Analysis of Simulation-Driven Numerical Performance Modeling Techniques for Application to Analog Circuit Optimization

Trent McConaghy
ESAT-MICAS
K.U. Leuven
Leuven, Belgium

Georges Gielen
ESAT-MICAS
K.U. Leuven
Leuven, Belgium

***Abstract*—There is promise of efficiency gains in simulator-in-the-loop analog circuit optimization if one uses numerical performance modeling on simulation data to relate design parameters to performance values. However, the choice of modeling approach can impact performance. We analyze and compare these approaches: polynomials, posynomials, genetic programming, feedforward neural networks, boosted feedforward neural networks, multivariate adaptive regression splines, support vector machines, and kriging. Experiments are conducted on a dataset used previously for posynomial modeling, showing the strengths and weaknesses of the different methods in the context of circuit optimization.**

I. Introduction

Automated sizing of analog circuits is important to industrial design practices because it can improve productivity in the analog design process. Over the years, there has been effort to improve automated approaches in both academia (e.g. [1-3]) and industry. Typical approaches use simulated annealing or an evolutionary algorithm, possibly executed in parallel using a farm of workstations. Simulator-in-the-loop optimization has been demonstrated to be especially effective because of the accuracy and flexibility of simulators. The drawback is runtime. Thus, it continues to be important and useful to improve the efficiency.

Recently, there have been proposals to leverage numerical performance modeling to improve efficiency [4-8]. A performance model is a mathematical model relating the performance characteristics of a circuit (e.g. gain/bandwidth) to the design variables. Accuracy is maintained by constructing the performance model based on sample sets of SPICE simulation data. Efficiency of circuit optimization is improved because the performance model can replace at least some of the time-consuming SPICE simulations. In each of those techniques, the choice of model type and model construction technique was made early in the research process without deep considerations for alternative approaches. However, there is a wide variety of possible approaches, so an arbitrarily chosen approach could easily be suboptimal.

This paper analyzes and compares a large set of modeling approaches in the context of a reference optimization flow. Section II describes the flow. Section III discusses modeling criteria. Section IV surveys modeling approaches in light of analog optimization. Section V experimentally compares the modeling approaches. Section VI concludes.

II. Reference Optimization Flow

There are many possible ways to use models to improve optimization efficiency. We limit ourselves to one reference flow, specifically the one that [4] suggests.

```
From an initial "center" design, repeat
until stopping criteria met:
```

- `Use Design Of Experiments (DOE) to sample several data points about the "center" design; simulate each and compute performance values`
- `Build one model for each performance measure, using those samples`
- `Choose a new "center" design by optimizing on the models`

III. Criteria for Modeling

Since circuit simulation is the bottleneck in simulator-in-the-loop optimization, improving efficiency roughly translates to reducing the number of simulations. However, it is important that the time taken to build the model or optimize on the model does not end up making the overall runtime longer. Lower model prediction error can reduce the number of iterations of the algorithm, but there is a tradeoff, as achieving lower prediction error takes more time in model building.

The most important criteria are prediction ability, model building time / scalability, and model simulation time. Specific targets for each criterion depend on the optimization flow, as well as the circuit simulation time, number of process and environmental corners, and number of design variables.

IV. Models and Model Building Algorithms

We choose a representative sample of modeling approaches based on performance, popularity, and diversity of origin. Figure 1 illustrates. The methods we investigate are at the leaf nodes.

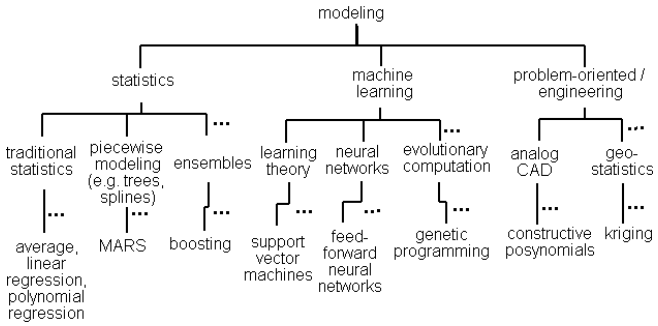

Figure 1. A sampling of modeling approaches with a diversity of origin

The following modeling methods are considered:

- As reference models, we use a constant (set as the mean of the data), a linear model, and a 2nd-order polynomial. A constant is not useful in practice, of course, as it would provide zero information on choosing the next center design. In this paper, no terms are pruned from the polynomials.
- Posynomials are compared because [4] considered them as *the* prime candidate in the reference optimization flow, because posynomials result in a convex optimization problem that can be solved very efficiently. Unlike the polynomials in this comparison, the posynomials in [4] follow a more constructive approach for model building, causing sparser models.
- A modified form of genetic programming (GP) [8] called CAFFEINE [10] is tested. CAFFEINE restricts GP to canonical functional forms via a grammar. In [10], CAFFEINE achieved low testing error in comparison to posynomials on the same dataset that this paper uses.
- Feedforward neural networks (FFNNs) have been applied to a wide variety of problems, including analog circuit sizing [5,7]. We use a state of the art training algorithm, OLMAM [11]. The number of hidden nodes, *NumHid*, is chosen as follows: set *NumHid* = 1; train *NumRestarts* times for *MaxEpochs* training epochs; increment *NumHid*; repeat. When the target nmse is hit, stop.
- Boosting [12] creates a "stack" of models; each model is learned on a weighted version of the data. Weights are based on the previous iteration's errors. The overall output is the average of the outputs of the individual models. Boosted FFNNs have been previously applied to simulation data [6]. We first determine *NumHid* in the same way as for single FFNNs. Then, *NumModels* are sequentially built, with each model having *NumHid* nodes.
- Multivariate Adaptive Regression Splines (MARS) [13] are piecewise polynomials, and are built as follows. In the constructive step, input variables are iteratively added on an "as-needed" basis for greedily chosen sub-regions of the input space. A pruning step follows. MARS scales to a high number of input dimensions but is locally accurate.
- Support vector machines (SVMs) [14] transform inputs into a space of much higher dimension and do linear regression in that space. We use a fast-learning variant, LS-SVM [15], with the following settings: radial basis functions, auto-learn model parameters σ^2 and γ; and auto-select input dimensions. A problem with radial kernels is that the inputs that are kept are treated uniformly, which exaggerates the influence of less important variables.
- Kriging originated in geostatistics, but it has been shown to be useful in optimization [16]. In kriging, prediction is the value of nearby samples "corrected" by a correlated error calculation. Model building time does not scale well with a high number of inputs or of samples. The behavior of both kriging and SVMs is highly dependent on the choice of "distance function" among input points.

V. Experimental Comparison

A. Experimental Setup

As this paper uses the flow advocated by the posynomial work, it is also useful to compare to that work. So, the problem setup and simulation data was identical to [17]. The circuit being modeled is a high-voltage CMOS OTA as shown in Figure 2. The goal is to build predictive models for low-frequency gain (A_{LF}), unity-gain frequency (f_u), phase margin (PM), input-referred offset voltage (v_{offset}), and the positive and negative slew rate (SR_p, SR_n), thus including both small-signal and large-signal transient characteristics.

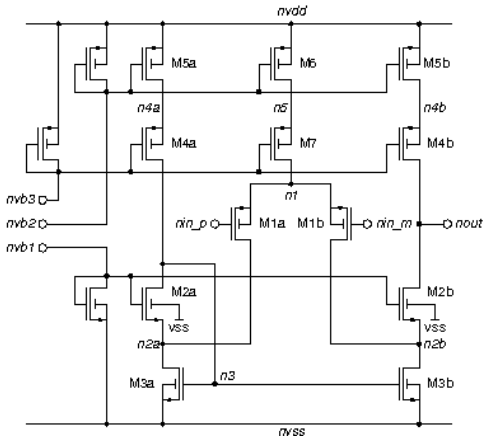

Figure 2. Schematic of high-speed CMOS OTA

The technology is 0.7 μm CMOS. The supply voltage is 5V. The nominal threshold voltages are 0.76V for NMOS devices and –0.75V for PMOS devices. The circuit has to drive a load capacitance of 10 pF. Currents and transistor drive voltages are chosen as design variables, for a total of 13 design variables. Full orthogonal-hypercube Design-Of-Experiments (DOE) sampling of design points was used, with scaled design space dx=0.1 to have 243 design point samples. These samples were used as training data inputs. Separate testing data had DOE sampling with dx=0.03. The testing nmse is equivalent to the posynomial "quality of fit" measure q_{tc} used in [17]. We build each model with a target training normalized mean-squared error (nmse) on the training data, then simulate it on a separate set of test data, and compare the testing nmse. All experiments were on a 3.0 GHz Pentium IV PC running Matlab 6.5 on Red Hat Linux.

The code to build constant, linear, and polynomial models was about 25 lines of Matlab. The model building time was a few seconds, at most. The posynomial results were taken directly from [17]; it reports that the model building time was 1-4 minutes (on a

slower machine). The target training nmse for the other model builders was the posynomial's training nmse from [17].

The CAFFEINE model builder was 2000 lines of Matlab code. Run settings were: maximum number of basis functions 15, population size 200, number of generations 5000, and maximum tree depth 8. Model building time was 12 hours. The time could be probably be lowered by 10x with a C implementation, and perhaps another 5x with a less arbitrary population sizing and number of generations, which would put the time at 14.4 minutes.

The FFNN was the Matlab code from [18]. Settings were *NumRestarts* = 10, *MaxEpochs* = 5000. The time to build a single network was about 10 s. A suitable nmse was typically found in the first or second restart of about 3 hidden neurons. Therefore the total model building time was about (10 s) * (10 restarts) * (first 2 neurons) + (10 s) * (2 restarts) * (1 final neuron) = 10*10*2 + 10*2 = 220 s = 3.7 min. A 10x speedup via a C implementation would make this about 22 s. The boosted FFNN was Matlab code wrapping the OLMAM code. Settings were *NumModels* = 20. Model building time was about (220 s to discover *NumHid*) + (10 s)*(20 models) = 220 s + 200 s = 420 s = 7.0 min. A 10x speedup via a C implementation would make this 42 s.

The MARS model builder was about 500 lines of Matlab code. The model building time was about 5 minutes. A 10x speedup via C would make this 30 s. The SVM model builder was Matlab code from [19], with all settings at "fully automatic." Model building time was about 5 minutes. The kriging model builder was about 200 lines of Matlab code, with Θ_{min}=0.0, Θ_{max}=10.0, p_{min}=0.0, p_{max}=1.99. Model building time was about 5 minutes; a 10x speedup via C would make this 30 s.

B. Model Prediction Results

Figure 3 summarizes how well each of the modeling approaches did in prediction of unseen data; Table I provides details. The Genetic Programming variant, CAFFEINE, does the best. MARS comes in very close. Kriging is next-best. The FFNN, boosted FFNN, and SVM are all very close, and perform about the same as the linear model. Of the non-reference modeling strategies, the posynomial approach does the worst, by far. The only strategy doing worse is the polynomial reference model. Interestingly, only three approaches, namely CAFFEINE, MARS, and kriging, did better at prediction than a constant.

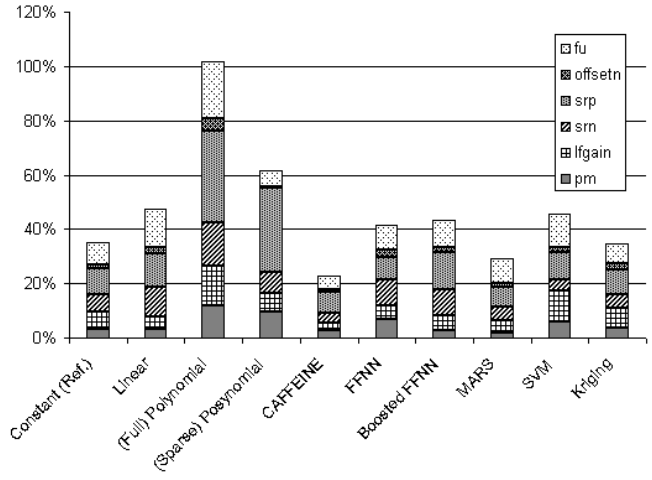

Figure 3. Total prediction (testing) error for each modeling approach

Perf.	*Constant*	*Linear*	*Poly.*	*Posy.*	*CAFFEINE*	*FFNN*	*Boosted FFNN*	*MARS*	*SVM*	*Kriging*
pm	3.3	3.1	11.7	9.7	2.6	6.8	2.8	**1.8**	5.8	3.8
lfgain	6.2	4.6	14.7	6.5	**2.8**	5.0	5.3	4.4	11.5	7.3
srn	6.5	11.0	16.1	7.8	**3.9**	9.5	9.7	5.4	4.1	5.1
srp	9.3	12.4	33.8	31.0	**7.4**	8.2	14.0	**7.2**	10.0	8.9
offsetn	1.8	2.2	4.3	**0.8**	**1.0**	2.9	1.4	**1.2**	1.8	2.2
fu	8.2	14.2	20.8	5.9	**5.0**	9.3	10.0	9.4	12.7	7.3

Table I. Prediction error (%), for each modeling approach and each performance measure. Best or near-best are in bold.

It should not be surprising that the polynomial does the worst, as polynomials are notorious for overfitting. The approach used here is particularly susceptible because it uses all the inputs in all combinations, and the size of the dataset is relatively small in comparison to the number of input dimensions. The posynomials, for which this dataset was originally created, performed terribly compared to *all* the other non-reference modeling approaches. Posynomials did almost as badly as the similar approach of polynomials; the posynomials did better probably because the model building algorithm was constructive, leading to more sparse models.

CAFFEINE had the opportunity to use more complex functions such as sin(), but in its quest to build models to suit the data, it used rationals on 5 of the 6 performance goals. CAFFEINE only selects input variables that really matter. It is biased towards the axes of the input variables rather than being affine-invariant. MARS did similarly, because it is also biased towards the axes and is selective of input variables. While CAFFEINE had the best or near-best prediction error on 5 of the 6 performance goals, MARS had the best or near-best on 3. As we shall see, the other approaches lose prediction accuracy because they have different biases.

Kriging performed fairly admirably in this setting. This is not surprising because it tends to perform well when the input samples have relatively uniform spacing, as they do here with the DOE sampling. Kriging, FFNNs, and boosted FFNNs did worse than CAFFEINE and MARS most likely because they did not have the helpful (for this application) bias towards the input axes. The boosted FFNN did not have noticeably superior performance to the FFNN, which means that overfitting was likely not an issue with the FFNN. The SVM's performance was poor probably because it treated the variables it selected too uniformly. Also, the support vector at the center of the sampling hypercube has to reconcile all the other samples, which it does not really have enough parameters to do properly. Because kriging did substantially better than SVMs, the choice of distance function was likely not an issue.

C. Relation to Optimization

The main factors in choosing a model builder for a given optimization flow are prediction ability, model building time, and model runtime. We have analyzed prediction ability and model building time, so let us examine model runtime.

Runtime of each model was always about 0.1 seconds or less in Matlab. A 10x speedup via C would make it 0.01 s. Let us say that one could optimize on the models (of six performance goals) using, say, 10,000 evaluations with a modest-performance optimizer such as simulated annealing. It is not necessary to get a perfectly optimal point because the model has prediction error anyway. The total runtime would be = (0.01 s / model) * (6 models / design point) * (10,000 design points) = 10 minutes.

We could perhaps devise a formula for choosing the allowed time for model building, so as to minimize total runtime. However, it would involve knowing *a priori* the number of iterations that the algorithm would undergo in order to solve the problem, something we do not know in advance, especially because it is a function of the prediction error, which in turn is a function of the modeling approach and model building time.

Instead, let us use a simple "rule of thumb": balance *(circuit simulation time)* with *(model building time + time to optimize on models)*. This roughly ensures that the extra computational cost incurred by modeling does not, in a worst case, make optimization runtime much longer than an optimization approach with simpler but less informed heuristics to choose new designs (such as simple mutation of a design point). And, for non-worst cases, the "more informed" choices could greatly improve runtime because the number of iterations would decrease. With each new center design point there are actually 243 new circuits simulated; if each circuit takes 10s to simulate, then that is 40 minutes of simulation time. So, following the rule of thumb, if at each iteration the circuit simulation time is 40 minutes and time to optimize on models is 10 minutes, then there remains (40 – 10) = 30 minutes to build the models. Therefore, with 6 performance goals, we get (30 minutes allocated model building time for all goals) / (6 goals) = 5 minutes allocated to build each model. All the modeling approaches discussed above can be built in 5 minutes except for CAFFEINE. That leaves MARS as the best-predicting approach that meets the constraint of model building time. For a first pass, MARS is what we would recommend for this particular optimization flow and problem. However, for a fully informed decision we would do more experiments on different center points, and with different inputs and outputs. As well, we would explore techniques similar to MARS, and GP speedup techniques.

It is interesting that the choice of [4] for using posynomials as a modeling approach was largely motivated by the fact that posynomials allow efficient convex optimization could be used to determine the next center design. However, as just discussed, the efficiency of that optimizer is the main issue because model simulation time is several orders of magnitude less than circuit simulation time.

VI. Conclusions

We have compared a total of ten approaches to automatically build circuit performance models from circuit simulation data, with application to more efficient analog circuit optimization. The approaches that predicted new data with the lowest error were CAFFEINE (a genetic programming variant) and MARS (multivariate adaptive regression splines). The approaches that did the next best were kriging, FFNNs (feedforward neural networks), boosted FFNNs, and SVMs (support vector machines). The reference approaches of simple constants and linear models did about the same as those approaches. The worst-predicting models were polynomials and posynomials.

The models were examined in light of a reference optimization flow, for which [4] advocated posynomials, with the argument that posynomials are easy to optimize on. However, in this flow the most important aspect is not ease of optimization on the model, but instead is reducing circuit simulation time through the most predictive models (subject to the constraints that models must be built and simulated fast enough). As CAFFEINE models are costly to construct, that makes MARS the choice for this application.

References

[1] R.Phelps, M.Krasnicki, R.A.Rutenbar, R.Carley, J.R.Hellums, "ANACONDA: simulation-based synthesis of analog circuits via stochastic pattern search," IEEE Trans. CAD 19(6), June 2000

[2] B. De Smedt, G. Gielen, "Watson: Design space boundary exploration and model generation for analog and RF IC design," IEEE Trans. CAD 22(2), pp.213-224, Feb. 2003

[3] H. Chang, E. Charbon, U. Choudhury, A. Demir, E. Felt, E. Liu, E. Malavasi, A. Sangiovanni-Vincentelli and I. Vassiliou. A top-down, constraint-driven design methodology for analog integrated circuits. Kluwer, 1997.

[4] W. Daems, G. Gielen, W. Sansen, "Simulation-based generation of posynomial performance models for the sizing of analog integrated circuits," IEEE Trans. CAD 22(5), May 2003, pp. 517-534

[5] P. Vancorenland, G. Van der Plas, M. Steyaert, G. Gielen, W. Sansen, "A layout-aware synthesis methodology for RF circuits", ICCAD 2001, San Jose, CA, p. 358

[6] H. Liu, A. Singhee, R.A. Rutenbar, L.R. Carley, "Remembrance of circuits past: macromodeling by data mining in large analog design spaces," DAC 2002, New Orleans, pp. 437 - 442

[7] G. Wolfe, R. Vemuri, "Extraction and use of neural network models in automated synthesis of operational amplifiers", IEEE Trans. CAD, Feb. 2003

[8] F. De Bernardinis, M.I. Jordan, A. Sangiovanni Vincentelli, "Support vector machines for analog circuit performance representation," DAC 2003, Anaheim, CA

[9] J.R. Koza. Genetic Programming. MIT Press, 1992.

[10] T. McConaghy, T. Eeckelaert, G. Gielen, "CAFFEINE: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming", DATE 2005 (not published)

[11] N. Ampazis, S.J. Perantonis, "Two highly efficient second order algorithms for training feedforward Networks", IEEE Trans. Neural Networks 13(5), pp. 1064-1074, Sept. 2002

[12] R.E. Schapire, "The boosting approach to machine learning: An overview," MSRI Workshop on Nonlin. Estimation and Classification, 2002

[13] J.H. Friedman, "Multivariate adaptive regression splines", Annals of Statistics 19, pp. 1-141, March 1991

[14] H. Drucker, C.J.C. Burges, L. Kaufman, A. Smola, and V. Vapnik, "Support vector regression machines," Adv. in Neural Information Processing Systems 9, Cambridge, MA, pp. 155-161, 1997

[15] J.A.K. Suykens, J.Vandewalle. Least Squares Support Vector Machines. World Scientific Pub. Co., Singapore, 2002

[16] D.R. Jones, M. Schonlau, W.J. Welch, "Efficient global optimization of expensive black-box functions," J. Glob. Opt. 13(4), pp. 455-492, 1998

[17] W. Daems, G. Gielen, W. Sansen, "An efficient optimization-based technique to generate posynomial performance models for analog integrated circuits", DAC 2002, New Orleans, LA

[18] N. Ampazis, S.J. Perantonis, "OLMAM neural network toolbox for Matlab," http://iit.demokritos.gr/~abazis/toolbox/, 2002

[19] J.A.K. Suykens, "LS-SVMlab software", http://www.esat.kuleuven.ac.be/sista/lssvmlab/