Behavioral models such as macromodels, surrogate models, metamodels, and response surface models have many applications in diverse research domains such as aerodynamics [1], hydrology [2], mechanical engineering [3], and many more. When considering the design flow of electronic devices, these models are often used to characterize the time- or frequency-dependent behavior of an electronic component while taking all electromagnetic (EM) phenomena into account: crosstalk, attenuation, dispersion, and coupling effects for example [4]. Such models are of crucial importance for efficient design space exploration, design optimization, and sensitivity analysis [5], [6]. A key advantage is that they are calculated independently of the device’s physics and that they are valid for over a wide range of design variables, taking into account multiple geometrical layout or substrate features. Additionally, the models can easily be linked together in a model cascade.

Some examples of behavioral models include polynomial/rational functions [7]–[10], Kriging models [11], [12], artificial neural networks (ANNs) [13], [14], and support vector machines (SVMs) [15]. To obtain a reliable model that satisfies all design requirements, significant challenges need to be addressed, such as which data collection strategy to use, which model type is most applicable, how to rank different models according to quality, etc. At the same time, electronic design automation (EDA) experts are typically not familiar with the intricacies of these design choices. Their primary concern is obtaining an accurate replacement model with minimal computational overhead. The selection of model types, model parameter optimization, and sampling strategy are of lesser or no interest to them since these are just necessary intermediate steps to solve the overall design problem [16].

In this article, a unified and automated modeling framework is presented that can assist an EDA domain expert in generating accurate behavioral models [17]. It drives the underlying system-level simulator and at the same time builds and tunes the model in such a way that the model accuracy and compactness are maximized. On the one hand, it does not require particular assumptions about the device under test, but on the other hand, as no algorithm is optimal for every problem, full control is still left with the EDA domain expert such that problem-specific assumptions or customizations can easily be applied. Therefore, this work can lower the barrier of entry for domain experts, promote benchmarking between existing methods, and facilitate the transfer of knowledge from behavioral modeling researchers to EDA domain experts.

Global Behavioral Modeling

From an abstract level, the system-level computer simulator can be seen as an unknown multivariate function $f: \Omega \rightarrow \mathbb{C}^q$ that is defined on some domain $\Omega \subset \mathbb{R}^d$ and whose function values $Y = \{f(x_1), ..., f(x_k)\} \subset \mathbb{C}^q$. 

Dirk Gorissen (dirk.deschrijver@gmail.com), Tom Dhaene, and Daniel De Zutter are with the Department of Information Technology (INTEC) at Ghent University—IBBT, Gaston Crommenlaan 8, 9050 Ghent, Belgium. Dirk Deschrijver is a post-doctoral researcher of The Research Foundation Flanders (FWO-Vlaanderen).
are known at a fixed set of pairwise distinct sample points \( X = \{x_1, \ldots, x_i\} \subset \Omega \) [18]. The behavioral model is then a suitable function \( \hat{f} \) that closely resembles \( f \) as measured by some criterion \( \xi \), where \( \xi \) is defined as a triplet that consists of following parts:

\[
\xi = (\Lambda, \epsilon, \tau).
\]

\( \Lambda \) is a model quality estimator, i.e., a function that assigns a positive score to a behavioral model where lower scores indicate a more desirable model. Many implementations of \( \Lambda \) have been described: the error in the samples, the hold-out, bootstrap, cross validation, jack-knife, Akaike’s information criterion (AIC), etc. [19]. An implementation of \( \Lambda \) is typically associated with an error function \( \epsilon \). While \( \Lambda \) specifies the model quality estimation algorithm such as cross validation, \( \epsilon \) specifies what error function should be used to calculate the actual quality score, e.g., mean relative error or maximum absolute error. Finally, \( \tau \) is the model quality target desired by the user.

The behavioral modeling problem (i.e., finding the best approximation \( \hat{f}^* \)) for a given set of data points \( D = \{(x_1,f(x_1)),\ldots,(x_i,f(x_i))\} \) can be formally defined as

\[
\hat{f}^* = \arg\min_{\tau \in \mathcal{T}} \arg\min_{\theta \in \Theta} \Lambda(\epsilon, \hat{f}, \theta, D)
\]

such that

\[
\Lambda(\epsilon, \hat{f}, \theta, D) \leq \tau,
\]

where \( \hat{f} \) is the parametrization \( \theta \) (from a parameter space \( \Theta \)) of \( f \), and \( \hat{f}_\theta \) is of model type \( t \) (from a set of model types \( T \)).

The first minimization over \( t \in \mathcal{T} \) is the task of selecting a suitable model type, i.e., a rational function, a neural network, a spline, etc. This is the model type selection problem. In practice, one typically considers only a single \( t \in \mathcal{T} \), though others may be included for comparison. In some cases, the choice of model type is also linked to physical properties of a system such as causality, stability, or passivity [20]–[22]. Then, given a particular model type \( t \), the task is to find the model parameter assignment \( \theta \) that minimizes the model quality measure \( \Lambda \), e.g., determine the optimal order of a polynomial model or the optimal neural network topology. This is referred to as the “hyperparameter optimization problem,” though generally both minimizations (over \( t \) and over \( \theta \)) are simply referred to as the “model selection problem.”

In order to construct \( \hat{f} \), the data set needs to be populated. Traditionally, the size and distribution of the data is chosen up-front using standard experimental designs such as a latin hypercube design. However, since \( f(\cdot) \) is expensive to compute, it becomes important to avoid unnecessary simulations. Consequently, since the complexity of the response surface is not known up front, defining an a priori data distribution is undesired. Instead, data points should be selected iteratively, at locations where the benefit to the model will be the greatest.

One starts by constructing an initial experimental design using one of the many algorithms available from the theory of Design and Analysis of Computer Experiments (DACE) [23]. The task is then to generate a new set of maximally informative samples based on one or more criteria. Examples of such criteria include distance from other points, distance from optima, prediction uncertainty, etc. In each iteration of a sampling algorithm, a best approximation model is obtained, leading to a sequence of models. The overall number of data samples should be kept to a minimum, while at the same time maximizing the benefit to the model \( \hat{f} \) with respect to \( \xi \).

This process is called “adaptive sampling” [24], but is also known as “active learning” [25], “reflective exploration” [7], “Optimal Experimental Design” [26], “Sequential Exploratory Experimental Design” [27], and “sequential design” [28].

Figure 1 shows how the model generation and data collection fit together in a high-level control flow.

There are two main parts to the flowchart: an outer adaptive sampling loop and an inner adaptive modeling loop. Given a set of sample points, the inner loop

---

**Figure 1.** General flowchart for adaptive global behavioral modeling. The outer sampling loop will iteratively sample the design space while an inner modeling loop will generate models that accurately capture the data selected each iteration.
will optimize the model hyperparameters (e.g., neural network topology) until no further improvement is possible on the given set of samples. Having reached a minimum of the hyperparameter space [the inner minimization in (2)], the algorithm then signals the sample selection loop to determine a new, maximally informative set of simulation points. These are then evaluated and merged intelligently with the existing sample points and the adaptive modeling process is allowed to resume. This whole process continues until the user-defined accuracy has been reached or some user-defined stopping criterion is met. The final behavioral model is returned to the user.

The flow of control in Figure 1 forms the basis for every data-based approximation methodology. For example, note the similarities with the work by Zhang in [29]. Of course, as is, Figure 1 has little practical value since it is very vague. Like the expectation-maximization algorithm [30] it is really a meta-algorithm that can have many different actual instantiations, depending on what techniques are used for each step. However, it should be clear that there are a large number of modeling options and choices available to the designer: different model types, different experimental designs, different sample selection strategies, different model selection criteria, different hyperparameter optimization strategies, etc. All choices that are difficult to make in an a priori manner [31].

Consequently, this generic control flow is taken as a starting backbone, extended with pluggable components for each of the different steps. This results in a generic solution that can still be customized for a specific problem if needed. In addition, custom extensions ensure that the whole is more than a bag of tools but some automation (e.g., for model type selection) is available if needed. The resulting modeling platform can easily help an EDA domain expert to choose between different techniques and help him apply advanced modeling methods to his problem with minimal overhead. Depending on the configuration, the modeling platform can run autonomously, for example if just a quick-and-dirty model is needed, or under full manual control with problem specific methods and customizations (to ensure optimal efficiency accuracy for a given problem). The framework in question is the MATLAB Surrogate Modeling (SUMO) Toolbox.

**SUMO Toolbox**
The SUMO Toolbox illustrated in Figure 2 [32], [33] is a flexible tool that integrates different modeling approaches and implements an adaptive behavioral model construction algorithm based on the flowchart in Figure 1. Given an EDA simulation engine (e.g., Simulation Program with Integrated Circuit Emphasis (SPICE), SpectreRF, High Frequency Structure Simulator (HFS), Momentum, Computer Simulation Technology (CST) etc.) the toolbox computes a model within the time and accuracy constraints set by the user. Different plug-ins are supported:

- **Model Types**
  - rational functions
  - Kriging
  - splines
  - SVM

- **Model Parameter Optimization Algorithms**
  - particle swarm optimization
  - efficient global optimization
  - simulated annealing

- **Sample Selection**
  - random
  - error based
  - density based

- **Sample Evaluation Methods**
  - local
  - on a cluster or grid.

The behavior of each software component is configurable through a central XML [34] configuration file, and components can easily be added, removed, or replaced by custom implementations. This is illustrated in Figure 3. In addition, the toolbox provides meta-plug-ins, which include, e.g., evolutionary algorithms to automatically select the best model type for a given problem (see [31] for details) or the possibility to use multiple model selection criteria in concert [35].

There is built-in support for high-performance computing. On the modeling side, the model generation process can take full advantage of multicore CPUs if the MATLAB Parallel Computing Toolbox is available, and even of a complete cluster or grid if the MATLAB Distributed Scheduler is available. This can result in a significant increase in speed for model types where the fitting process can be expensive such as in neural networks.

Likewise, sample evaluation (simulation) can occur locally with the option to take advantage of multicore architectures or on a separate computer cluster or...
All interfacing with the grid middleware (submission, job monitoring, rescheduling of failed/lost simulation points, etc.) is handled transparently and automatically (see [36] for more details). Also, the sample evaluation component runs in parallel with the other components (nonblocking) and not sequentially. This allows for an optimal use of computational resources.

The SUMO Toolbox has already been applied successfully to a very wide range of applications, including RF circuit block modeling [32], hydrological modeling [37], electronic packaging [38], aero
dynamic modeling [39], automotive data modeling [35], and EM compatibility [50]. Besides global modeling capabilities, the SUMO Toolbox also includes a powerful optimization framework based on the Efficient Global Optimization framework developed by Jones [40].

**Applications**

To demonstrate the performance of the framework described above, two EM device modeling problems are discussed: a passive component (bandstop filter) and an active device [low-noise amplifier (LNA)].

**EM Behavioral Modeling of a Bandstop Filter**

The first application concerns the modeling of a parametrized double-folded microstrip stub bandstop filter [6]. The filter with ports $P_1$ and $P_2$ is shown in Figure 4.

The substrate is 0.1270 mm thick with a relative dielectric constant $\varepsilon_r = 9.9$ and a loss tangent $\tan\delta = 0.003$. The lines are infinitely thin and perfectly conducting with $W = 0.1219$ mm. The parametric macro-model of the scattering matrix is built as a function of the varying length of each folded segment $L \in [1.97, 2.41]$ mm and varying spacing between a folded stub and the main line $S \in [0.06, 0.24]$ mm over the frequency range $[5, 20]$ GHz. The EM simulation engine used is ADS Momentum.

To assess the accuracy of the models objectively, a dense $50 \times 30 \times 151$ ($L \times S \times$ frequency) reference grid was calculated. It is important to note that this data set is not used during the modeling process in any way since, typically, such a reference grid is not available. It is simply used to objectively test the quality of the models a posteriori.

**Modeling Settings**

The SUMO Toolbox v6.2.1 is configured with the generic ANN and multivariate rational modeling plug-ins. No problem-specific tuning or settings were used. Thus the results will be indicative of how well a behavioral model can be obtained if the framework is applied without further problem-specific knowledge, using only concepts from a data-modeling perspective. The multivariate rational models are based on a custom implementation [41]. The order selection is performed using a genetic algorithm (population size: 30, number of generations: 20), thus this need not be done manually. The model quality estimator, $\Lambda()$, is fivefold cross validation [42], [19] configured with a mean square error (MSE) function ($\varepsilon$). The ANN models

![Figure 3. Illustration of how the SUMO Toolbox enables algorithms and methods to be combined in different ways through the use of a plug-in-based infrastructure (depicted with permission from [32]).](image)

![Figure 4. Double-folded microstrip stub bandstop filter.](image)
are based on the MATLAB Neural Network Toolbox and are trained with Levenberg-Marquardt back-propagation with Bayesian regularization [43], [44] (600 epochs). Since the ANN models do not support complex data directly, the real and imaginary components are fitted separately using an ANN model with two outputs. The topology and initial weights are determined by an evolutionary strategy-like algorithm, with 25 models being generated each modeling iteration. To assess the model quality and drive, the topology selection, $\lambda$, is taken as the sum of two criteria: the in-sample error (using an MSE) and the linear reference model (LRM) score [45]. The combined scores for each output (real/imaginary) are then added together to obtain the overall score of the model. The LRM score penalizes a model if it exhibits unwanted bumps or ripples between the sample points. It can be seen as a kind of smoothness penalty that has the added benefit of keeping the neural network model complexity low. We found that the advantage of using these two metrics together is that they produce better ANN models and are much faster to evaluate than cross validation.

**Sampling Settings**

The modeling starts with an optimal Latin hypercube design of four points augmented with the corner points in the two-dimensional (2-D) $L \times S$ space. Each iteration a new sample is selected using the local linear approximation (LOLA)-Voronoi adaptive sampling algorithm [46]. LOLA-Voronoi identifies new sample locations by performing a trade-off between exploration, which is covering the design space evenly, and exploitation, concentrating on regions where the actual response is nonlinear. LOLA-Voronoi’s strengths are that it scales well with the number of dimensions, makes no assumptions about the underlying problem or behavioral model type, and works in both the $\mathbb{R}$ and $\mathbb{C}$ domains. LOLA can automatically identify nonlinear regions in the domain and sample these more densely than the more linear, flatter regions. To do this, LOLA-Voronoi depends only on previously evaluated data points.

Because frequency is sampled automatically by ADS Momentum, LOLA-Voronoi samples in the 2-D instance space defined by the geometric parameters $L$ and $S$. New samples are submitted to ADS Momentum, which returns a set of $S$-parameters over the frequency range of interest. In order to select a sample in the reduced 2-D design space (without the frequency parameter), slices are taken at multiple frequencies, and LOLA-Voronoi is used on each slice separately. This results in a nonlinearity estimation for each frequency slice, covering the entire 2-D design space. These estimations are aggregated into one score, which is used to select new samples in locations with the highest nonlinearity over the entire frequency range.

Momentum is configured to return 31 frequency samples, and the SUMO Toolbox is set to terminate after 136 instance simulations ($= 136 \times 31 = 4,216$ data points). Therefore, instead of using an explicit target accuracy value, simulations are performed until the computational budget is exhausted ($\tau = -\infty$).

**Results**

Figure 5 shows a plot of the final rational models for $S_{11}$ and $S_{12}$. For conciseness, the following discussion will only treat $S_{11}$. The results for $S_{12}$ are completely analogous.

Figure 6 shows that the ANN model generation code is able to reduce the true error evaluated over a dense
reference grid quite effectively while minimizing the estimated error. This means that the in-sample error/LRM combination is a good approximator of the true accuracy. In line with previous results [45], we see that the decrease in true error is quite steady with no large jumps.

Figure 7 shows the probability density function of the absolute errors (obtained using kernel density estimation [47]) for the final best ANN model found by the toolbox after 136 momentum simulations. The figure shows that very good accuracy is achieved. Note also the small difference between the training and test curves, meaning there is no overfitting and the models show good generalization. The final ANN model for $S_{11}$ is a $3 - 8 - 16 - 2$ network (210 parameters) and for $S_{12}$ a $3 - 12 - 15 - 2$ network (275 parameters).

This notation denotes the number of elements of the input vector and the number of neurons in each layer of the ANN.

Let us now examine the results for the multivariate rational functions. The evolution of the true error and estimated error during the model generation process is shown in Figure 8. Compared to the ANN results in Figure 6, we see that the error reduction is more erratic in the rational case, particularly in the beginning when only little data is available. This is due to the use of cross validation as the accuracy estimator. Even though we ensure an even distribution of the different folds, when data is relatively sparse, cross validation is known to give biased results [19] and can mislead the order selection procedure.

Again, good accuracy is achieved on $S_{11}$, with the mean accuracy being better than for the ANN models. Given the rational nature of the underlying transfer function, this should not be surprising. However, it should be noted that the modeling effort was not the same for both model types. Since the rational functions are fast to construct and train, we can afford to build more of them during each modeling iteration than neural network models (since these are much slower to train). In this case, $20 \times 30 = 600$ rational models are built each modeling iteration versus only 25 neural networks.

**EM Behavioral Modeling of a Low-Noise Amplifier**

The second application is concerned with the accurate capturing of the nonlinear behavior of an LNA. An LNA is characterized by performance figures such as voltage gain, linearity, and noise figure, which are functions of design parameters such as width and length of transistors, bias conditions, and values of passive components [48]. The goal of the design process is to figure out one or more sets of design parameters resulting in a circuit, which fulfills the specifications, that is, constraints given on the performance. More information on the modeling of this problem can be found in [32].

Obtaining the required circuit design parameters can be done through an approximation of the circuit performance figures based on one or more behavioral models. This is referred to as a “forward model” of the circuit. A forward model can be either obtained via direct modeling of circuit performance using the one-step approach or by using intermediate surrogate models of a convenient set of behavioral parameters (e.g., admittances and noise functions) and by computing performances via analytical equations in a post-processing or two-step approach. This is illustrated in Figure 9.

To illustrate the indirect modeling approach, we consider the LNA and attempt to reproduce the behavior of the input-noise current response variable $\sqrt{i_{in}}$. The input parameters are the MOSFET width $W$, the inductances $L_{in}L_{m1}$, and the frequency, leading to a
four-dimensional (4-D) problem [32]. To illustrate the direct-modeling approach, we model the \( I_{IP2} \) performance parameter (denoting the second-order non-linearity). We also use this approach to illustrate the scalability of the modeling algorithms used. Instead of fixing the number of inputs, we vary them from two to six. This will give insight into how the model accuracy and required number of data points changes as the dimensionality is increased from 2-D to six-dimensional (6-D). The relevant input parameters are the transistor width \( W \), the source inductance \( L_s \), the load resistance \( R_L \), the voltage bias of the transistor \( V_{GS} \), the transistor length \( L \), and the resistance in series with the generator \( R_S \) (the generator series resistance).

**Modeling Settings**

We use the same setup as the previous section but now only use ANN models since experience showed these to perform best on this problem. To assess the accuracy of the produced models objectively, reference test sets of size \( 51, 15, 11, 7, 5, 234, 5, 6 \) are available. Again, remember that these do not influence the modeling process itself.

**Sampling Settings**

The same sample selection algorithm is used as with the filter application, only this time there is no autosampling (the frequency is not treated specially) and the sample budget is extended to 1,580 points in the indirect case and 3,000 points in the direct case. The data source is now a custom MATLAB script instead of ADS Momentum.

**Results**

Figure 10 shows the evolution of the relative error histogram over the reference data during ANN model generation for the 4-D indirect LNA modeling problem. More lighter regions mean a higher percentage of the reference points have a low error. The final model after 1,580 points has a mean relative error of 2.5%.

\[
RRSE(y, \hat{y}) = \sqrt{\frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}}, \tag{4}
\]

where \( y, \hat{y}, \bar{y} \) are the true, predicted, and mean true response values, respectively.
The continuous emergence of new devices and circuit design techniques has led to the development of a wide range of behavioral modeling methods that facilitate design space exploration, optimization, and sensitivity analysis. This article discussed how the use of data modeling techniques can be leveraged in a flexible modeling framework to facilitate domain experts in their modeling task. By way of example, accurate behavioral models with good generalization and scalability were generated for EM bandstop filter and LNA modeling problems. Sample selection and model complexity setting were handled fully autonomously. Future work includes the integration of more specialized fitting algorithms (such as [49] or [51]) as plugins into the SUMO Toolbox so as to ensure maximum accuracy and interpretability of models for electronic devices and systems.

For Further Information
The SUMO Toolbox framework which was used for the tests, including all algorithms and features discussed here, is available under an open source license (AGPLv3) for download at http://www.sumo.intec.ugent.be. The software package contains a collection of demos, data sets, and examples. Additional documentation is also included, and a wiki page with information is available at http://www.sumowiki.intec.ugent.be.

References


