonging to this family. In this case a subdivision of the analysed object is performed, but not of the surrounding space and, differently from PDE-based methods, each subdivision interacts with all other subdivisions. The equivalence theorem is then used to produce equivalent currents radiating in free space out to infinity.

5.2 Software tools

In the following, the software tools are presented and briefly described. The choice of the solvers was merely based on their availability, both through Academic License or in their Student Edition version.

- **Keysight ADS - Momentum**: Advanced Design System is an EDA software for the design of planar objects in RF, microwave and high speed digital applications.The integrated solver Momentum uses MoM formulated for planar structures. This represents a limitation for the design of full 3D objects.

- **Keysight EMPro**: Keysight EMPro is an electromagnetic simulation software for the analysis of 3D EM effects. The analysis can be carried out both in Frequency Domain, by means of FEM, or in Time Domain, though FDTD. What makes it particularly interesting for the specific application in mind is the possibility to use advanced scripting features (in Python language) to improve automation.

- **ANSYS Electronics Desktop**: The evolution of Sysoft HFSS is one of the most diffused tools in industry, thanks to both its flexibility and the wide variety of applications it is designed for. It is based on Finite Element Method.

- **CST Studio Suite**: CST Microwave Studio is one of the most complete software tools available. It employs the Finite Integration Technique (FIT). One of its peculiarities is the possibility to choose among Time Domain and Frequency Domain. Unfortunately, only the Student Version was available, and the limitations it poses on the maximum number of grid cells makes it really difficult to obtain results comparable to those achieved with the other software tools.

5.3 Validation of EM solvers

The validation process consists in modelling and simulating the same structure in all the available field solvers, comparing then the results obtained. This procedure is presented in the current section, where a simple stripline structure is analysed.

5.3.1 The structure

The structure taken into consideration is a simple object composed by a ground square metal plate, 20x20 mm in side, surmounted by a 0.035 mm thick copper trace. The trace is connected to the plate by means of two cylindrical vias, 0.13 mm in diameter and 0.4 mm in height, which in turn join the plate thanks to two 50 Ω resistors. The two resistors constitute the ports of the structure. A 3D representation of the structure is reported in 5.1.

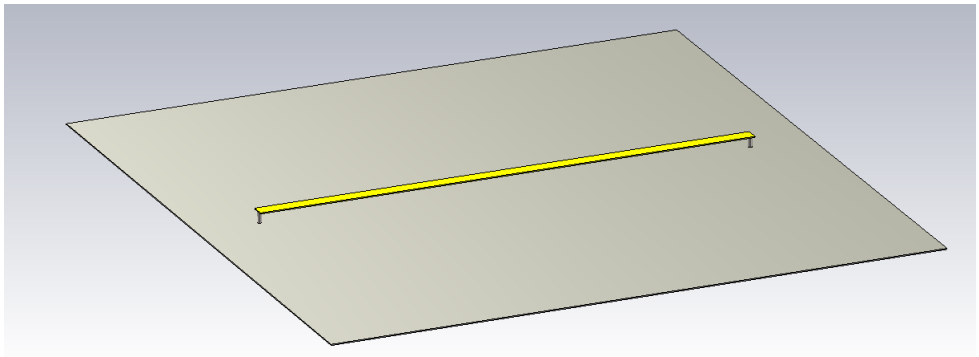


Figure 5.1: 3D representation of the microstrip structure

The simulations performed on the structure are of two types: the first is the measurement of **S parameters**, while the second is the measurement of **coupling voltage** in one of the ports under the excitation of an external **incident plane wave**.
The procedure to measure S parameters is done automatically by the solver, but it is necessary to identify the ports of the structure, which correspond to a voltage source in series with a 50 Ω resistor. The ports are excited one at a time, and each time the output

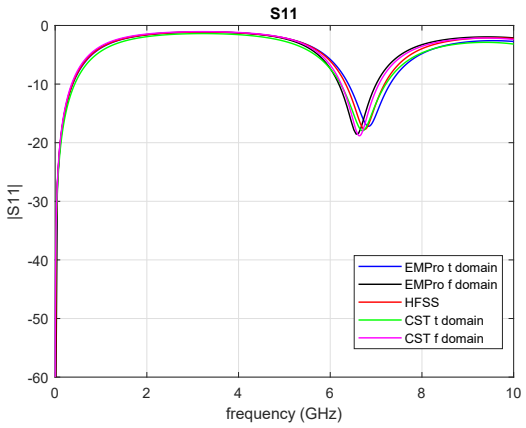


Figure 5.2: Comparison of S_{11} parameters

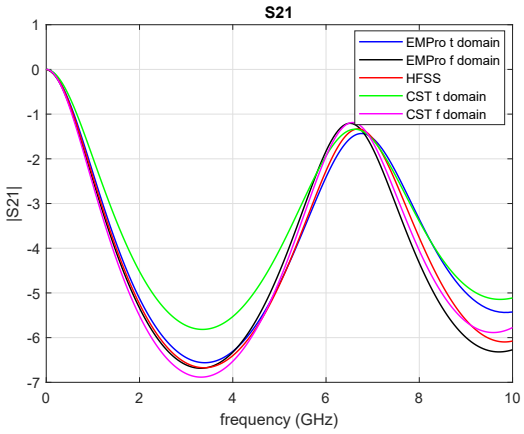


Figure 5.3: Comparison of S_{21} parameters

present at all the other ports is collected. For the software tools which offer both a time domain and a frequency domain solver, the solutions for both analyses have been included. For the second measurement, the structure is exposed to an incident plane wave, with incident plane normal to +z direction (**sidefire excitation**), and E-field vector (x=1, y=0, z=0). Then, the voltage across resistor R_1 is measured [26]. In order to compare the results obtained from CST and EMPro, the data of the voltages in time domain are collected and manipulated in Matlab: here the FFT is computed, making sure that the two sets of data have the same number of points. The results are reported in figures 5.4 and 5.5.

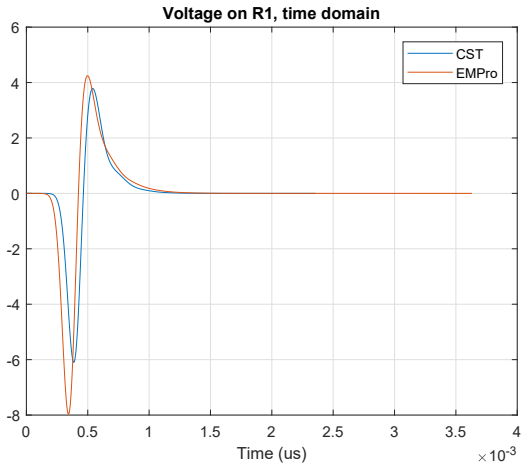


Figure 5.4: Voltage on R1, time domain, sidefire excitation

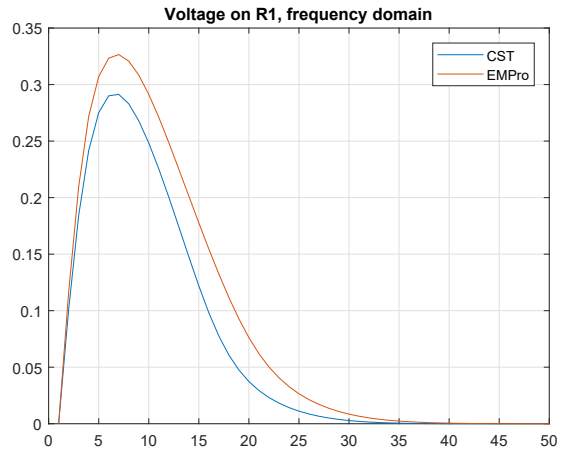


Figure 5.5: Voltage on R1, frequency domain, sidefire excitation

As a further analysis, the structure is subjected to a **broadside excitation** (wave incident in +y direction, E field directed toward -z direction).
Finally, a simulation with **endfire excitation** is performed, with propagation in -x direction and E field directed toward -z direction.

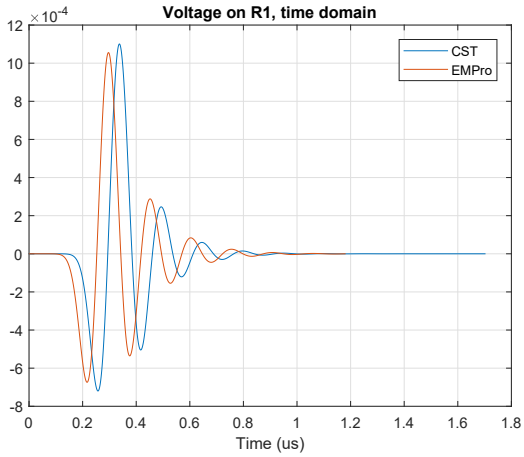


Figure 5.6: Voltage on R1, time domain, broadside excitation

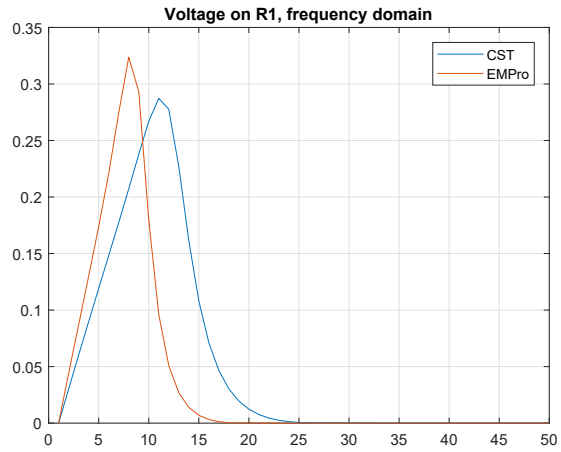


Figure 5.7: Voltage on R1, frequency domain, broadside excitation

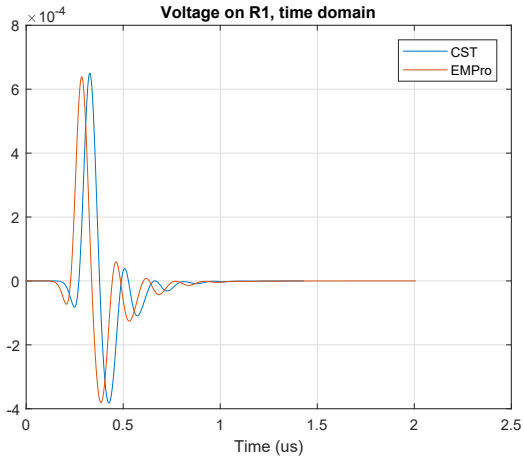


Figure 5.8: Voltage on R1, time domain, endfire excitation

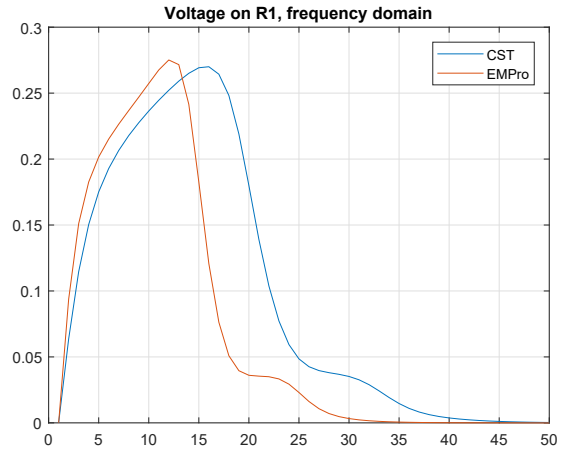


Figure 5.9: Voltage on R1, frequency domain, endfire excitation

After this analysis, it is possible to conclude that the results provided by the different solvers are in good accordance with each other. In all the analysed cases, there is not a perfect correspondence among the results provided by the different tools, but this is acceptable, as they employ different types of numerical techniques, which inevitably lead to different levels of accuracy.

5.4 Choice of the EM Solver

After the first approach with the different solvers thanks to the validation phase, the acquired knowledge on them is sufficient to make a wise choice on the best one for the intended purpose.
Due to the fact that CST was available only in the Student Version (with strong limitations

on a number of features, including mesh refinement) and to the difficulties encountered with ANSYS HFSS on some types of measurements, like those with excitation from an incident field, combined with the fact that a FEM solver is generally slower compared to those employing other techniques, the choice fell on Keysight ADS and EMPro.
The following sections present the workflow used for the integration of both EM solvers with the developed software tools. Before getting into the details, some clarifications are necessary. The two workflows developed appear to be really different one from the other, as are different the two software tools on which they are based. In particular, EMPro is generally simpler to use compared to ADS (which, on the contrary, spans a broader range of applications), it includes a 3D CAD tool (so more intuitive) and all documentation concerning its API is available to the final user. Moreover, the entire code behind EMPro has been written in Python.
On the other side, ADS is an extremely complete (and complicated) tool, written in a proprietary language called AEL (Application Extension Language), where the integration results to be much more difficult.

5.5 EMPro

As already stated at the beginning of the chapter, Electromagnetic Professional (EMPro) is a Keysight EEsof EDA's electromagnetic (EM) simulation software design platform for analyzing the 3D EM effects of components such as high-speed and RF IC packages, bondwires, antennas, on-chip and off-chip embedded passives and PCB interconnects [27].
An important feature in EMPro is Python scripting: the use of an object-oriented scripting language gives the possibility to create and simulate a project automatically, or to automatize a process that should be repeated manually a large number of times.
A Python script can be executed in the user interface of EMPro from the script editor. This allows to write scripts tailored for a particular project and run them inside the corresponding workspace. Another possibility is to use the commandline Python Interpreter, which allows to run Python scripts without accessing EMPro user interface. This can be done by running EMPro from *emproenv.bat* shell script [28].

5.5.1 EMPro Scripting

The approach used for managing EMPro integration with the MATLAB tool was a gradual automation of the necessary steps, namely:

- create a geometry;
- set up the solver;
- run the simulation;
- collect results.

Provided that the first phase, so creating the object geometry, is not intended to be automated, as it must be always done directly by the user through EMPro user interface, the remaining steps are instead expected to be done automatically by the developed framework.

Thus, the first scripting phase has been done with EMPro user interface, and it concentrates mainly on the last point, so the collection of simulation results.
The idea, in this case, is to collect the necessary results (voltage, current or S-parameters) and save them in a text file. These can be then read from MATLAB and, after the necessary manipulations, Touchstone files can be created. Saving results then requires just a few lines of code, devoted to identify the correct result and export it in the desired format.
Such Python code is:

```
#get the right result with dataset.getResult
voltage = empro.toolkit.dataset.getResult(sim=x, object='P', result='V');
time = voltage.dimension(0)
 out = file("filename.txt", 'w')
 for t, v in zip(time, voltage):
         out.write("%s\t%s\n" %(t, v))
 out.close()
```

The two remaining points, instead, can be developed by using the commandline Python interpreter. The final goal is to be able to simulate and collect results by running EMPro from shell script. This allows the MATLAB tool, through MATLAB command `system(command)` to run EMPro from command line and execute a Python script in charge of all aforementioned passages.
The Python script developed for this purpose is reported below:

```
from collections import OrderedDict
class LastUpdatedOrderedDict(OrderedDict):
    def __setitem__(self, key, value):
        if key in self:
            del self[key]
        OrderedDict.__setitem__(self, key, value)

def load_and_run_new_simulation(project_dir, engine=None, parameters=None):
    '''
    loads a project, creates a new simulation and runs it.
    '''
    load_project(project_dir)
    sim_id = create_new_simulation(engine, parameters)
    return run_simulation(sim_id, project_dir, parameters)


def load_project(project_dir):
    '''
    Load a project from disk as active project
    '''
    import os
    import empro
    print "-- loading project %s" % os.path.basename(project_dir)
    if not empro.activeProject.loadActiveProjectFrom(project_dir):
        raise RuntimeError("Failed to load project %r" % project_dir)
```

```
def create_new_simulation(engine=None, parameters=None):
    '''
    Creates a new simulation in the Project, without starting it.
    It assumes the project is correctly preconfigured to create new simulations.
    '''
    import os
    import empro
    from empro.toolkit.project import FEM, FDTD
    print "-- creating simulation"
    settings = empro.activeProject.createSimulationData()
    if engine:
        settings.engine = engine
    if parameters:
        set_parameters(parameters)
        settings.name = ", ".join("%s=%s" % p for p in parameters.items())
    sim = empro.activeProject.createSimulation(False)
    if settings.engine == FEM:
        # turn of mesher display.
        import pyfemPost as fem
        optionMgr = fem.getOptionManager(os.path.join(sim.simulationPath(),...
        ..."emds_dsn","design"))
        optionMgr.setValue("mesh", "showProgress", "off")
        optionMgr.write(2)
    sim_id = os.path.basename(sim.simulationPath())
    return sim_id


def run_simulation(sim_id, project_dir, parameters):
    '''
    Adds an existing simulation to the queue and wait for it.
    '''
    import os
    import empro
    from empro.toolkit import simulation
    print "-- running simulation"


    oldcwd = os.getcwd()
    os.chdir(os.path.join(os.environ['EMPROHOME'], os.pardir, os.pardir))
    try:
        sim = get_simulation_by_id(sim_id)
        sim.setQueued(True)
        empro.activeProject.simulations().isQueueHeld = False
        simulation.wait(sim)
        print sim.status
        if sim.status != 'Completed':
            raise RuntimeError("Simulation failed")
    finally:
        os.chdir(oldcwd)
```

```
    get_results(sim_id, project_dir, parameters)

def get_simulation_by_id(sim_id):
    '''
    looks up simulation from empro.activeProject.simulations by ID
    '''
    import os
    import empro
    for sim in empro.activeProject.simulations():
        if os.path.basename(sim.simulationPath()) == sim_id:
            return sim
    raise KeyError("No simulation with ID %r" % sim_id)


def set_parameters(parameters):
    '''
    Sets parameters of active project:
    set_parameters(abc='1 mm', def=2)
    '''
    import empro
    activeParams = empro.activeProject.parameters()
    for (name, formula) in parameters.items():
        print "%s=%s" % (name, formula)
        assert activeParams.contains(name), "Project does not...
        ... define parameter %r" % name
        activeParams.setFormula(name, formula)


def get_results(sim_id,project_dir, parameters):
    '''
    Retrieves results from current simulations;
    Saves results in .txt files in a specific folder inside the project
    Results currently saved are S parameters
    '''
    from empro.toolkit import dataset
    import empro
    parameters_name=[]
    parameters_formula=[]
    filename_s11 = "/s_param_results/s11"
    filename_s21 = "/s_param_results/s21"
    filename_s12 = "/s_param_results/s12"
    filename_s22 = "/s_param_results/s22"
    for (name, formula) in parameters.items():
        parameters_name.append(name)
        parameters_formula.append(formula)
    sim = get_simulation_by_id(sim_id)
    s11=dataset.getResult(sim, run=1, object='P1', result='SParameters')
    print"--saving results"
    frequency = s11.dimension(0)
```

```
    for i in range (0, len(parameters_name)):
        filename_s11 = filename_s11 + "_" + parameters_name[i] + "_" ...
        ... + parameters_formula[i].split()[0] + "_" ...
        + parameters_formula[i].split()[1]
        filename_s12 = filename_s12 + "_" + parameters_name[i] + "_" ...
        ...+ parameters_formula[i].split()[0] + "_" ...
        + parameters_formula[i].split()[1]
        filename_s21 = filename_s21 + "_" + parameters_name[i] + "_" ...
        ...+ parameters_formula[i].split()[0] + "_" ...
        + parameters_formula[i].split()[1]
        filename_s22 = filename_s22 + "_" + parameters_name[i] + "_" ...
        ...+ parameters_formula[i].split()[0] + "_" ...
        + parameters_formula[i].split()[1]

    out_s11 = file(project_dir + filename_s11 + ".txt", 'w')
    for f, s in zip(frequency, s11):
        out_s11.write("%s\t%s\t%s\n" %(f, s.real, s.imag))
    out_s11.close()

    s12=dataset.getResult(sim, run=2, object='P1', result='SParameters')
    out_s12 = file(project_dir + filename_s12 + ".txt", 'w')
    for f, s in zip(frequency, s12):
        out_s12.write("%s\t%s\t%s\n" %(f, s.real, s.imag))
    out_s12.close()

    s21=dataset.getResult(sim, run=1, object='P2', result='SParameters')
    out_s21 = file(project_dir + filename_s21 + ".txt", 'w')
    for f, s in zip(frequency, s21):
        out_s21.write("%s\t%s\t%s\n" %(f, s.real, s.imag))
    out_s21.close()

    s22=dataset.getResult(sim, run=2, object='P2', result='SParameters')
    out_s22 = file(project_dir + filename_s22 + ".txt", 'w')
    for f, s in zip(frequency, s22):
        out_s22.write("%s\t%s\t%s\n" %(f, s.real, s.imag))
    out_s22.close()



if __name__ == "__main__":
    from optparse import OptionParser
    parser = OptionParser(usage="%prog [options] projectdir ...")
    parser.add_option("", "--fdtd", action="store_const", dest="engine",...
    ... const="FDTD", help="use the FDTD engine")
    parser.add_option("", "--fem", action="store_const", dest="engine",...
    ... const="FEM", help="use the FEM engine")
    parser.add_option("-p", "--param", action="append", dest="params",...
    ... type="string", metavar="name=formula", help="design parameters,...
    ... specify as many as you like")
    options, args = parser.parse_args()
```

```
from empro.toolkit.project import FEM, FDTD
engine = {"FDTD": FDTD, "FEM": FEM, None:None}[options.engine]

params = LastUpdatedOrderedDict()
if options.params != None:
    for p in options.params:
        if '=' not in p:
            parser.error('-p, --param options must be of the form...
            ... name=formula: %s' % p)
        split_list = p.split('=',1)
        params[split_list[0]] = split_list[1]
for key,value in params.items():
    print 'key={} and value={}'.format(key,value)

for projectdir in args:
    load_and_run_new_simulation(projectdir, engine, params)
```

This script can be executed from commandline by typing:
`\path\empro2017\emproenv.bat python \path\run_and_save_results.py -p p1_name = "p1_value" -p p2_name = "p2_value" projectname`
It sufficient to specify, preceded by `-p`, all the parameters whose value must be set in the simulation, together with the corresponding value.
The final workflow, then is the following:

- The algorithm identifies the new parameter values to use for the next simulation;
- A MATLAB script writes a batch file (*temp.bat*) containing the commands necessary to run the Python script from commandline (the ones reported above) with the correct parameters values;
- through the command `system(temp.bat)` the batch file is run, and immediately after execution it is cancelled;
- MATLAB can access simulation results and carry on with the remaining operations.

5.6 ADS - Advanced Design System

Advanced Design System is an electronic design automation software system. It is targeted to designers of RF electronic products. With ADS it is possible to develop every step of the design process, from the schematic, to the layout, supporting optimization procedures and both frequency-domain and time-domain simulation. The electromagnetic simulation, instead, is done by an element in ADS called **Momentum**. Momentum is a 3-D planar EM simulation software: it solves Maxwell's equations using method of moments. Any type of batch simulation in ADS must go through some AEL scripting. In particular, there are two possibilities: the first is the use the Command Line option embedded in ADS, which allows to insert one AEL command at a time and execute it; the second is to write a complete AEL script (which then includes more AEL commands) and to save it inside the project workspace. In this way, everytime a simulation of the project is launched, the

script is executed as well.
For the purpose of this thesis, however, the situation is more complicated, since what must be done from Windows command prompt is a series of operations, including launching a Momentum simulation and changing the design of the object (to vary parameters).
In order to do so, some fundamental steps for the development of the workflow have been identified, and they are listed below.

- First of all, a project with the associated workspace must be created. This can be done through the GUI and does not need to be automated.
- A new schematic is generated, where schematic designs are created. At this stage the design is made by placing components, ports, data items, units, variables, equations, etc. Also this step is not automated.
- Starting from the schematic, the corresponding layout is generated. This step is fundamental because the electromagnetic simulation can be launched only once the layout has been created (it cannot run on the schematic). Moreover, any time something in the schematic is changed (for example a parameter value is modified), the layout needs to be updated. This can be achieved through a simple AEL command: `de_dse_s2l("", , , , 0, 0, , 0);`
- Once the layout has been generated/updated, it is necessary to write the input files for the electromagnetic simulation. This step can be done by using the AEL function: `dex_em_writeSimulationFiles(libName, cellName, emSetupViewName, simulationDirectory)`.
- At this point the Momentum simulation can be run, and it is possible to do it from the command prompt by means of the command `adsMomWrapper`.
- The last step of the workflow consists in saving the results of the EM simulation. The computed S-parameters are provided in the form of a .cti file, which can be converted in .snp format by means of a simple parser in MATLAB.

All the steps presented above (except for the creation of the workspace and the design of the schematic) must be executed in an automated way, without using the GUI.
The procedure, then, consists in:

- writing an .ael file from MATLAB. The model of this file is the following:

```
de_open_workspace("full_path_to_workspace");
decl designContext = de_get_design_context_from_name
("libName:cellName:schematic");
decl windowH = de_bring_context_to_top_or_open_new_window(designContext);
decl item=de_edit_item("VAR1");
de_set_item_parameters(item, list(list(prm_ex("ads_datacmps",
"VarFormStdForm",prm_ex("ads_datacmps","VarNameForm","w1"),
prm_ex("ads_datacmps","VarValueForm","500 um")),
prm_ex("ads_datacmps","VarFormStdForm",
prm_ex("ads_datacmps","VarNameForm","l2"),
prm_ex("ads_datacmps","VarValueForm","1343 um")))));
```

```
de_end_edit_item(item);
de_dse_s2l("", , , , 0, 0, , 0);
dex_em_writeSimulationFiles("libName","cellName","emSetup","simulation");
de_close_workspace_without_prompting();
de_close_all();
decl fd_out;
fd_out = fopen("full_path_to_workspace", "w");
fprintf(fd_out,"text\n");
fclose(fd_out);
```

The code provided allows for the execution of all the operations presented before. After the workspace has been opened, the corresponding schematic is selected and opened in a new window. At this point it is possible to vary some parameters, which must be declared in a "VAR" element inside the schematic, by means of the function `de_set_item_parameters`. Once the new values for the parameters have been set, the layout is updated, and the simulation input files are written, so that Momentum can be run.
The last lines of the code are devoted to closing all the windows containing the schematic and exiting from the workspace: this ensures that, when accessing the program next time, it will be possible to operate on the workspace again (otherwise the project would be opened as read-only);

- writing a .bat file, whose aim is to set the environment variable necessary for ADS to run batch, execute ADS executing the AEL file written previously, and finally run the Momentum simulation.
  The .bat file is the following:
  ```
  set hpeesof_dir=D:\Programs\Keysight
  set ads_license_file=licensefile.lic
  set path=%hpeesof_dir%\bin; %path%
  cd/d path_to_workspace
  del sem_lock.txt
  ads -m set_ads_for_simulations.ael
  :loop
  timeout 2 > null
  if not exist sem_lock.txt goto loop
  cd path_to_simulation_folder
  adsMomWrapperelisa -O -3D proj proj
  ```
- converting the output file in .cti format to .snp format.

Chapter 6

Further analyses and comments

The algorithm reported in Chapter 3 represents the initial version of the developed software tool. In this chapter it is used as the starting point for a number of analyses, whose aim is both to investigate the performances of some parts of the code, but hopefully to bring also some improvements to the initial code.
In particular, the analyses reported in this chapter refer to:

- the role that each metric plays in the choice of the new points, and a possible use of combined metrics;
- new methods for finding the position of the new points;
- a comparison with models built from a random distribution of points in space and a homogeneous distribution of points on a fine grid.

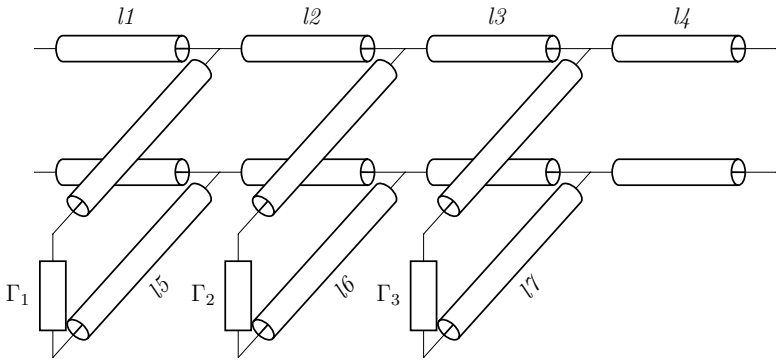


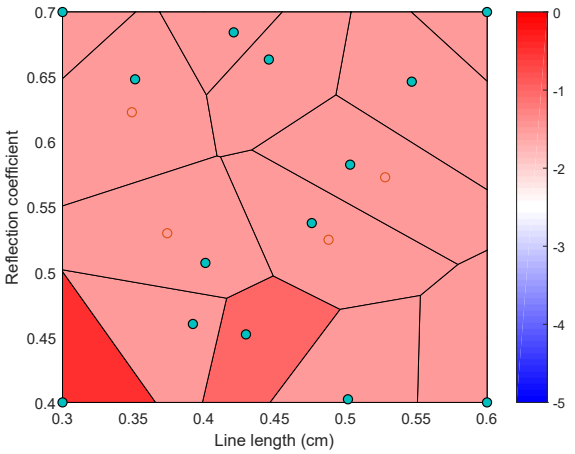
Figure 6.1: Schematic of transmission line filter

6.1 The individual roles of selection criteria

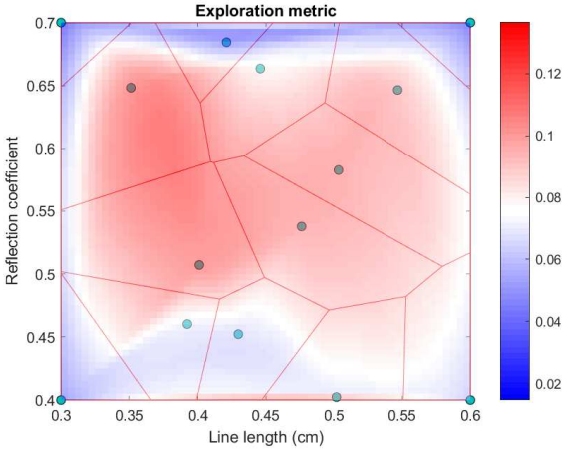
In order to show the importance of each selection criterion for the choice of new points, it is useful to analyse the individual role that each of them plays in such choice. To do so, each criterion has been "isolated", meaning that the algorithm has been run with each criterion as the sole metric upon which the choice of points was based. In this way, it is possible to see the different results to which they lead, demonstrating how a combined metric results to be the optimal selection rule.
For illustration purposes, we consider here as the structure to be modeled a Transmission Line filter, which will be analysed more in detail in Chapter 4, while here it is just used as an example for algorithm analysis and testing. The schematic of the filter is reported in 6.1. In this example the two design parameters are the length of the internal line segment l_2 and the value of the impedance connected to the central stub (the actual parameter is the reflection coefficient Γ_2, which varies as $\Gamma_2 = sin(\theta_2)$). The line length varies from 3 mm to 6 mm, while Γ_2 spans from 0.4 to 0.7.

6.1.1 Selection based only on exploration

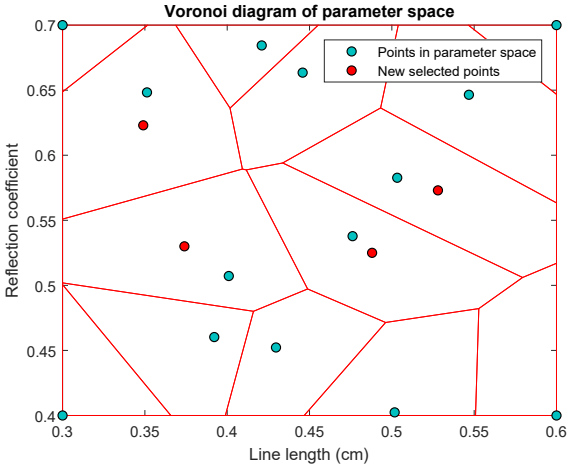
The first analysis is on the exploration metric alone: the choice for the next points will then be based solely on the distribution of present points, with the aim of filling the areas where less points are present. Figure 6.2(a) represents the relative error, in logarithmic scale, of the model constructed with the initial set of points (before adding any), composed of 10 points randomly scattered in the space and 4 points placed at the corners. This plot is important both because it represents the overall quality of the generated model, but also because the quality of the model is used as a termination criterion for the algorithm (once a sufficiently accurate model has been produced, the algorithm stops). In 6.2(b) the Voronoi tessellation of the parameter space is depicted together with a color metric indicating the value of Exploration criterion (from blue, for small cells, to red, for bigger cells). It is intuitive that the next points will be placed in the red areas, and this hypothesis is confirmed by 6.2(c), which indicates where the new points have been placed. The number of added points at each iteration has been fixed to $\frac{1}{3}$ of the already present points.
Once the new points have been added, a new model is generated, and the relative error is depicted in 6.3(a). It is possible to see, when compared to 6.2(a), that the model is more accurate, as the majority of cells are not red anymore.



(a) Model error, log scale



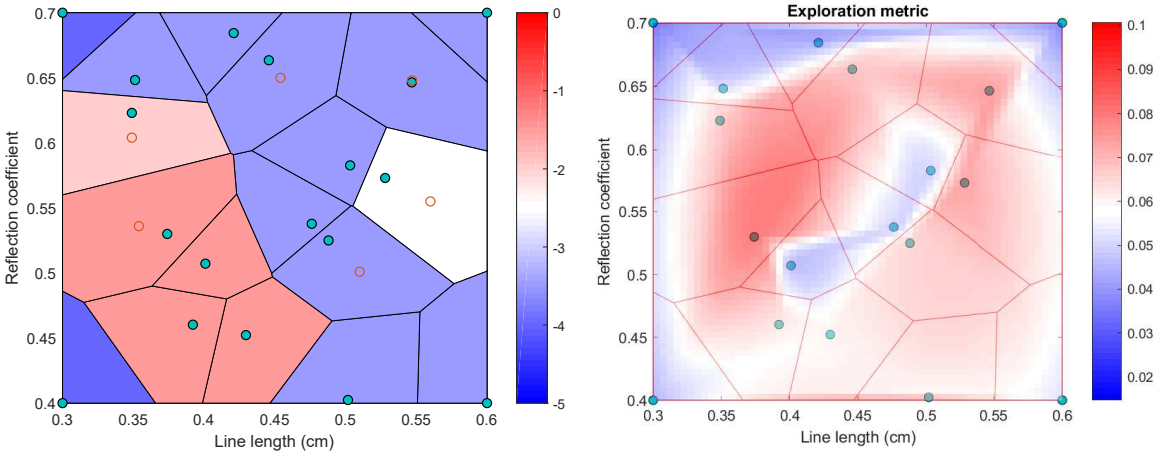
(b) Exploration metric



(c) Selection of new points

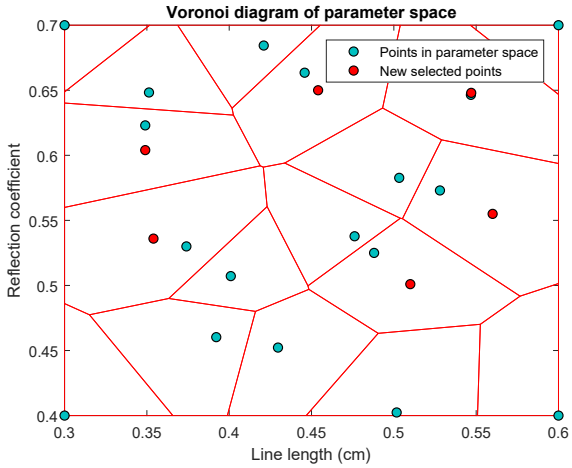
Figure 6.2: Selection based on exploration, first iteration.

The exploration metric is again evaluated on this new tessellation, giving as a result the plot in 6.3(b). This leads to the identification of new points represented in 6.3(c).



(a) Model error, log scale

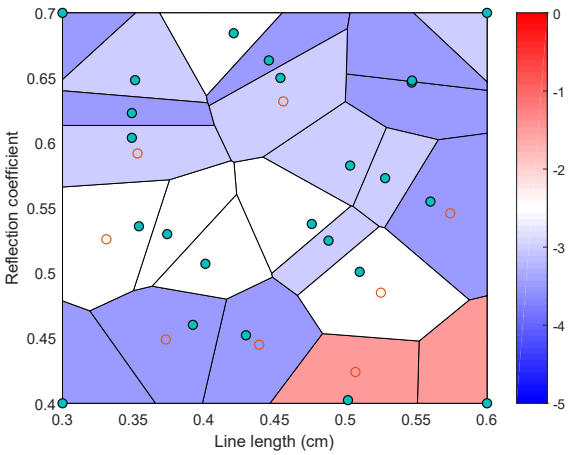
(b) Exploration metric



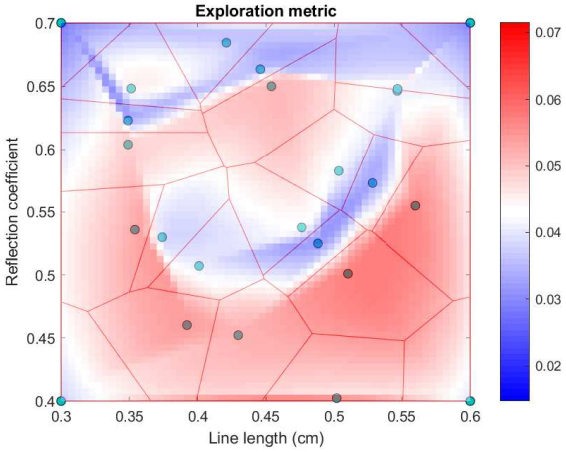
(c) Selection of new points

Figure 6.3: Selection based on exploration, second iteration

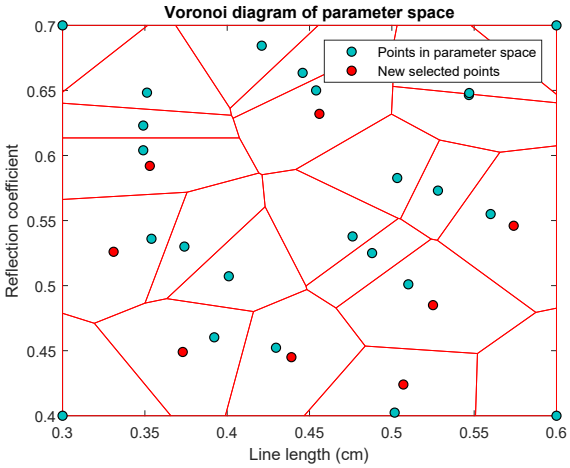
The new model for the new set of points is generated, and the number of red cells has decreased, as reported in figure 6.4(a). Once again the metric is evaluated (6.4(b)) and the new points identified (6.4(c)).



(a) Model error



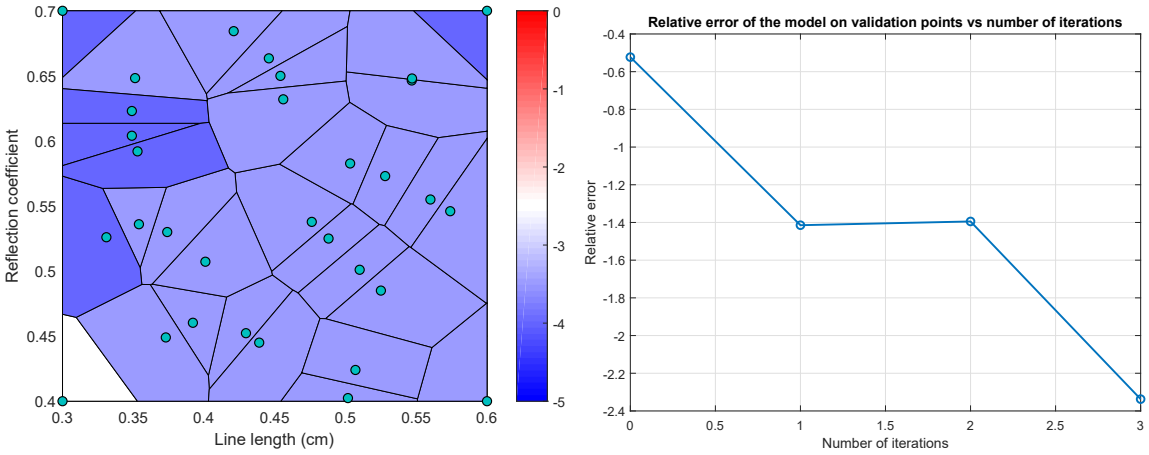
(b) Exploration metric



(c) Selection of new points

Figure 6.4: Selection based on exploration, third iteration

Now the parameter space contains 32 points, and the generated model, as reported if 6.5(a) is extremely accurate, so the algorithm is stopped.



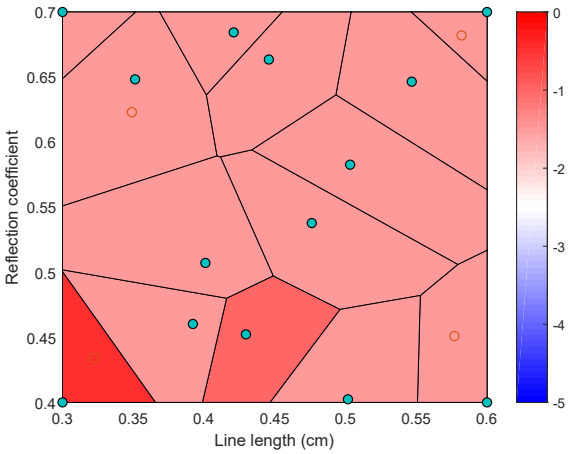
(a) Model error

(b) Data-model error vs iteration, log scale

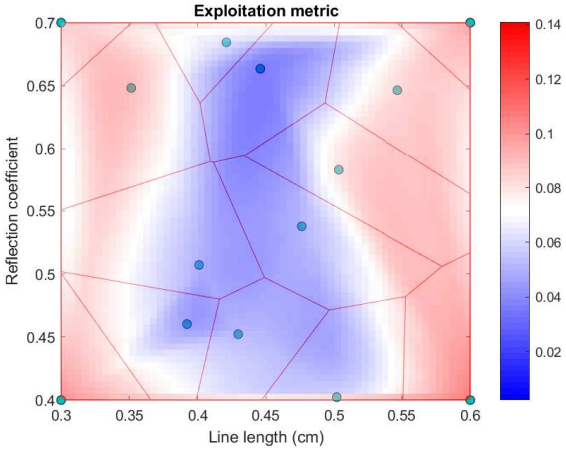
Figure 6.5: Selection based on exploration, final results

6.1.2 Selection based only on exploitation

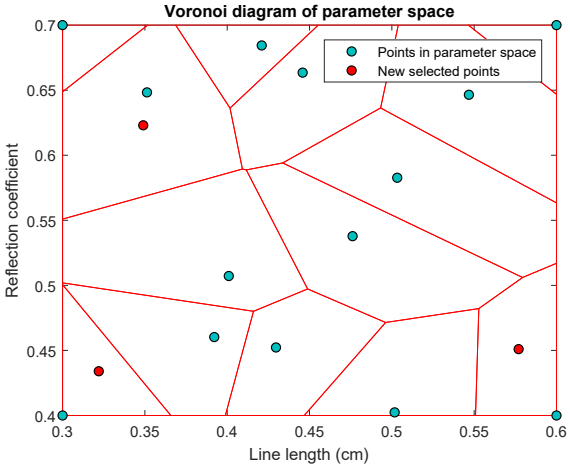
The same analysis is carried out, but this time considering only exploitation metric, so identifying the regions with a larger variability of data. The initial random distribution of points is always the same as in the exploration analysis, so that results can be comparable. For this reason, the first model built with the initial 14 points, reported in 6.6(a), is the same as in the exploration case (see 6.2(a)).
The exploitation metric is evaluated on this initial distribution of points, and the results are depicted in 6.6(b). It is worth noticing that this metric, in this example, results to be opposite to the exploration one, as the "hottest" regions tend to be on the edges and in the center of the design space, leading to a completely different choice of points. The choice of new points in the first iteration is reported in 6.6(c). The new model is constructed (6.7(a)) and, since the final error is not small enough, the exploitation metric is once again evaluated (6.7(b)) and new points selected (6.7(c)).



(a) Model error

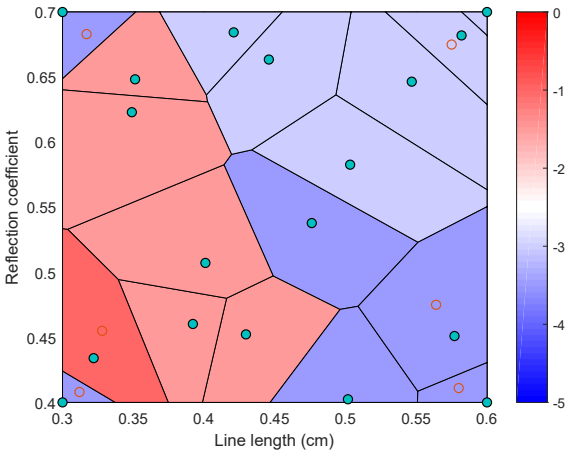


(b) Exploitation metric

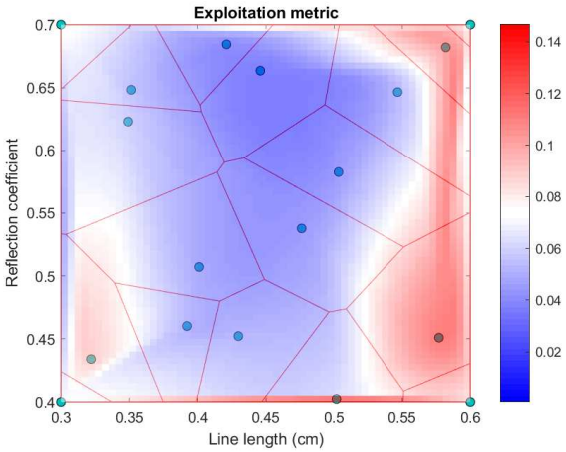


(c) Selection of new points

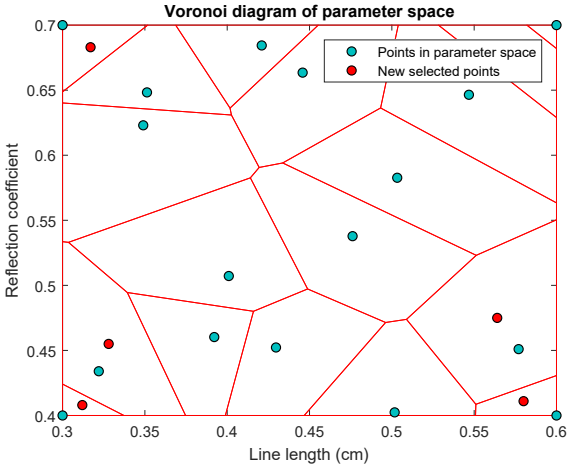
Figure 6.6: Selection based on exploitation, first iteration



(a) Model error



(b) Exploitation metric

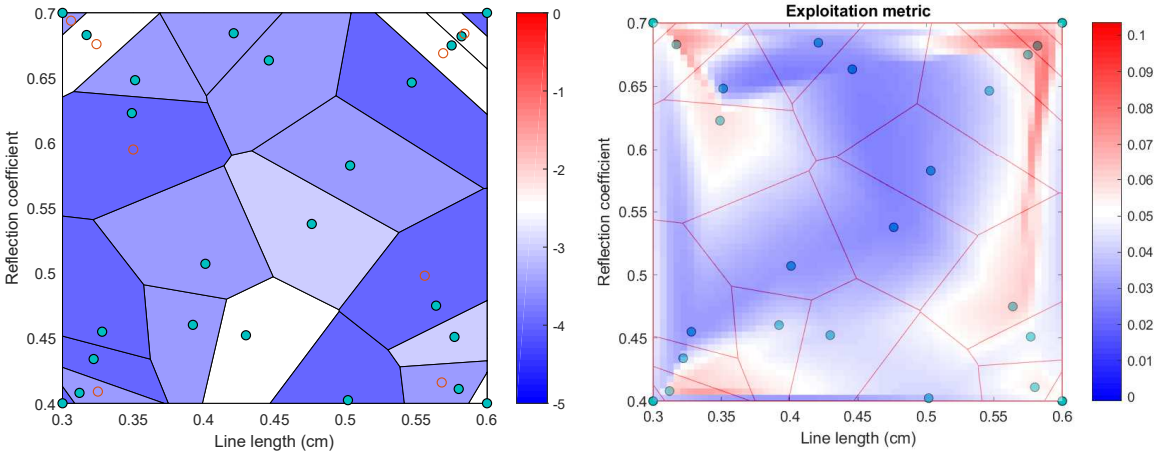


(c) Selection of new points

Figure 6.7: Selection based on exploitation, second iteration

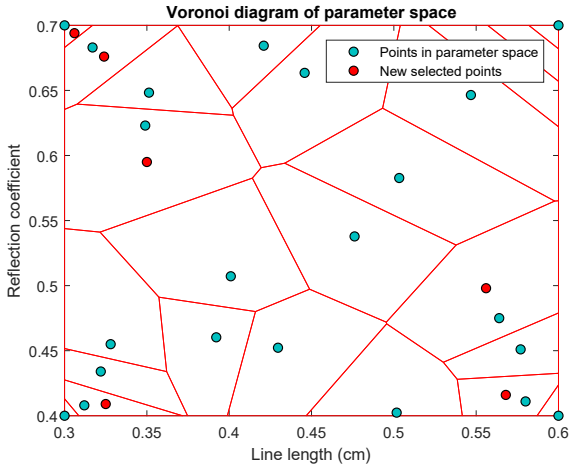
The algorithm is repeated again for another iteration (6.8(b), 6.8(c)) until a sufficiently

accurate model is achieved (6.9(a)).



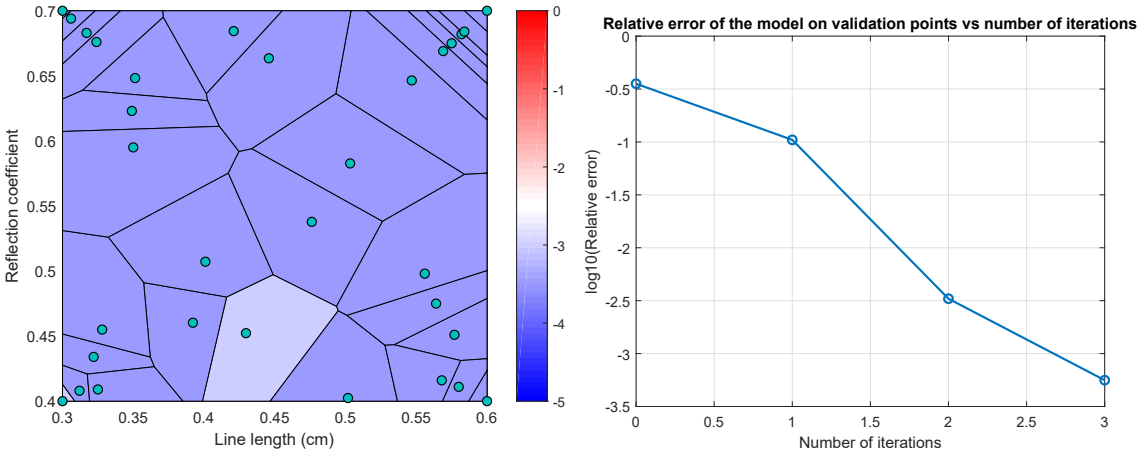
(a) Model error

(b) Exploitation metric



(c) Selection of new points

Figure 6.8: Selection based on exploitation, third iteration



(a) Model error

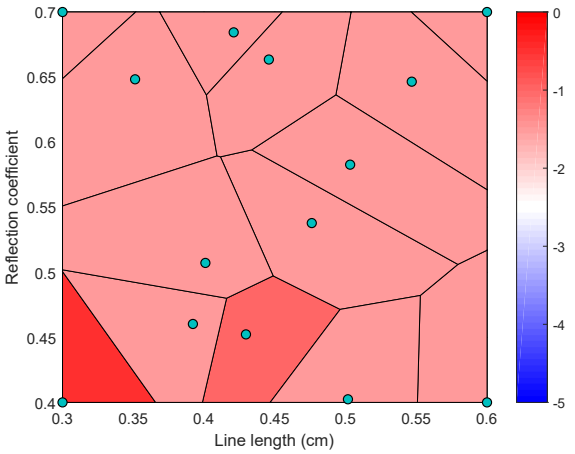
(b) Data-model error vs iteration, log scale

Figure 6.9: Selection based on exploitation, final results

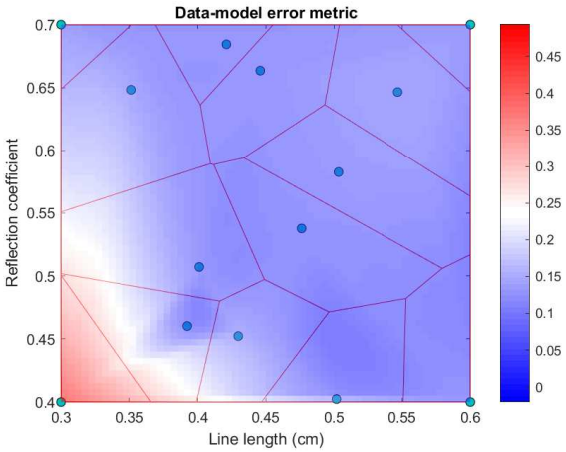
A first comment can be already made by looking at the results presented up to now. It turns out, in fact, that for the same total number of data points, the exploitation criterion leads to a more precise model than the one based on the exploration one (compare 6.9(b) to 6.5(b)). Thus, the exploration criterion cannot be considered, by itself, an optimal selection metric.

6.1.3 Selection based only on data-model error

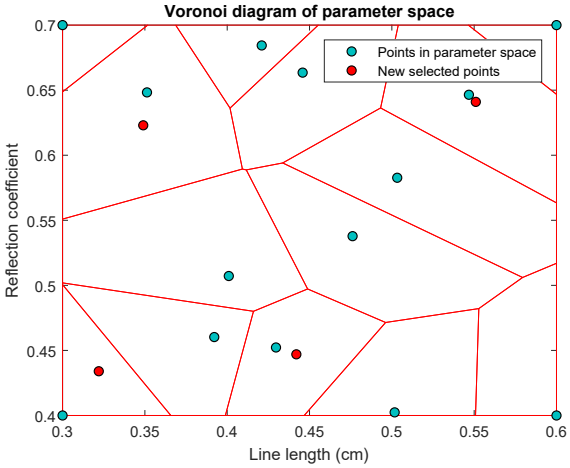
The last metric yet to be analysed is the data-model relative error metric, which places the new points where the model results to be less accurate. The workflow is the same as in the previous cases, so we present only the results without discussion in 6.10, 6.11, 6.12, 6.13. It is, however, worth noticing how the final distribution of data points is extremely different from those obtained with the other two metrics. If we consider 6.13(a), in fact, it is clear how some regions have a concentration of points much higher than the others, leading to a final tessellation of space which is highly non homogeneous.



(a) Model error

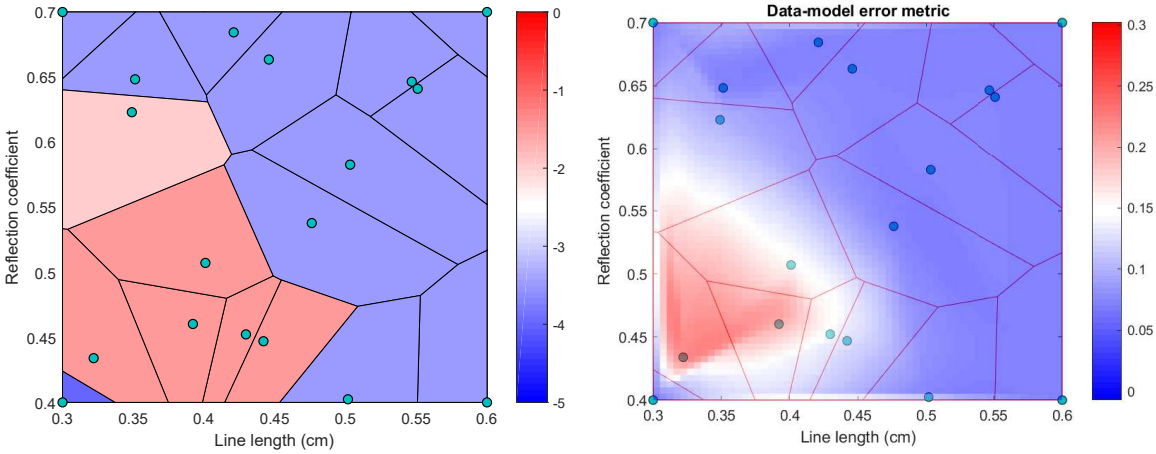


(b) Data-model error metric



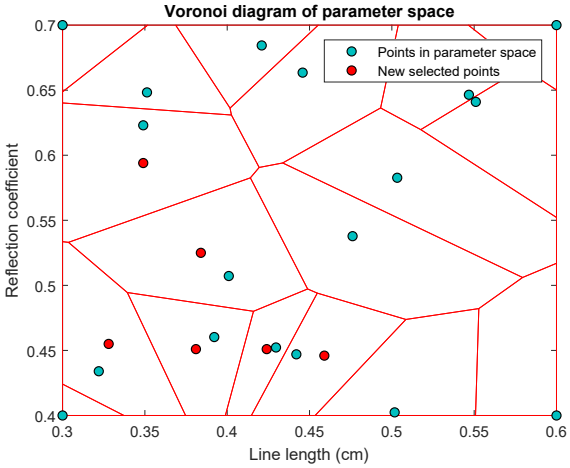
(c) Selection of new points

Figure 6.10: Selection based on model error, first iteration



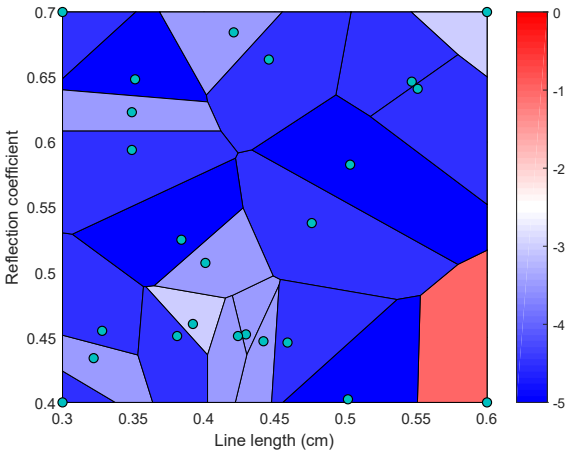
(a) Model error

(b) Data-model error metric

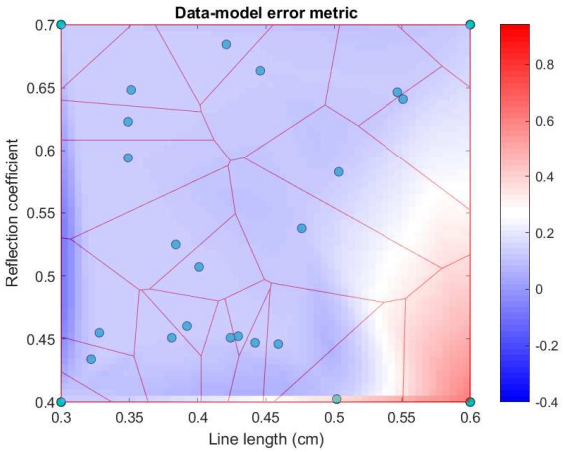


(c) Selection of new points

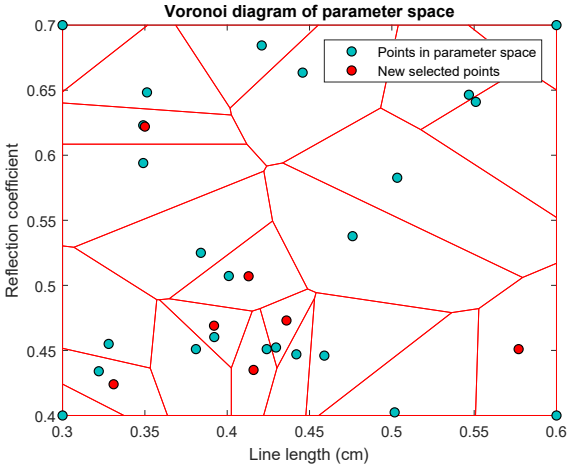
Figure 6.11: Selection based on model error, second iteration



(a) Model error

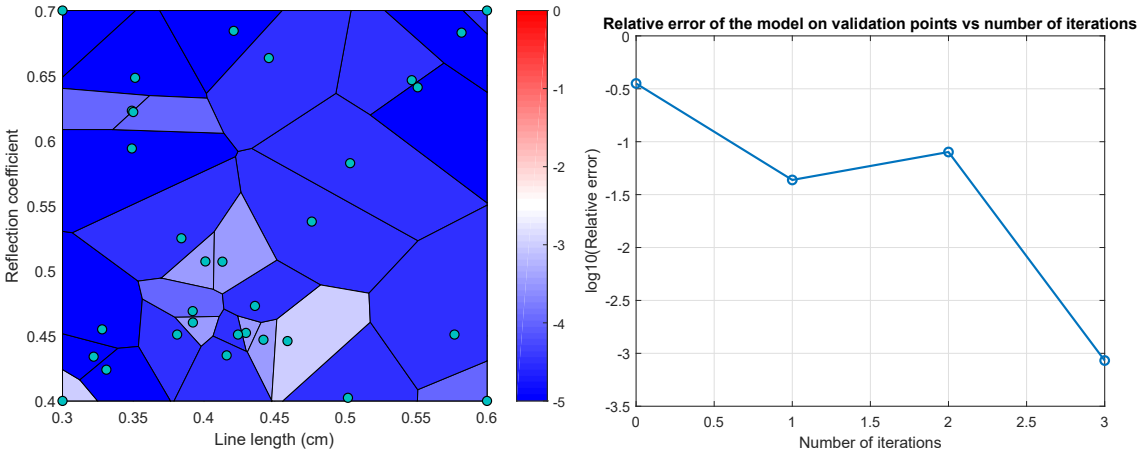


(b) Data-model error metric



(c) Selection of new points

Figure 6.12: Selection based on model error, third iteration



(a) Model error

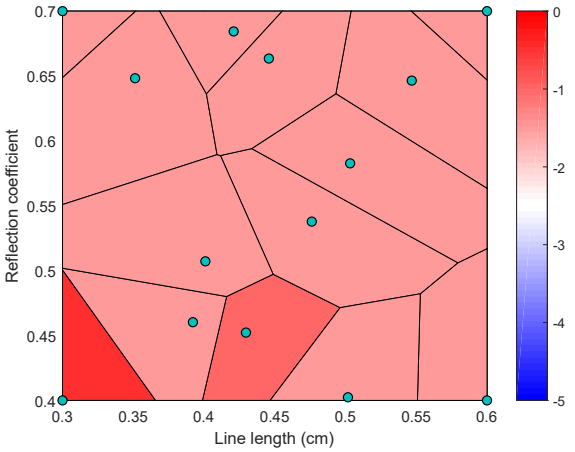
(b) Data-model error vs iteration, log scale

Figure 6.13: Selection based on data-model error, final results

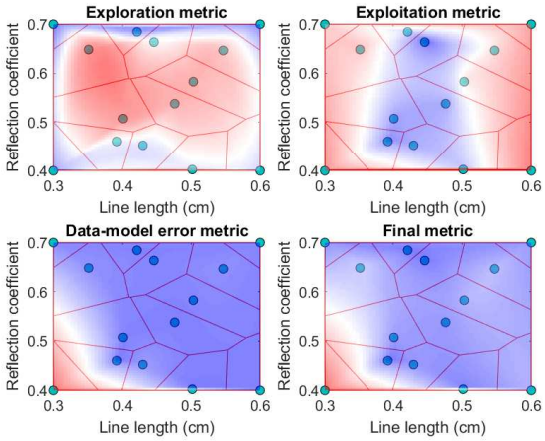
Before going to the next section, a comment on the last results is necessary. The accuracy obtained in the final model, if using model error as the only metric, is clearly really high, especially when compared to the results provided by the two other metrics alone. However, in this last case the effectiveness of the algorithm is assessed by using the same metric adopted for the choice of new points (the model error), thus the final result will for sure be optimal in this sense. However, even if model error is a good indicator of the results that the algorithm can provide, it completely overlooks some aspects that are indeed important for evaluating the optimality of the chosen points (how they are evenly distributed on the design space, for example). So, even if at a first glance the data-model error metric seems to be far superior than the other two, it cannot be used as the only and final metric, as it is still complementary to the other selection criteria.

6.1.4 Selection based on combined metric

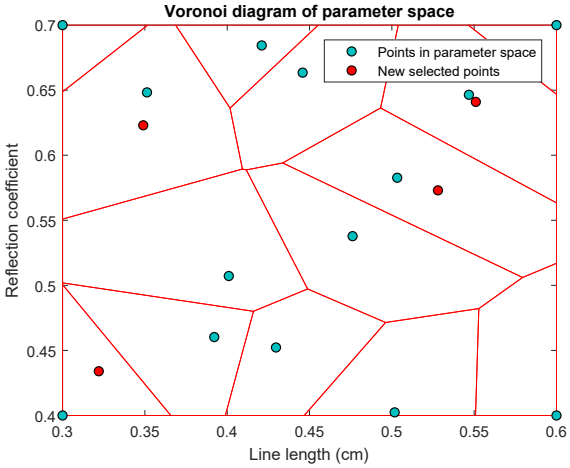
The last step consists in analysing the behaviour of the algorithm when the final metric is used, so when a combination of the previously analysed selection criteria is employed. After all the plots for the various iterations are provided, some final comments are in order. In 6.14 the graphs concerning the first iteration of the algorithm are provided. In 6.14(a) the model error on the initial random distribution of points is depicted, then in 6.14(b) the results on all the metrics (three individual plus the final one) are reported, leading to the selection of the new points in 6.14(c).
The same happens in 6.15, this time with the results of the second iteration, and in 6.16 for the third iteration. The final model error is reported in 6.17(a), together with a comparison among metrics in 6.17(b), from which it is possible to conclude that the combined metric is the best solution for the selection of new points, as it provides the most accurate model. A representation of the model responses, compared to the original data, is provided in 6.18, plotting separately a selection of responses used for fitting and another used for validation. It is possible to see how the accuracy obtained is high in both cases.



(a) Model error

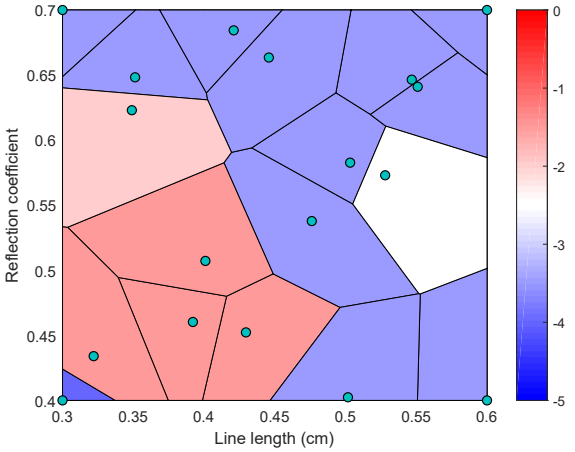


(b) Metrics, first iteration

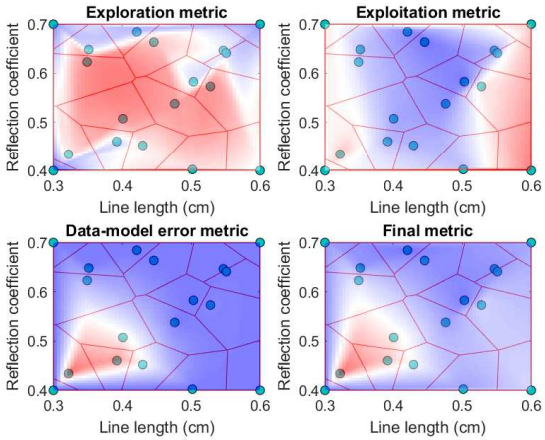


(c) Selection of new points

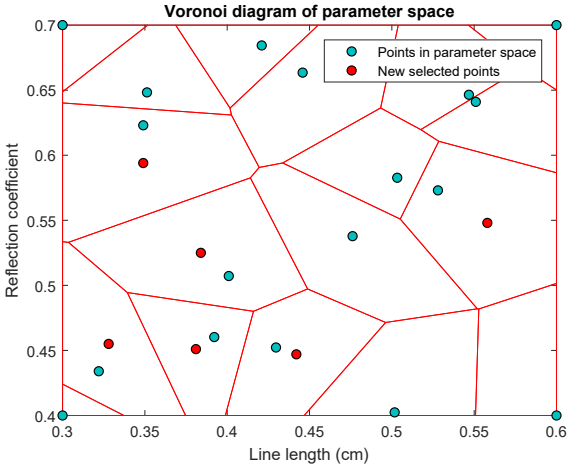
Figure 6.14: Selection based on combined metric, first iteration



(a) Model error

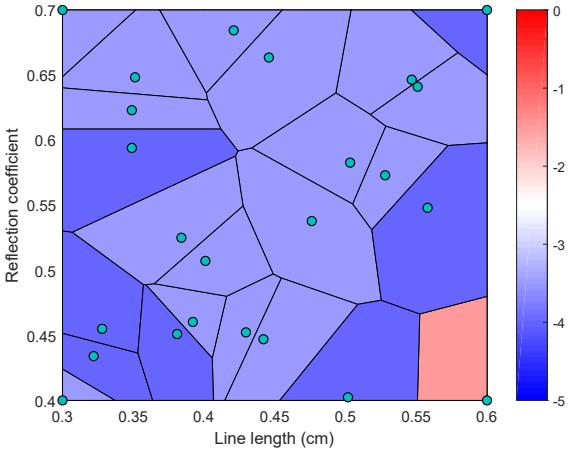


(b) Metrics

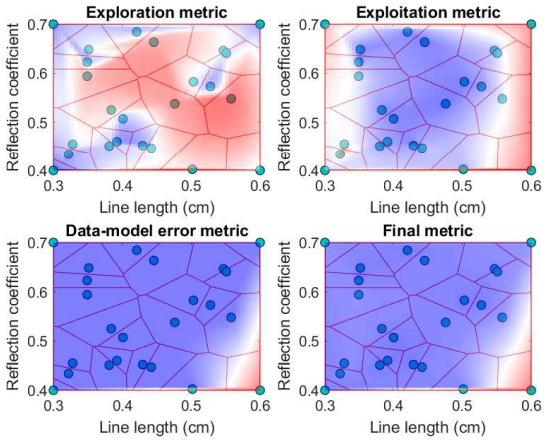


(c) Selection of new points

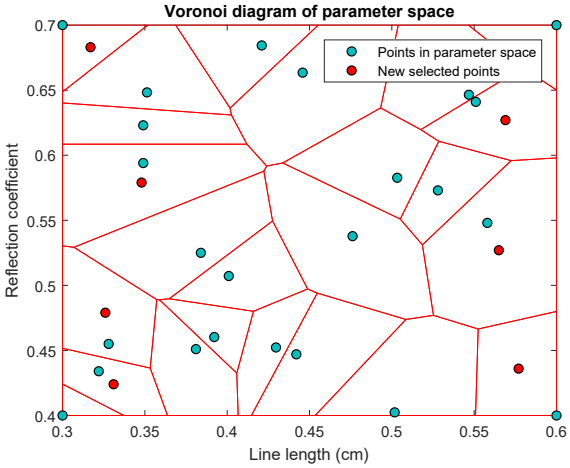
Figure 6.15: Selection based on combined metric, second iteration



(a) Model error

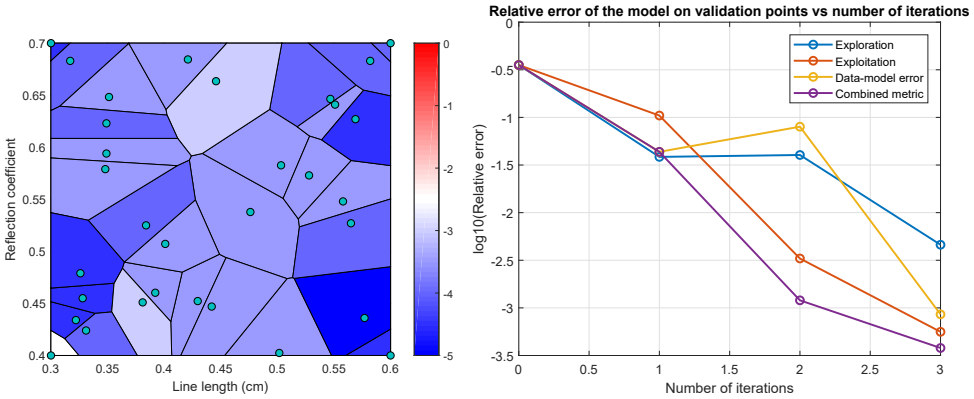


(b) Metrics



(c) Selection of new points

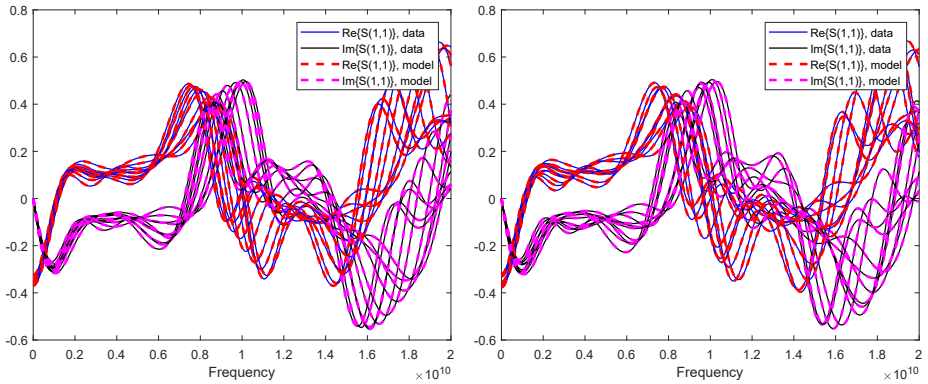
Figure 6.16: Selection based on combined metric, third iteration



(a) Model error

(b) Comparison among metrics

Figure 6.17: Selection based on combined metric, final results



(a) Final model, selection of fitting responses

(b) Final model, validation responses

Figure 6.18: Final model

As it is clear from 6.17(b), the combined metric is the one providing, in the end, the best results compared to the use of single metrics. This is due to the optimality of the selected points, which satisfy multiple criteria. The final distribution of points, thus, will be more dense where the model tends to be less accurate, or where data show higher variability, but the presence of the exploration metric still guarantees an homogeneous distribution on the design space.

For the sake of completeness, the analysis has been repeated on the other examples as well, and the results are reported in the tables below.

For the case of the filter with parameterized stub length and load, the model errors at each iteration considering each metric separately is reported in table 6.1.

Table 6.1: TABLE OF RESULTS FOR FILTER STUB-LOAD

Iteration	N. of points	Exploration	Exploitation	Model error
1	14	0.0445	0.0445	0.0445
2	18	0.0013	0.0013	0.0011
3	24	3.12e-4	6.873e-4	1.89e-4
4	32	2.69e-4	3.068e-4	1.75e-4

For the case of the filter with parameterized stub length and line length, instead, the results are reported in table 6.2.

Table 6.2: TABLE OF RESULTS FOR FILTER STUB-LINE

Iteration	N. of points	Exploration	Exploitation	Model error
1	14	0.4743	0.4743	0.4743
2	18	0.0309	0.0974	0.0235
3	24	0.0440	0.0396	0.0183
4	32	0.0161	0.0353	0.0048

These results simply confirms the conclusions already reported previously: among the three metrics, the one based on model error is the metric leading to the most accurate model. The exploitation leads to similar results, as intuitively the model tends to be less accurate where the variability of data is higher. The exploration metric, on the other hand, results to be the least significant among the three.

6.2 A new metric for cell ranking: use of individual weights

Another possibility which has been taken into consideration is the use of a different combined metric, which associates to each single selection criterion a specific weight, as reported in the equation below:

$$G(\boldsymbol{p}_k) = (1 + \alpha A(\boldsymbol{p}_k)) \cdot (1 + \beta E(\boldsymbol{p}_k)) \cdot (1 + \gamma Err(\boldsymbol{p}_k)) \tag{6.1}$$

It is important to notice that, if weights are used, a new normalization of metrics must be employed. Up to now, in fact, $A(\boldsymbol{p}_k)$, $E(\boldsymbol{p}_k)$ and $Err(\boldsymbol{p}_k)$ have been normalized with respect to the sum of the metric values in all the points. This means that a sort of weight is already "embedded" in the metric, giving more importance to those selection criteria which assume very different values among data points. If we consider Exploration, for example, the areas of the cells tend to have very similar values, so in the final metric the Exploration metric will be weighted less.
If instead explicit weights must be introduced, this "implicit weight" needs to be removed,

and the solution is to use a different normalization, one which normalizes metrics between 0 and 1. In this way all the metrics are comparable.
Since the analyses on single selection criteria have showed how Exploitation and Model Error are more important, the values used for the single weights are $\alpha = 1$, $\beta = 5$, $\gamma = 5$.
The tests have been carried out on two examples, and the results are reported in the tables below.

Table 6.3: TABLE OF RESULTS FOR FILTER STUB-LOAD

Old metric	**New metric**
0.0445	0.0445
0.0014	0.0013
0.0039	0.0091
0.00053	0.00025

Table 6.4: TABLE OF RESULTS FOR FILTER STUB-LINE

Old metric	**New metric**
0.4743	0.4743
0.0839	0.2234
0.0484	0.0289
0.0069	0.0112
0.0057	0.0025

As reported in 6.3 and 6.4, the results in the two cases are comparable: the old metric seems to be more precise in the intermediate iterations, while the new metric leads to a final more accurate in model.
In the end, the use of a weighted metric does not provide a significant improvement to the algorithm.

6.3 Finding the position of new points

In the previous chapters, once the cell with the highest ranking is identified, the new point inside it is placed at the center of such cell. This is an arbitrary choice, thus some alternative rules for the positioning of the new point have been identified and analysed. In particular, the new point can be placed:

- at the centroid of the polygon formed by the neighboring points (**first criterion**);
- the furthest from the neighboring points (L^2-norm) (**second criterion**);
- the furthest from the neighboring points (infinity norm) (**third criterion**);

- the furthest from the neighboring points, including the point already present in the cell (**fourth criterion**);
- at the center of the cell, as already done previously (**fifth criterion**).

The first criterion The first criterion, as stated above, consists in placing the new point at the center of the polygon formed by the neighboring points.
The centroid of a polygon defined by n vertices (x_0,y_0), (x_1,y_1), ..., (x_{n-1},y_{n-1}) is the point (C_x, C_y) defined as

$$C_x = \frac{1}{6A}\sum_{i=0}^{n-1}(x_i + x_{i+1})(x_i y_{i+1} - x_{i+1}y_i) \tag{6.2}$$

$$C_y = \frac{1}{6A}\sum_{i=0}^{n-1}(y_i + y_{i+1})(x_i y_{i+1} - x_{i+1}y_i) \tag{6.3}$$

where A is the polygon's signed area, defined as

$$A = \frac{1}{2}\sum_{i=0}^{n-1}(x_i y_{i+1} - x_{i+1}y_i) \tag{6.4}$$

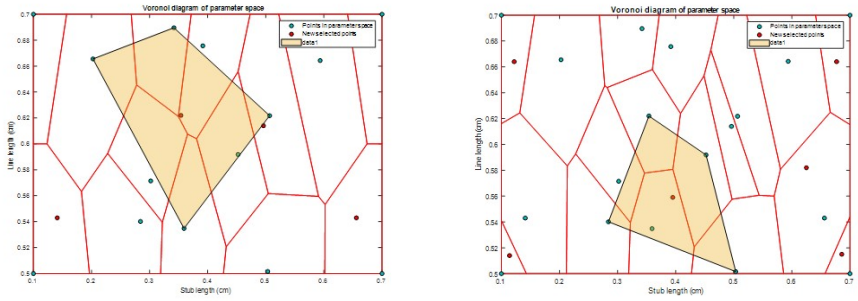
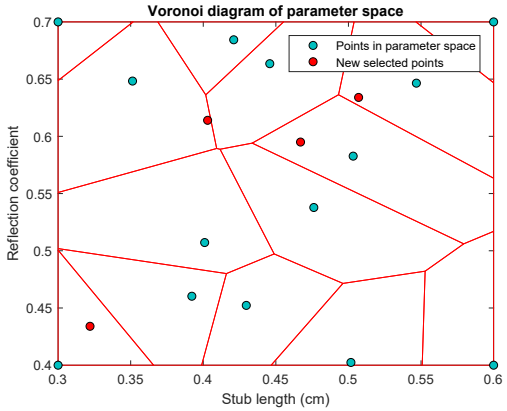


Figure 6.19: Examples of first criterion

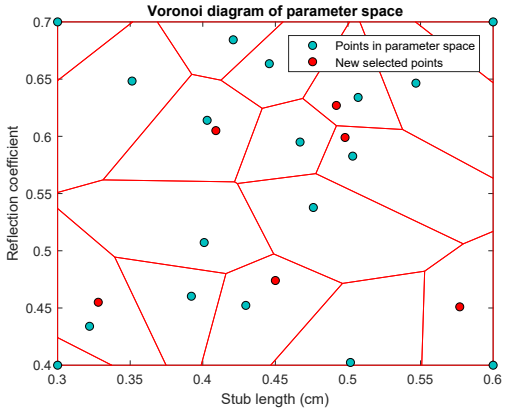
The second criterion The second criterion consists in placing the new point the furthest from the neighbouring points. This is obtained with a Monte Carlo approach, by selecting a sufficient number of points with random position inside the cell, computing the distance of each of these points from the neighbours, and finally selecting the one with the highest distance (defined as the Euclidean distance, so the L^2 norm).

6.3.1 Comparison among different criteria

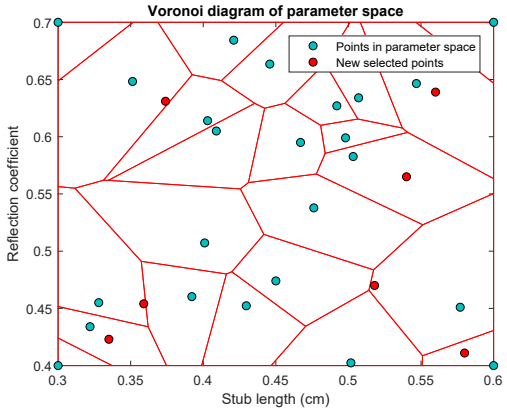
A complete analysis has been conducted on the results that each selection criterion provides, both in terms of distribution of points in the parameters space and final accuracy of the model.



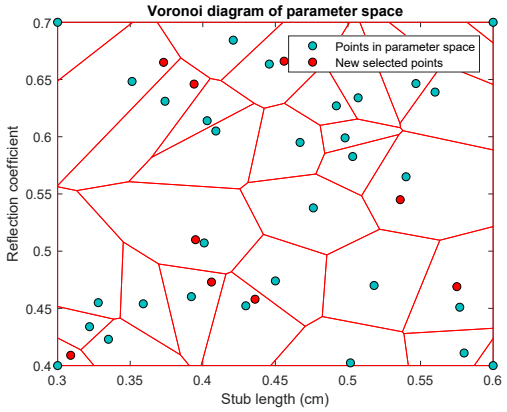
(a) First iteration



(b) Second iteration

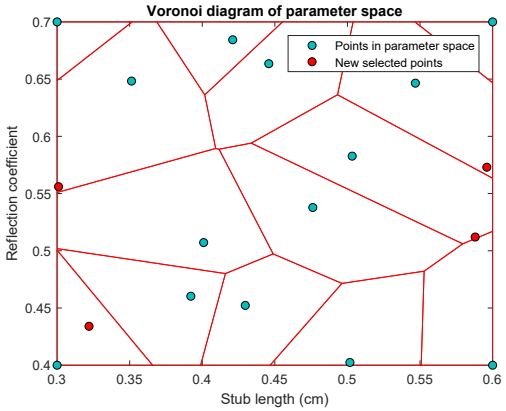


(c) Third iteration

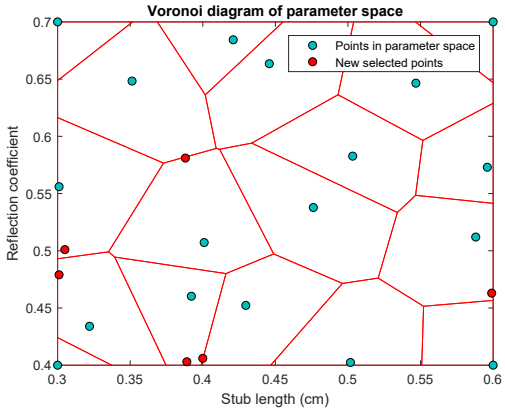


(d) Fourth iteration

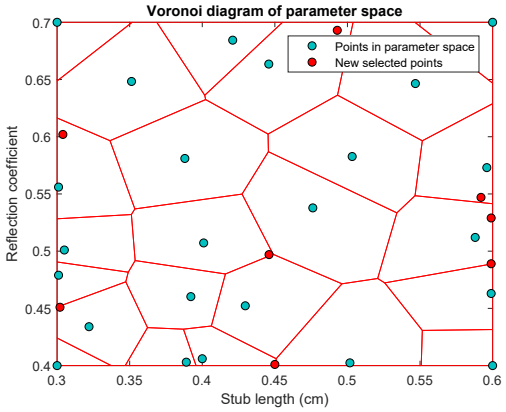
Figure 6.20: Selection based on first criterion



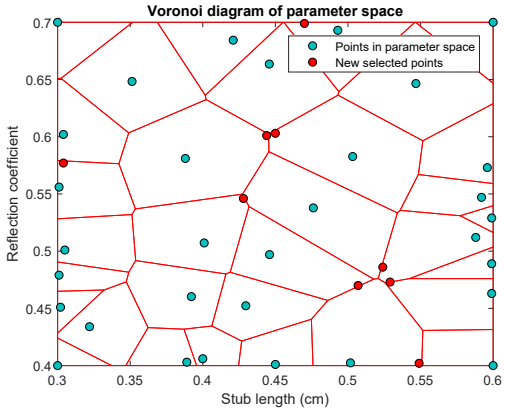
(a) First iteration



(b) Second iteration

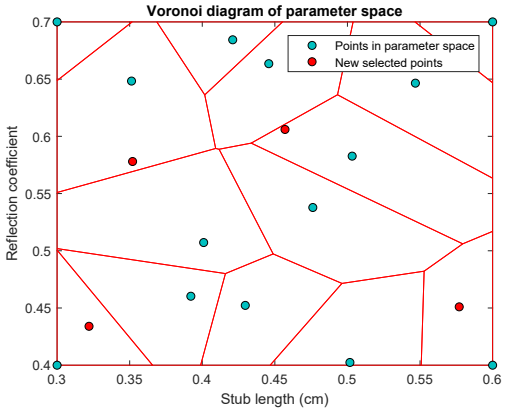


(c) Third iteration

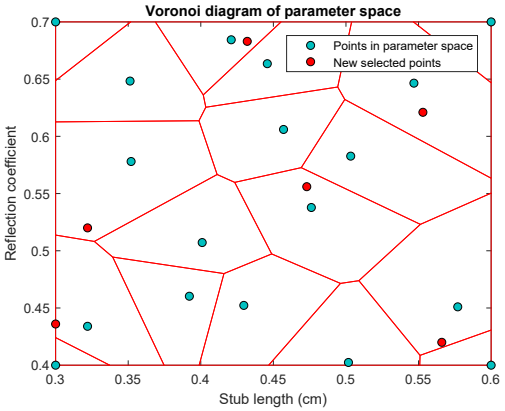


(d) Fourth iteration

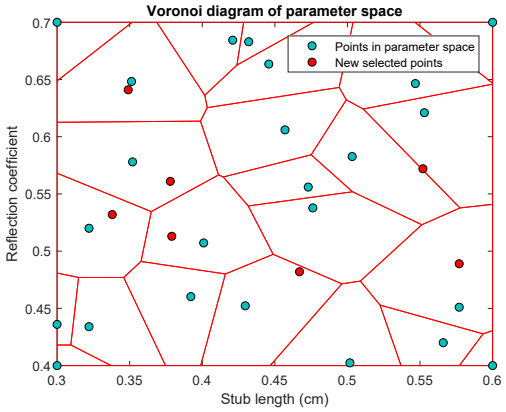
Figure 6.21: Selection based on second criterion



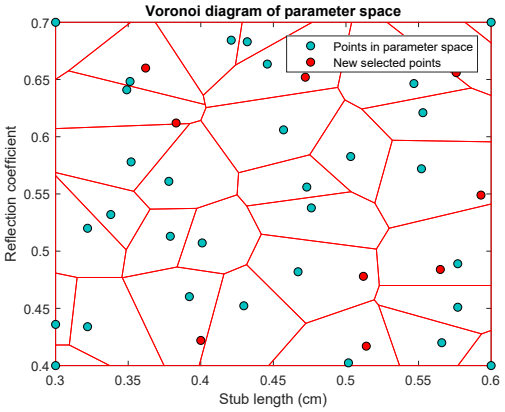
(a) First iteration



(b) Second iteration

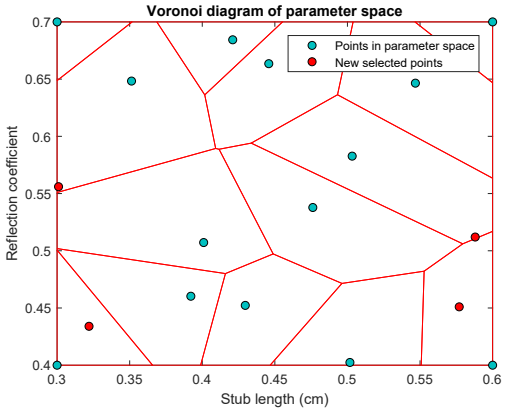


(c) Third iteration

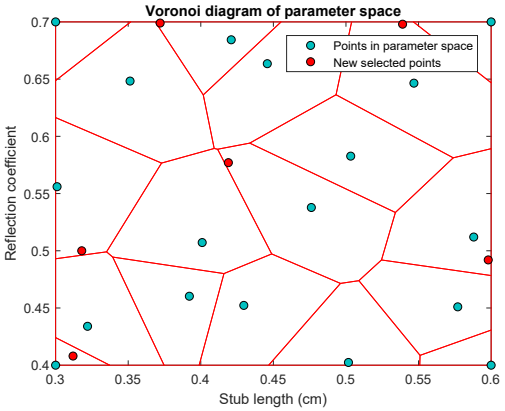


(d) Fourth iteration

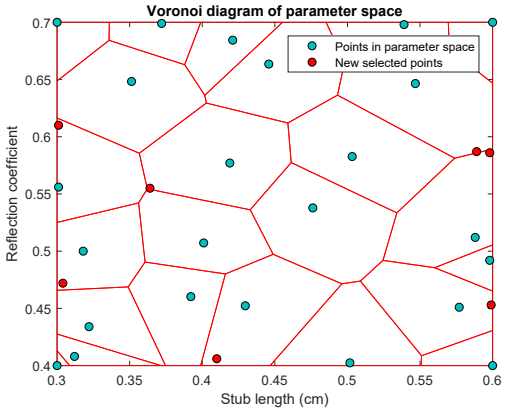
Figure 6.22: Selection based on third criterion



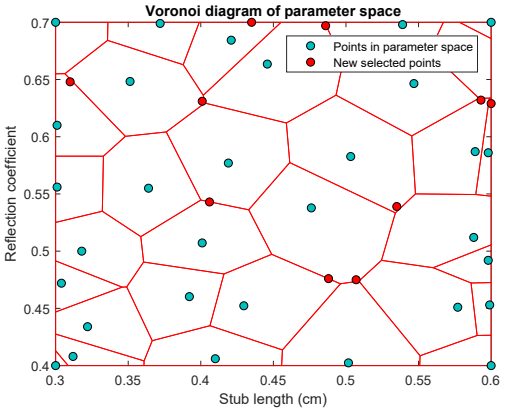
(a) First iteration



(b) Second iteration

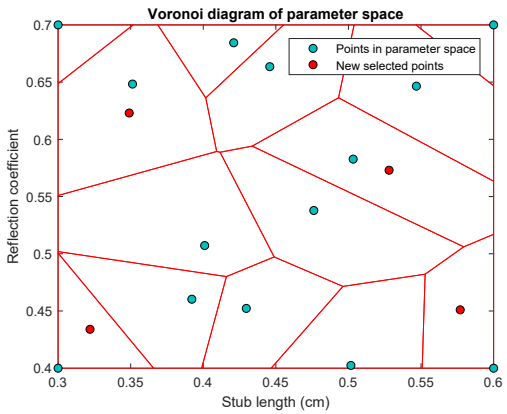


(c) Third iteration

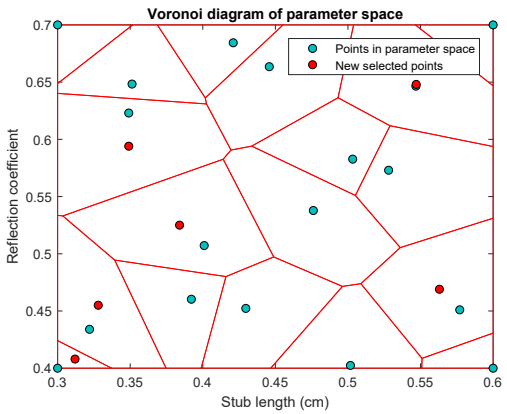


(d) Fourth iteration

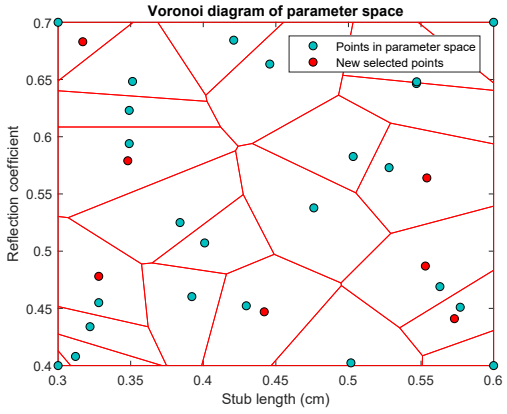
Figure 6.23: Selection based on fourth criterion



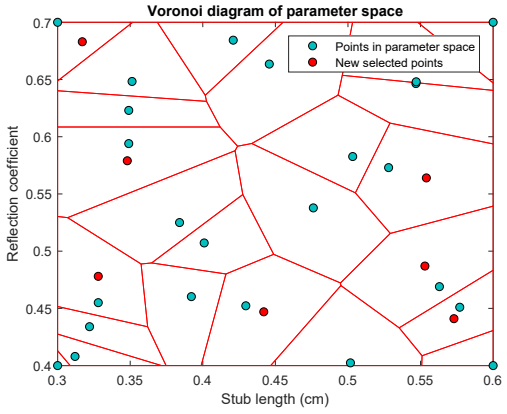
(a) First iteration



(b) Second iteration



(c) Third iteration



(d) Fourth iteration

Figure 6.24: Selection based on fifth criterion

By looking at the figures reported above, some conclusions can be made on the difference among the selection criteria.
By using the first criterion, points seem to be well positioned across the space (not too

close to the already present points), and the cells obtained tend to have more regular sizes with respect to other criteria.
With the second criterion (6.21), it can be noticed that the new points tend to be placed on the edge of the cells and on the boundaries of the parameters space, which is not optimal. The third and the fourth criteria seem to be both valid, especially when compared to the fifth criterion, in which, instead, the new points are often really close to the point already present in the cell. This means that the space is not explored homogeneously, and the addition of the new point could be less effective.
A table reporting the model errors for each criterion can be found below.

Table 6.5: COMPARISON AMONG SELECTION CRITERIA - FILTER LINE-LOAD

Iteration	**First crit.**	**Second crit.**	**Third crit.**	**Fourth crit.**	**Fifth crit.**
1	0.3728	0.3728	0.3728	0.3728	0.3728
2	0.0038	0.0399	0.035	0.013	0.0088
3	0.0250	0.0023	0.003	0.0052	0.0193
4	9.7748e-4	5.37e-4	3.96e-4	4.15e-4	4.9579e-4

Table 6.6: COMPARISON AMONG SELECTION CRITERIA - FILTER STUB-LOAD

Iteration	**First crit.**	**Second crit.**	**Third crit.**	**Fourth crit.**	**Fifth crit.**
1	0.0445	0.0445	0.0445	0.0445	0.0445
2	0.0014	7.035e-4	0.0035	0.00092	0.0012
3	5.68e-4	2.18e-4	1.12e-4	5.16e-4	4.883e-4
4	2.628e-4	4.14e-4	4.36e-4	3.21e-4	1.19e-4

Table 6.7: COMPARISON AMONG SELECTION CRITERIA - FILTER STUB-LINE

Iteration	**First crit.**	**Second crit.**	**Third crit.**	**Fourth crit.**	**Fifth crit.**
1	0.4743	0.4743	0.4743	0.4743	0.4743
2	0.1482	0.0565	0.087	0.0096	0.2587
3	0.0998	0.0686	0.035	0.064	0.0452
4	0.0392	0.0377	0.023	0.045	0.0210
5	0.0032	0.0056	0.0021	0.0027	0.0033

6.4 Comparison with different models

The last analysis conducted on the algorithm is probably also the most important, since it consists of comparing the final model generated from the adaptive selection of points with other two models, generated from the same number of data points: in the first data points are randomly scattered on the parameters space, while in the second data points are placed on a regular grid. The results for three test cases are reported in the table below.

Table 6.8: COMPARISON AMONG MODELS - FINAL MODEL ERROR

Case	Adaptive	Random	Uniform
stub-line	0.0025	0.0149	0.0016
stub-load	0.00019	0.0018	0.00055
line-load	0.000382	0.0127	0.00057

The results reported in the table 6.8 lead to some important comments: the adaptive model results to be always better than the random one, thus confirming the effectiveness of the proposed algorithm.
Moreover, the adaptive model is always at least comparable to the uniform one, often better: this means that the proposed algorithm represents a valid alternative to the standard workflow for model creation (with data points placed on a fine grid). The adaptivity of the choice of points, in fact, allows to add some points at each iteration, still keeping those found at the previous iterations. When using a fine grid, instead, if the number of data points is not sufficient, a finer grid must be defined, which will place the new points in new locations with respect to the previous ones. Thus, all simulations must be recomputed, making this procedure highly ineffective.
This last analysis, then, clearly shows the quality of the proposed adaptive workflow.

Chapter 7

Conclusions

This thesis provides a software tool which implements a class of adaptive and greedy algorithms for the creation of parameterized macromodels. These models are created from tabulated data coming from an EM solver. The presented tool provides a full integration both with the model generator tool and with the field solver.
The algorithm has been tested on a number of examples, and it has proven its effectiveness in all the considered cases.
A number of further analyses has been conducted on many aspects of the algorithm. In particular, the analysis on the selection criteria of the new points inside the cell has brought up some possible improvements to the algorithm with respect to the initial version. The analysis on the combined metric, instead, has confirmed that the standard, initial metric is the best tradeoff in terms of simplicity and effectiveness.
Some aspects of the tool can be further investigated and improved. Among the others, the stability of the generated model should be assessed. Up to now, in fact, the created models are not guaranteed to be stable and can thus be used only for frequency-domain analysis.

Bibliography

[1] S. Grivet-Talocia and B. Gustavsen, *Passive macromodeling: Theory and applications*. John Wiley & Sons, 2015, vol. 239.

[2] B. Gustavsen and A. Semlyen, "Rational approximation of frequency domain responses by vector fitting", *IEEE Transactions on power delivery*, vol. 14, no. 3, pp. 1052–1061, 1999.

[3] T. V. F. Website, "`http:/www.sintef.no`", [Online; accessed 05/30/2018].

[4] C Sanathanan and J. Koerner, "Transfer function synthesis as a ratio of two complex polynomials", *IEEE transactions on automatic control*, vol. 8, no. 1, pp. 56–58, 1963.

[5] W. Hendrickx and T. Dhaene, "A discussion of" Rational approximation of frequency domain responses by vector fitting"", *IEEE Transactions on Power Systems*, vol. 21, no. 1, pp. 441–443, 2006.

[6] S. Grivet-Talocia and E. Fevola, "Compact parameterized black-box modeling via Fourier-rational approximations", *IEEE Transactions on Electromagnetic Compatibility*, vol. 59, no. 4, pp. 1133–1142, 2017.

[7] S. Grivet-Talocia and P. Triverio, "MSDT7 2.0 Documentation",

[8] P. Triverio, M. Nakhla, and S Grivet-Talocia, "Parametric macromodeling of multiport networks from tabulated data", in *Electrical Performance of Electronic Packaging, 2007 IEEE*, IEEE, 2007, pp. 51–54.

[9] P. Triverio, S Grivet-Talocia, and M. Nakhla, "An improved fitting algorithm for parametric macromodeling from tabulated data", in *Signal Propagation on Interconnects, 2008. SPI 2008. 12th IEEE Workshop on*, IEEE, 2008, pp. 1–4.

[10] D. Deschrijver, K. Crombecq, H. M. Nguyen, and T. Dhaene, "Adaptive Sampling Algorithm for Macromodeling of Parameterized S-Parameter Responses", *IEEE Transactions on Microwave Theory and Techniques*, vol. 59, no. 1, pp. 39–45, 2011.

[11] B. Delaunay, "Sur la sphere vide", *Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, no. 793-800, pp. 1–2, 1934.

[12] "http://en.wikipedia.org/Delaunaytriangulation", [Online; accessed 06/06/2018].

[13] N. Metropolis and S. Ulam, "The monte carlo method", *Journal of the American statistical association*, vol. 44, no. 247, pp. 335–341, 1949.

[14] H. Cohn and A. Kumar, "Universally optimal distribution of points on spheres", *Journal of the American Mathematical Society*, vol. 20, no. 1, pp. 99–148, 2007.

[15] "http://en.wikipedia.org/wiki/Polytope", [Online; accessed 06/06/2018].

[16] "http://en.wikipedia.org/wiki/Cross-polytope", [Online; accessed 06/06/2018].

[17] D. Wirtz and B. Haasdonk, "A vectorial kernel orthogonal greedy algorithm", *Dolomites Research Notes on Approximation*, vol. 6, no. Special_Issue, 2013.

[18] C. Lessig, M. Desbrun, and E. Fiume, "A constructive theory of sampling for image synthesis using reproducing kernel bases", *ACM Transactions on Graphics (TOG)*, vol. 33, no. 4, p. 55, 2014.

[19] H. Wendland, "Piecewise polynomial, positive definite and compactly supported radial functions of minimal degree", *Advances in computational Mathematics*, vol. 4, no. 1, pp. 389–396, 1995.

[20] R. Schaback, "A Practical Guide to Radial Basis Functions", [Online; accessed 06/06/2018].

[21] "Compactly Supported Radial Basis Functions, `www.math.iit.edu`", [Online; accessed 06/06/2018].

[22] C. R. Paul, *Analysis of multiconductor transmission lines*. John Wiley & Sons, 2008.

[23] D. De Zutter and L. Knockaert, "Skin effect modeling based on a differential surface admittance operator", *IEEE Transactions on microwave theory and techniques*, vol. 53, no. 8, pp. 2526–2538, 2005.

[24] W. L. Stutzman and G. A. Thiele, *Antenna theory and design*. John Wiley & Sons, 2012.

[25] G. A. E. Vandenbosch and A. Vasylchenko, "A Practical Guide to 3D Electromagnetic Software Tools, Microstrip Antennas, Prof. Nasimuddin Nasimuddin", [2011, Online, accessed 02/01/2018].

[26] C. R. Paul, *Introduction to electromagnetic compatibility*. John Wiley & Sons, 2006, vol. 184.

[27] Keysight, "http://www.keysight.com", [Online; accessed 05/30/2018].

[28] ——, "Keysight EMPro Scripting Cookbook", [Online; accessed 05/30/2018].