On-the-Fly Fidelity Assessment for Trajectory-Based Circuit Macromodels

Saurabh K Tiwary, Rob A Rutenbar
Electrical & Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA
{stiwary, rutenbar}@ece.cmu.edu

***Abstract*— Trajectory methods offer an attractive methodology for automated extraction of macromodels from a set of training simulations. A pervasive concern with models based on regression is the lack of certainty about where they fit correctly. We show how the unique structure of a scalable trajectory model allows it to monitor the "fidelity" of the fit automatically, and flag where additional model training is warranted. Experimental results demonstrate this self-monitoring ability in practical circuit examples.**

I. Introduction

Macromodels are simplified circuits which capture just the essential behaviors of some target circuit, and are fast enough to support full system simulation. Recent work on a class of macromodels called *trajectory based models* [6] [5] [1] [4] [7] [8] have shown the potential to generate macromodels on demand. Trajectory methods sample the state trajectory of a circuit as it is simulated in the time domain, and build a macromodel by reducing the linearizations created at an appropriately chosen subset of the time points visited during training simulations, and then interpolating among them. Interpolation combines these reduced linearizations to predict the dynamic behavior of the circuit at any new point in the state-space not visited during prior training. Trajectory methods can build macromodels *on demand*: both the "template" and the "fitting" come directly from training runs.

The trajectory methodology has been demonstrated across a range of realistic circuits [2] [6]. Recent work [8] [7] has shown how the core interpolation scheme can be made scalable to support very large amounts of training data – and thus, more robust model fits – how it can be embedded in a real simulator (SPICE3 [9]), and how trajectory models can be hierarchically inserted into larger SPICE simulations.

However, there remains one unaddressed issue in the proposed versions of trajectory models – that of model *fidelity*. Since trajectory models are generated using a regression based approach, the accuracy of the generated models is strongly dependent on the quality of model training. There are often instances when the model produces inaccurate output because of inadequate training. Traditionally, the only way to check for this was to compare the macromodel's output waveforms against a known correct solution, i.e., a full (slow) SPICE simulation of the original circuit. From the designer's point of view, we would ideally like a model which is never wrong. Failing that, we believe a model should at least be able to flag when its output behavior is "suspicious", i.e., label those parts of the result waveforms in which the model is unable to vouch for the fidelity of the result. This would make working with macromodels much less risky, since we know when we need to go check on the goodness of the fit. As it turns out, a scalable trajectory model in the style of [8] [7] has an interpolation mechanism that can be extended to support this goal. We refer to this as "on-the-fly" fidelity assement, since the goal is not to correct the model (which is very hard) but simply to flag where the model is diverging from areas where the fit is acceptable.

II. Background

A. Trajectory-Based Models

Circuit simulators represent a circuit as a set of nonlinear differential equations. In state-space form, these can be written as:

$$\left.\begin{aligned} \frac{dg(x)}{dt} &= f(x) + B(x)u \\ y &= C^T x \end{aligned}\right\} \qquad (1)$$

The twin difficulties of simulating complex circuits are: (a) the nonlinearities associated with each transistor require many expensive model evaluations as the circuit moves through its state-space, and (b) the order (N) of the resulting equations is large. Together, these make large circuits with complex models slow to simulate. Trajectory models solve the problem by first linearizing the circuit at different points in the state space to capture the non-linear behavior of circuit across the *reachable* state space of the circuit. They then use model order reduction techniques to project the linearizations down to a reduced order state space. To predict the behavior of the circuit at any given point in the state space, they interpolate these *reduced* order linearizations and solve the linear state space equation at that particular point. The reduced order state space is obtained by merging together the Krylov subspaces [3] at all the trajectory linearizations using a biorthogonalization algorithm.

This simplistic version of trajectory models faces obvious problems of scalability and non-optimal reduced order state space. For any serious implementation of the trajectory modeling infrastructure, the model generated for an analog circuit would have large number of trajectory linearizations ($\sim 10,000$). The model would therefore, require a proportionally large amount of time for interpolation, thus slowing down model evaluation greatly. The reduced order basis generated by merging together the Krylov subspaces for all these linearizations would not be the most optimal as well.

Recent extensions [7] [8] to the trajectory strategy called *scalable trajectory models* have addressed these issues. They generate the state space equation at a new point by interpolating only the k nearest neighbors of that particular point (i.e., the k nearest points in state space visited during the training simulation runs used to build this model). These neighbors are found by using a smart search algorithm which makes the model evaluation time essentially independent of the number of linearizations. Also, the reduced order basis is individually computed for different *local* regions of the state space and the linearizations inside that region are projected using the computed basis. If the model is currently in a particular local region, the corresponding reduced order subspace is used for interpolation and model evaluation.

B. Accuracy of Generated Models

The quality of the models generated using trajectory based methodology is dependent on the type of training used to generate the trajectory linearizations. The trajectories are traced by the circuit in its state space for the inputs for which the model is trained. These trajectories are then sampled and linearizations generated at these sampled points. During actual use of the model, if the model is excited to unsampled regions of the state space, the output produced by the macromodel may not match the actual transistor level circuit output. Thus, an obvious way to generate a robust model is to train the circuit for a range of input waveforms such that almost all of the reachable state space is sampled. Unfortunately, we have no formal means to guarantee this, in the case of general circuits with arbitrary nonlinearities. This is, from the designer's viewpoint, quite annoying: "*is that glitch in my waveform a bug in my circuit, or a bug in my macromodel?*" If resolution of this question requires re-simulation of the original circuit at device level, this is obviously quite inefficient. A much more attractive solution would be if the model can "self monitor", that is, if the model can track some internal measure of its own confidence in the fidelity of its output, and tag low-confidence regions of the simulated waveforms for further investigation, or model refinement. As it turns out, the interpolation structure of a scalable trajectory model has all the proper information to allow us to do this.

III. Self-Monitoring Models

A. Distance as Error Criterion

The notion of sampling the state space for capturing the non-linearities, as used in trajectory based models, gives us an idea of how one could implement *self-monitoring models*. Since the dynamics of the circuit is interpolated from a set of sampled linearizations that are close to the current point, we might guess that the quality of interpolated equations depends on how "close" the linearizations actually are. If the interpolating equations are very close by, then the linearized state space equation at the interpolated point should be very similar to the ones for the previously sampled training points. Hence, using these interpolating equations should produce "high confidence" waveforms. On the other hand, if the sampled linearizations are far away from the current point, there is a high probability that the interpolated equations are very different from the actual circuit behavior, leading the model to mispredict the circuit's response. In other words, can we use some sort of a *distance metric* to predict model fidelity for a particular time-step evaluation?

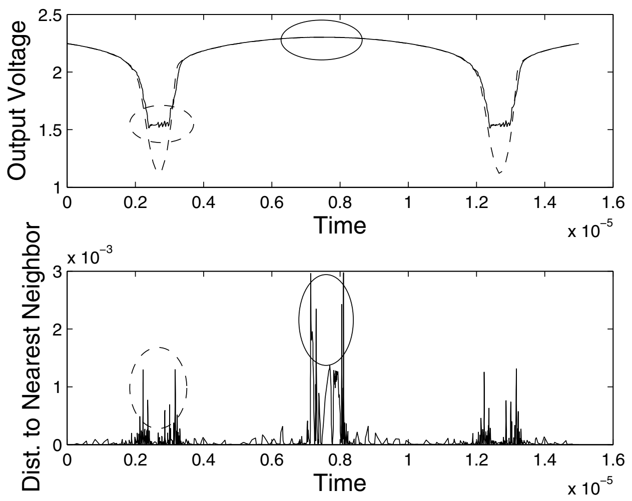


Fig. 1. Using simply the distance metric does not help estimate model accuracy. (a) Figure compares the output produced by the model (solid) and circuit (dashed). (b) The distance to the previously visited nearest neighbor (NN) training point as the model is evaluated through state space. When the distance is large, the model does produce inaccurate results (dashed ellipses). However, in some cases, it produces good results even when the distance is large (solid ellipes).

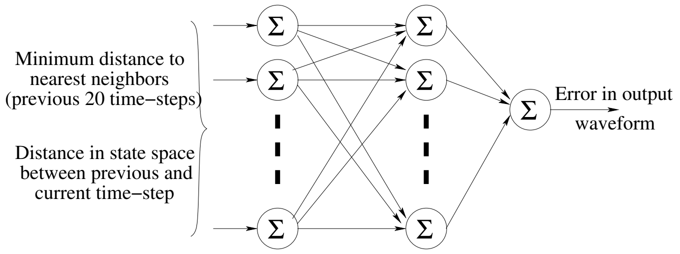
Figure 1 shows the problems with using this idea. In regions where the model prediction is inaccurate, there does seem to be a correlation between the model error and distance from the nearest neighbors. However, there are other instances where the interpolating distance is quite large but the model accuracy is quite good. We can explain this as follows. There are regions in the state space where the dynamics of the circuit are changing very fast. Hence, if the interpolating points are far off, the model produces an inaccurate output. However, there are other regions in the state space where the circuit is very stable with minimal dynamics and small non-linear behavior. Thus, in these regions, even though the sampled points are far off, they all predict the same behavior, which is why we get good model response. We need a simple way to distinguish between these two cases, if we are to employ distance as a surrogate for our confidence in the model's fidelity.

B. Neural Networks for Fidelity Prediction

The above discussion suggests that we need to use some sort of a *locality* information along with the distance heuristic to predict model fidelity *on-the-fly*. That is, in certain regions of the state space, large interpolating distances mean inferior model response while in certain others, distance does not matter that much. The model output is good no matter what the distance from the nearest neighbor is. Modeling this kind of behavior is a standard machine learning problem and we use a standard technique of using Neural Networks to handle it. The locality information is utilized by training different neural networks for different regions of the state space. The trajectory modeling algorithm already segments the regions of the state space for local reduced order state space generation (Section II-A). For each of these regions, we train a neural network, corresponding to that region, to predict the waveform error of the local trajectory interpolation, as compared to the error-free training waveform.

We use the following information to train the neural networks.

- Distance from the nearest neighbor at the current time-step and 20 previous time-steps. The past history trains the network for "memory effects". If the interpolation was wrong at earlier time points, the model, most likely, will be producing incorrect output at the current time-step as well.
- Distance between the location of the model at the current and previous time-step. If the model dynamics are changing very fast, it is likely that it is producing incorrect output.



Minimum distance to nearest neighbors (previous 20 time−steps)
Distance in state space between previous and current time−step
Σ
Error in output waveform

Fig. 2. Architecture of the neural network used for self-monitored fidelity assessment by the macromodel. It is a three layered neural network with one hidden layer. Sigmoidal fuction is used at each of the neurons to compute their respective outputs.

Once the trajectory samples are generated, the neural networks are trained by comparing the model response against the circuit output for a test input. Figure 2 shows the architecture of the neural network used for predicting model fidelity. If the local models for a particular region of the state space are used for interpolating the linearizations, we train the neural network corrsponding to that region with the input information and errors in output waveform. Once, they have been trained, we could fire these neural networks to get an estimate of model error at each evaluated time-step.

C. Applications

We need to point out that as is the case with any machine learning algorithm, the trained neural network output will not always be perfect. Hence, the error estimates generated by them should not be used to "correct" the output of the trajectory model. Rather, our central idea is to employ these as an indicator of our confidence in the fidelity of the model fit for this time point. If the predicted error is sufficiently large in comparison to the waveform, we flag the waveform as suspect. The goal is to provide the designer with a quick "go / no-go" indicator that the model is working correctly. When parts of the waveform are flagged, the user should check the waveforms against full SPICE simulation of the device-level circuit and if possible, re-train the circuit. In cases where the predicted errors are small, the user can keep using the model without any alterations.

Another point to note is that since these neural network predictors are added on top of the core scalable modeling framework that we have developed [6] [8] [7], using them slows down the model. They require extra computation to predict the errors in model output. We lose simulation speed-ups while using them. Thus, we envision a special "diagnostic" mode in which the designer enables this self-monitoring capability. If one is unsure about model accuracy, one can turn on the predictor feature and get an idea of how good the models actually are. If the errors are too large, it means we are visiting unsampled/sparsely sampled regions of the state space. This means we need to "repair" the model.

An additional, attractive feature native to the trajectory based modeling methodology is that it is quite inexpensive to repair an incompletely trained model. We just need to populate the unsampled region of the state space with the trajectory linearizations for the failing test input and add these to the already created model database. There is no need to regenerate all the trajectory samples across all the training input once again. Once the model is incrementally updated, it is ready for use as a replacement of the original circuit.

IV. Experimental Results

We have implemented a version of the scalable trajectory modeling methodology into Berkeley-SPICE3. We have also added the *on-the-fly* fidelity predictor on top of the modeling infrastructure. In this section, we present results showing the efficacy of this predictor.

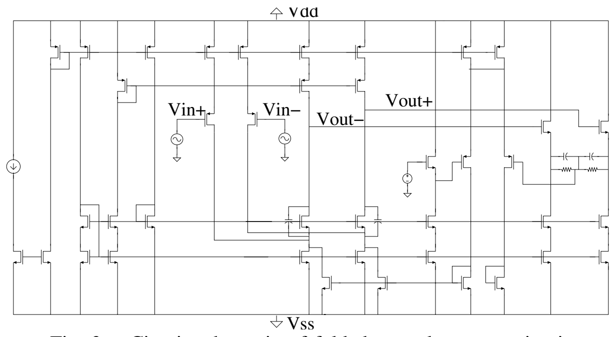


Fig. 3. Circuit schematic of folded-cascode opamp circuit.

We used a folded cascode opamp circuit (Figure 3) to test our idea of predicting fidelity on-the-fly. It is a 40 transistor circuit with 24 dimensional state space. Once the model was trained and reduced trajectory linearizations generated corresponding to local models, we trained our predictors (neural networks) for a series of test waveforms. There was one neural network (Figure 2) for each local region that we (automatically) partitioned the state space into. That particular network was trained when the local models corresponding to its region were fired for interpolating the state space equations. After training, when a new set of test waveforms was used as input to the model, the neural network corresponding to the interpolating cluster was again fired. However, this time, we used the output produced by it as an estimate of the error in the waveform produced by the model. Using a simple threshold magnitude for the error waveform, we labeled the output as *good* or *bad* depending on whether the error was within the set threshold or more than it.

In the experiment, we trained our model for inputs ranging from 1mV to 10mV. However, we tested the circuit with an amplitude of 19mV. In Figure 4, the top figure compares the wavefroms generated by the circuit and its trajectory model. The bottom figure compares the real error waveforms between model and circuit output and the ones generated by our predictor models. We can see that there is a close match between the two. It is important to reiterate here that since the predictor neural networks are themselves based on regression based training, their error estimates should not be taken in absolute terms, rather, only in a classifier sense – small versus large error. We used a threshold of 0.05V to generate the high confidence (white regions in top part of Figure 4) and low confidence (greyed regions in top part of Figure 4) regions of the output waveforms for the trajectory models.

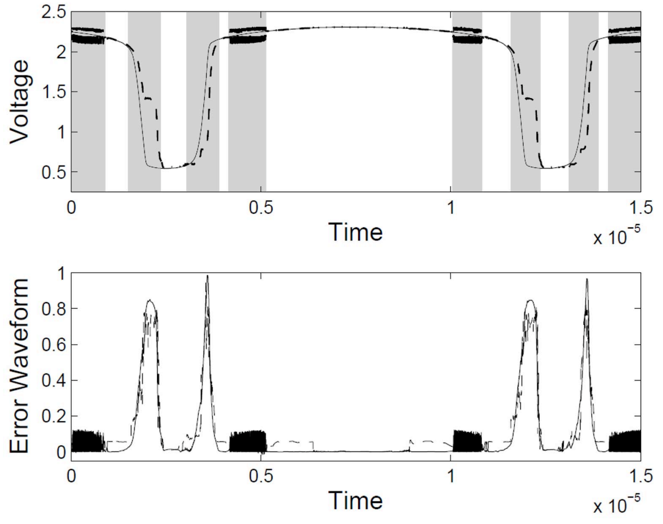


Fig. 4. Top figure plots the circuit and model output for a test input outside the range for which model was initially trained. Greyed areas are regions where the predictors suggest low model accuracy. Bottom figure compares the estimates of error in circuit and model waveforms by the predictor (dashed) compared to real errors (solid).

Since the trajectory model was intentionally under-trained, we found a test input (corresponding to the output waveforms plotted in Figure 4) for which the model output was different from the circuit output. There were regions where the model showed ringing behavior (black patches in Figure 4) due to incomplete model training that was accurately captured by our fidelity predictor as well. To “repair” the model, we *trained* the model for this failing test input and added the new linearizations to the database from prior training. The new model, thus, should not only capture the circuit behavior for this failing test input but also for the range of training inputs that it was initially trained for. For example, when we excited this *repaired*-model again with the failing test input, we see a close match between the circuit and model output (top part of Figure 5). This is also validated with low values of error waveforms (except at transition edges) computed by our predictor (lower part of Figure 5). A few more training inputs also correct the slight slewing errors still flagged by this rather conservative setting for the fidelity threshold.

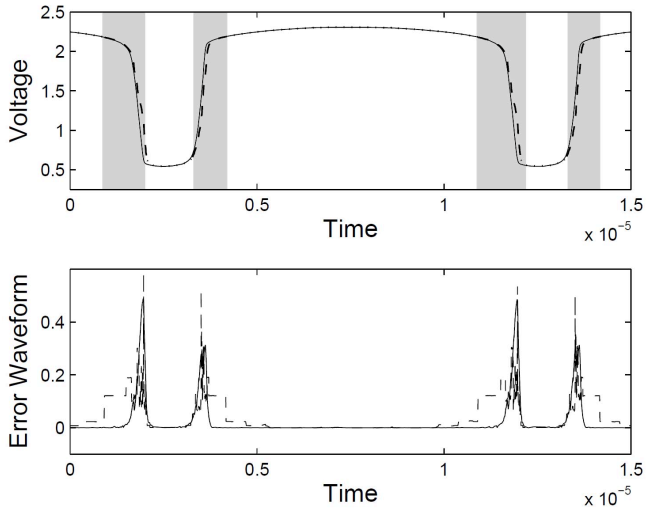


Fig. 5. Top figure plots the circuit and *repaired*-model output for the same failing test input of Figure 4. Greyed areas are regions where the predictors suggest low model accuracy. Bottom figure compares the estimates of error in circuit and model waveforms by the predictor (dashed) compared to real errors (solid).

V. Conclusion

We have presented an *on-the-fly* fidelity assessment methodology for trajectory based models. The quality, applicability and on-demand nature of trajectory models make them a suitable candidate for solving system level simulation problems. This mechanism and its co-existense with the incremental model update feature provides a strong foundation for building better, more accurate models. The predictors also act as a platform for giving real-time feedback about model accuracy to the working circuit designer, trying to use a trajectory-based model.

References

[1] N. Dong and J.Roychowdhury. Automated extraction of broadly applicable nonlinear analog macromodels from SPICE-level descriptions. In *CICC*, 2004.
[2] N. Dong and J.S.Roychowdhury. Automated nonlinear macromodelling of output buffers for high-speed digital applications. In *DAC*, pages 51–56, 2005.
[3] E.Grimme. Krylov projection methods for model reduction. In *PhD Thesis, UIUC*, 1997.
[4] M.Rewienski and J.White. A trajectory piecewise linear approach to model order reduction and fast simulation of non-linear circuits and micromachined devices. *TCAD*, pages 155–170, 2003.
[5] M. Rewienski and J. White. A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. In *TCAD*, pages 155–170, 2003.
[6] M. J. Rewienski. A trajectory piecewise-linear approach to model order reduction of nonlinear dynamical systems. *PhD Dissertation, MIT*, 2003.
[7] S.K.Tiwary and R.A.Rutenbar. Scalable trajectory methods for on-demand analog macromodel extraction. In *DAC*, pages 403–408, 2005.
[8] S. K. Tiwary. Scalable trajectory methods for on demand analog macromodel extraction. In *Phd Thesis (in preparation), CMU*, 2006.
[9] T.Quarles. The SPICE3 implementation guide. In *UCB/ERL M89/44*, April 1989.