17

Automated Model Generation Approach Using MATLAB

Likun Xia
Universiti Teknologi PETRONAS
Malaysia

1. Introduction

High level models comprise both faulty and fault-free models. High level fault-free modeling may simply indicate behavior of a fault-free circuit, but normally it is not able to cope with faulty conditions with strong nonlinearity. The only way to solve this is to replace the fault-free model with a faulty one. Furthermore, in fault-free simulation, the difference in term of simulation speed between transistor level and high level may not be obvious, but this can be shown under fault simulation. High level fault modeling (HLFM) techniques have shown the potential ability to deal with at least some degree of nonlinearity in large systems.

Unlike for linear systems, no technique currently guarantees for completely general nonlinear systems, even in principle, to produce a macromodel that conforms to any reasonable fidelity metric. The difficulty is due to the fact that nonlinear systems can be widely varied, with extremely complex dynamical behavior possible, which is very far from being exhaustively investigated or understood. Generally in view of the diversity and complexity of nonlinear systems, it is difficult to conceive of a single overarching theory or method that can be employed for effective modeling of an arbitrary nonlinear block.

Models can be obtained either manually or automatically. Automated model generation (AMG) approaches are becoming an increasingly important component of methodologies for effective system verification. Similar to manual creation, AMG can generate lower order macromodels via an automated computational procedure by receiving the information from transistor level models (Roychowdhury 2003; Roychowdhury 2004).

Unfortunately, there are not any approaches describing the use of AMG approaches for HLFM at a system level except for the publication in (Xia, Bell et al. 2010). For straightforward system simulation relatively simple models may be adequate, but they can prove inadequate during HLFM. The accuracy and speedup of existing models may be doubted when fault simulation is implemented because faulty behavior may force (nonfaulty) subsystems into highly nonlinear regions of operation, which may not be covered by their models. Multiple training data is required to cover the potentially wide range of operating conditions.

The chapter is organized as follows. Section 2 reviews various AMG approaches using MATLAB. A specific AMG approach using MATLAB is presented in sections 3. Section 4 demonstrates the results in simulated and real environments followed by conclusion for the chapter in section 5.

For clarity, an attempt has been made to adhere to a standard notational convention. Lower case **boldface** characters will generally refer to vectors. Upper case **BOLDFACE** characters will generally refer to matrices. Vector or matrix transposition will be denoted using $(.)^T$ and $(.)^*$ denotes conjugation for complex valued signals. $\Re^{K\times K}$ denotes the real vector space of $K\times K$ dimensions.

2. Review of automated model generation approaches using MATLAB

Automatic generation of circuit models for handling strong nonlinearity has received great interest over the last few years. It is essential for realistic exploration of the design space in current and future mixed-signal SoCs (system-on-chips) and SiPs (system-in-packages). Generally such techniques take a detailed description of a block such as SPICE level netlist and then generate a much smaller macromodel via an automated computational procedure. The advantage of this approach is its generality. As long as the equations of the original system are available numerically, knowledge of circuit structure, operating principles and so on are not very important (Roychowdhury 2003).

The model generated by AMG can be structured as either linear-time invariant (LTI), linear-time varying (LTV), nonlinear-time invariant or nonlinear-time varying. LTI no doubt form the most important class of dynamical systems. The basic structure of a LTI block for mixed mode circuits is illustrated in Fig. 1, where u(t) and y(t) represent inputs, and output to the system in the time domain, respectively. U(s) and Y(s) are forms in the Laplace domain. The definitive property of any LTI system is that the input and output are related by convolution with an impulse response h(t) in the time-domain, i.e., $y(t) = x(t) * h(t)$, their transforms are related to multiplication with a system transfer function H(s), i.e., $Y(s) = X(s) \cdot H(s)$. Their relationship can be expressed by partial differential equations (PDEs) or ordinary differential equations (ODEs). Such differential equations can be easily implemented using analogue hardware description language (AHDL).

A typical model structure for LTI is AutoRegressive with eXogenous (ARX) that is able to describe any single-input single-output (SISO) linear discrete-time dynamic system (Ljung 1999).

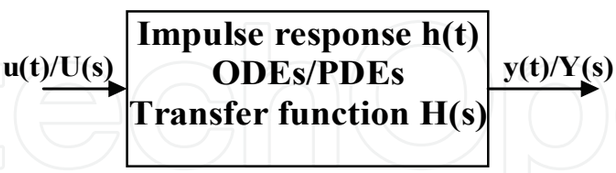


Fig. 1. Linear time-invariant block

LTV models are used in practice because most real-world systems are time-varying as a result of system parameters changing as function of time. They also permit linearization of nonlinear systems in the vicinity of a set of operating points of a trajectory. Similar to LTI systems, LTV can also be completely characterized by impulse responses or transfer functions. The main difference between them is that time-shift in the input of LTV does not necessarily result in the same time-shift of the output. A basic structure of LTV is depicted in Fig. 2, where u(t) and y(t) represent inputs, and output to the system in the time domain, respectively. U(s) and Y(s) are forms in the Laplace domain.

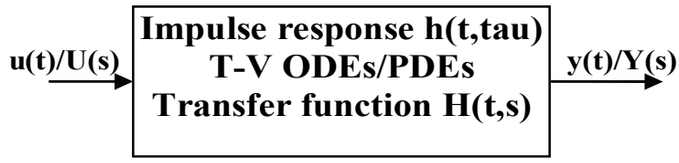


Fig. 2. Linear time-varying block

LTV are capable of handling time variation in state-space forms (Ljung 1999). Furthermore, nonlinear models such as the Wiener and Hammerstein model, and Situation-Dependent AutoRegressive with eXogenous (SDARX) give much richer possibilities to describe systems.

These models can be generated by using estimation algorithms, which comprise lookup tables (Yang and McGaughy 2004), radial basis functions (RBF) (Mutnury, Swaminathan et al. 2003), artificial neural networks (ANN) (Davalo and Naïm 1991; Zhang and Gupta 2000) and its derivations such as fuzzy logic (FL) (Verbruggen and Babuška 1999) and neural-fuzzy network (NF) (Uppal and Patton 2005), and regression (Simeu and Mir 2005). Model generators can also be categorized into the black, grey or white box approaches, depending on the level of existing knowledge of the system's structure and parameters. Dong et al (Dong and Roychowdhury 2005) indicates that white-box methods can produce more accurate macromodels than black-box methods. However, this work was only applied to a limited number of digital circuits.

Regression using MATLAB is an approach that is of interest in this chapter. It is a form of statistical modeling that attempts to evaluate the relationship between one variable (termed the dependent variable) and one or more other variables (termed the independent variables) (Ljung 1999). It can be divided into linear regression and nonlinear regression for generating linear or nonlinear models. (McConaghy, Eeckelaert et al. 2005; McConaghy and Gielen 2005) use the regression approach (Hong, Sharkey et al. 2003), via the predicted residual error sums of squares (PRESS) statistic (Breiman 1996), to test predictive robustness of linear models that are generated by an automatic symbolic model generator named CAFFEINE (Canonical Functional Form Expression in Evolution). CAFFEINE takes SPICE simulation data as inputs to generate open-loop symbolic models by using genetic programming (GP) via a grammar that is specially designed to constrain the search to a canonical functional form without cutting out good solutions. Results show that these models are interpretable, and handle nonlinearity with better prediction quality than posynomials (coefficients of a polynomial need not be positive, and the exponents of a posynomial can be real numbers, while for polynomials they must be non-negative integers). However, McConaghy et al did not address whether the generated model can be fitted into a large system and model nonlinearity well. Additionally, speed of model generation was not mentioned.

Unfortunately, AMG may produce high order models of excessive complexity for both continuous-time and discrete-time systems, so model order reduction (MOR) techniques are required. The purpose of MOR is to use the properties of dynamical systems in order to find approaches for reducing their complexity, while preserving (to the maximum possible extent) their input-output behavior. It comprises a branch of systems and control theory (Roychowdhury 2004). Combining MOR with the model structures produces new model structures dubbed LTI MOR (Pillage and Rohrer 1990), LTV MOR (Phillips 1998; Roychowdhury 1999) and weakly nonlinear methods including polynomial-based (Li and

Pileggi 2003; Li and Pileggi 2005), trajectory piecewise linear (TPWL) (Rewienski and White 2001), and piecewise polynomial (PWP) (Dong and Roychowdhury 2003).
Mathematically, a LTI model with a MOR method is expressed as a set of differential equations. In (1) u(t) represents the input waveforms to the block and y(t) are the outputs. The number of inputs and outputs is relatively small compared to the size of x(t), which is the state of the internal variables of the block. A, B, C, D and E are constant matrices, $E \,\&\, A \in R^{n\times n}, B \in R^{n\times p}, C \in R^{p\times p}, u(t) \in R^{p}$.

$$\begin{aligned} E\dot{x} &= Ax(t) + Bu(t) \\ y(t) &= C^T x(t) + Du(t) \end{aligned} \tag{1}$$

MOR methods for LTI systems fall into two major groups: Projection-based methods and Non-projection based methods. The former consists of such methods as Krylov-subspace (moment matching methods), Balanced-truncation method, proper orthogonal decomposition (POD) methods etc. Krylov-subspace based techniques such as Padé-via-Lanczos (PVL) techniques (Feldmann and Freund 1995), Krylov-subspace projection methods were an important milestone in LTI MOR macromodeling (Grimme 1997). Non-projection based methods comprise methods such as Hankel optimal model reduction, singular perturbation method, various optimization-based methods etc. Via Krylov-subspace operation, reduced models are obtained in (2), where $\tilde{E}, \tilde{A}, \tilde{B}, \tilde{C}$ are reduced order matrices, $\tilde{E} \,\&\, \tilde{A} \in R^{q\times p}$, $\tilde{B} \in R^{q\times p}$, $\tilde{C} \in R^{p\times q}$, W and V are matrices for spanning the matrices.

$$\tilde{E} = W^T E V, \tilde{A} = W^T A V, \tilde{B} = W^T B, \tilde{C} = CV \tag{2}$$

However, the reduced models using Krylov methods retained the possibility of violating passivity, or even being unstable (Roychowdhury 2003).
LTI MOR may not be applicable for many functional blocks in mixed signal systems that are usually nonlinear. It is unable to model behaviors such as distortion and clipping in amplifiers. Therefore, LTV MOR is required. The detailed behavior of the system is described using time-varying differential equations as shown in (3):

$$\begin{aligned} E(t)\dot{x} &= A(t)x(t) + B(t)u(t) \\ y(t) &= C(t)^T x(t) + D(t)u(t) \end{aligned} \tag{3}$$

The dependence of A, B, C, D and E on t is able to capture time-variation in the system. This time-variation is periodic in some practical case such as in mixers, the local oscillator input is often a square waveform or a sine waveform, switched or clocked systems are driven by periodic clocks (Roychowdhury 2003).
Although LTV MOR may be used when modeling some weakly nonlinear systems, in most of cases nonlinear system techniques are required for such systems. A standard nonlinear system formation is based on a set of nonlinear differential-algebraic equations (DAEs) shown in (4), where, $x \in R^{n}$, n is the order of matrices, x(t) and y(t) indicate the vectors of circuit unknowns and outputs, u is the input, $q(\cdot)$ and $f(\cdot)$ are nonlinear vector functions, and b and c are input and output matrices, respectively.

$$\begin{aligned} \dot{q}(x(t)) &= f(x(t)) + bu(t) \\ y(t) &= c^T x(t) \end{aligned} \tag{4}$$

A polynomial approximation is simply extension of linearization, with f(x) and q(x) replaced by the first few terms of a Taylor series at the bias point x_0 as shown in (5), where q(x) = x (assumed for simplicity), $\otimes$ is the Kronecker tensor products operator, $A_i = \frac{1}{i!}\frac{\partial^i f}{\partial x^i}\Big|_{x=x_0} \in R^{n\times n^i}$. The utility of this system in (5) is that it becomes possible to leverage an existing body of knowledge on weakly polynomial differential equation systems.

$$\begin{aligned} &\frac{d}{dt}(x(t)) = f(x_0) + A_1(x-x_0) + A_2(x-x_0)\otimes(x-x_0) + \cdots A_i(x-x_0)^{(i)} + bu(t) \\ &y(t) = c^T x(t) \end{aligned} \tag{5}$$

Volterra series theory (Schetzen 1980) and weakly nonlinear perturbation techniques (Nayfeh and Balachandran 1995) can then be used to justify a relaxation-like approach for this kind of systems. The former provides an elegant way to characterize weakly nonlinear systems in terms of nonlinear transfer functions (Volterra 2005). By using Volterra series, response x(t) in (5) can be expressed as a sum of responses at different orders, i.e., $x(t) = \sum_{n=1}^{\infty} x_n(t)$, x_n is the n^{th} order response. The linearized first order through third order nonlinear responses in (5) need to be solved recursively using Volterra series as shown from (6) to (8), where $\overline{(x_1 \otimes x_2)} = \frac{1}{2}((x_1 \otimes x_2) + (x_2 \otimes x_1))$.

$$\frac{d}{dt}(x_1(t)) = A_1 x_1 + bu \tag{6}$$

$$\frac{d}{dt}(x_2(t)) = A_1 x_2 + A_2(x_1 \otimes x_1) - \frac{d}{dt}(x_1 \otimes x_1) \tag{7}$$

$$\frac{d}{dt}(x_3(t)) = A_1 x_3 + 2A_2\overline{(x_1 \otimes x_2)} + A_3(x_1 \otimes x_1 \otimes x_1) + \frac{d}{dt}(x_1 \otimes x_1 \otimes x_1) - 2\overline{(x_1 \otimes x_2)} \tag{8}$$

The n^{th}-order response can be related to a Volterra kernel of order n, $h_n(\tau_1,...,\tau_n)$, which is an extension to the impulse response function of the LTI system exhibited in (9), to capture both nonlinearities and dynamics by convolution. Volterra kernels are the backbone of any Volterra series. They contain knowledge of a system's behavior, and predict the response of the system (Volterra 2005).

$$x_n(t) = \int_{\infty}^{\infty} \cdots \int_{\infty}^{\infty} h_n(\tau_1,\ldots,\tau_n) u(t-\tau_1)\ldots u(t-\tau_n) d\tau_1 \ldots d\tau_n \tag{9}$$

Alternatively, a variant that matches moments at multiple frequency points is shown in (10), where $h_n(\tau_1,...,\tau_n)$ is transformed into the frequency domain via Laplace transform.

$$H_n(s_1,\ldots,s_n) = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} h_n(\tau_1,\ldots,\tau_n) e^{-(s_1\tau_1+\cdots+s_n\tau_n)} d\tau_1 \ldots d\tau_n \tag{10}$$

$H_n(s_1,\ldots,s_n)$ is referred to as the nonlinear transfer function of order n. The nth-order response, xn, can also be related to the input using $H_n(s_1,\ldots,s_n)$.

Unfortunately, the size of Volterra based nonlinear descriptions often increase dramatically with problem size. Li et al combines and extends Volterra and projection approaches using a method termed NORM (Nonlinear model Order Reduction Method) to reduce the model size (Li and Pileggi 2003).

Outside a relatively small range of validity, but polynomials are known to be extremely poor for global approximation (Roychowdhury 2004), so other methods such as piecewise approximation can be used to achieve better solutions. (Rewienski and White 2001) developed an approach termed trajectory piecewise-linear (TPWL) using a piecewise-linear (PWL) system. Initially Rewienski et al select a reasonable number of "centre points" along a simulation trajectory in the state space, which is generated by exciting the circuit with a representative training input. Around each centre point, system nonlinearities are approximated by implicitly defined linearization. A model is generated if the current state point x is 'close enough' to the last linearized point x_i, i.e., $\|x - x_i\| < \varepsilon$, which means that x lies within a circle of radius of ε and centred at x_i. Each of the linearized models takes the form shown in (11), with expansions around states x_0 ,…, x_{s-1}: where x_0 is the initial state of the system and A_i are the Jacobians of f(.) evaluated at states x_i.

$$\frac{dx}{dt} = f(x_i) + A_i(x - x_i) + Bu \tag{11}$$

A Krylov subspace projection method is then used to reduce the complexity of the linear model within each piecewise region. Rewienski et al then combined all s linear models according to a weighting equation in (12), where $\tilde{w}_i(x)$ are weights depending on state x.

$$\frac{dx}{dt} = \sum_{i=0}^{s-1} \tilde{w}_i(x) f(x_i) + \sum_{i=0}^{s-1} \tilde{w}_i(x) A_i(x - x_i) + Bu \tag{12}$$

TPWL is more suitable for circuits with strong nonlinearities such as comparators, and has more advantages than PWL because as the dimension of the state-space in PWL grows one concern with these methods is a potential explosion in the number of regions which may severely limit simplicity of a small macromodel. However, Rewienski et al did not address the criterion of the training stimulus. Moreover, because PWL approximations do not capture higher-order derivative information, the ability of TPWL to reproduce small-signal distortion or intermodulation is limited. Therefore, Krylov-TBR TPWL was developed using TBR projection to obtain further order reduction (Vasilyev, Rewienski et al. 2003).

The piecewise polynomial (PWP) technique (Dong and Roychowdhury 2003), which is a combination of polynomial model reduction with the trajectory piecewise linear method, is able to improve TPWL by dividing the nonlinear state-space into different regions, each of which is fitted with a polynomial model around the centre expansion point. These points can be selected either from "training simulation" or from DC sweeps. The resulting macromodel is refined incrementally by new piecewise regions until a desired accuracy is reached. Firstly they expand a polynomial function into many points, each of them is then simplified by approximating the nonlinear function in each piecewise region to obtain much smaller size models. These models are then stitched together. Finally a scalar weight function is used to ensure fast and smooth switching from one region to another. A key advantage of PWP is that a macromodel generated can capture not only linear weakly

nonlinear (such as distortion and intermodulation) but also strongly nonlinear (such as clipping and slewing) system dynamics. Moreover, fidelity in large-swing and large-signal analysis can be retained. PWP is further implemented in (Dong and Roychowdhury 2004) for extracting broadly applicable general-purpose macromodels from SPICE netlists such that the generated model is able to capture different loading effects, simultaneous switching noise (SSN), crosstalk noise and so on. Furthermore, a speed up of eight times simulation speed is achieved (Dong and Roychowdhury 2005). However, multiple training data has to be used to cover different operating regions.

Xia et al (Xia, Bell et al. 2010) developed an algorithm to generate multiple macromodels automatically to perform HLFM and high level modeling (HLM). Moreover, the models generated contain low-orders (2nd), so MOR is not required. More details on the approach will be discussed in section 3.

3. The multiple model generation approach using MATLAB

3.1 Introduction to least square estimate

Linear models can be obtained using recursive least square (RLS) estimation. It is a mathematical procedure for finding the best-fitting curve to a given set of points by minimizing the sum of the squares of the offsets of the points from the curve (Ljung 1999). Its general process is shown in Fig. 3, where u(t) is the input stimulus, which is used to connect both a system and the estimator; y(t) is the output response from a system using the transistor level simulation (TLS); $y_E(t)$ is the output response using an estimation approach such as the RLS.

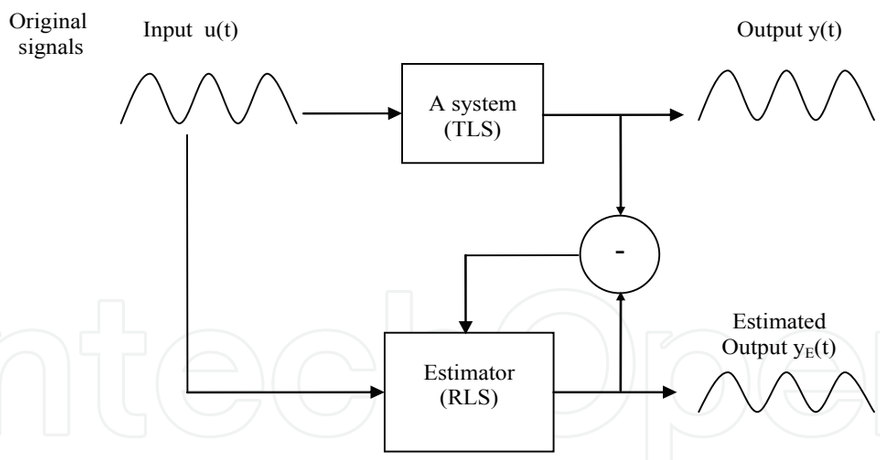


Fig. 3. General process of the estimation

Both the system and estimator use the input stimulus to produce individual output response, which are then compared, if the difference is significant, the parameters of the model need to be modified in order to reduce difference.

Wilkinson et al (Wilkinson, Roberts et al. 1991) combined RLS estimation with the delta operator (Middleton and Goodwin 1990) to obtain the transfer function of a real time controller for a servo motor system instead of using discrete-time transfer function because that model coefficients in discrete-time models strongly depend on the sampling rate, which result in aliasing and slow simulation time. By using the delta operator the coefficients

produced relate to physical quantities, as in the continuous-time domain model, but are less susceptible to the choice of sampling interval (Wilkinson, Roberts et al. 1991). Initially a discrete-time system is given in (13):

$$y(t) = -a_1 y(t-1) - a_2 y(t-2) - \ldots a_{na} y(t-na) + b_1 u(t-1) + b_2 u(t-2) + \ldots b_{nb} u(t-nb) \quad (13)$$

A linear regression form of the system is shown in (14):

$$y(t) = \varphi^T(t)\theta \quad (14)$$

where θ is the parameter vector shown in (15), φ(t) is the regression vector displayed in (16).

$$\theta = \left[a_1\, a_2 \ldots a_{na}\, b_1\, b_2 \ldots b_{nb}\right]^T \quad (15)$$

$$\varphi^T(t) = [-y(t-1) \ldots -y(t-na) \quad u(t-1) \ldots u(t-nb)] \quad (16)$$

The least square estimate (LSE) of the parameter vector can be found from measurements of u(t) and y(t) using (17) (Ljung 1999):

$$\theta(t) = \left[\frac{1}{N}\sum_{t=1}^{N}\varphi(t)\varphi^T(t)\right]^{-1}\left[\frac{1}{N}\sum_{t=1}^{N}\varphi(t)y(t)\right] \quad (17)$$

Its recursive form is expressed in (18), where ε(t) is the prediction error, λ(t) represents forgetting factor (ff), P(t) indicates covariance matrix, and L(t) is the gain vector.

$$\begin{aligned}
&\theta(t) = \theta(t-1) + L(t)\varepsilon(t)\\
&\varepsilon(t) = y(t) - \varphi^T(t)\theta(t-1)\\
&L(t) = \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)}\\
&P(t) = \frac{1}{\lambda(t)}\left[P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)}\right]
\end{aligned} \quad (18)$$

The linear regression is then restructured using the delta operator as shown in (19) (Middleton and Goodwin 1990), where δ represents delta, q is the forward shift operator and Ts is the sampling interval. The relationship between δ and q is a simple linear function, so δ can offer the same flexibility in the modeling of discrete-time systems as q does.

$$\delta = \frac{q-1}{Ts} \quad (19)$$

This operator behaves as a form of the forward-difference formula, as shown in (20) (Burden and Faires 1985). This is used extensively in numerical analysis for computing the derivative of a function at a point.

$$f'(x) = \frac{f(x+h) - f(x)}{h} \quad (20)$$

The delta operator makes use of the discrete incremental difference (or delta) operator that whilst operating on discrete data samples, is similar to those of the continuous-time Laplace operator. A better correspondence can be obtained between continuous and discrete time if the shift operator is replaced by a difference operator that is more like a derivative (Middleton and Goodwin 1990).
A similar procedure is used to achieve regression based on the delta operator. This starts by considering a continuous time transfer function shown in (21).

$$G(s) = \frac{b_0 s^n + b_1 s^{n-1} + \ldots b_n s^0}{s^m + a_1 s^{m-1} + \ldots a_m s^0} \tag{21}$$

When Ts is sufficiently short, the continuous time transfer function G(s) is equal to the delta transfer function G(δ) (Middleton and Goodwin 1990) displayed in (22).

$$G(\delta) = \frac{y(t)}{u(t)} = \frac{b_0 \delta^n + b_1 \delta^{n-1} + \ldots b_n \delta^0}{\delta^m + a_1 \delta^{m-1} + \ldots a_m \delta^0} \tag{22}$$

After arranging this, equation (23) is obtained:

$$y(t)\delta^m = -(a_1 \delta^{m-1} + \ldots + a_m) y(t) + (b_0 \delta^n + \ldots + b_n) u(t) \tag{23}$$

This can be written as (24) (Middleton and Goodwin 1990), which is similar to (14):

$$\delta^m y(t) = \varphi^T(t)\theta \tag{24}$$

where

$$\theta = \left[a_1 \, a_2 \ldots a_m \, b_0 \, b_1 \ldots b_n \right]^T$$

$$\varphi^T(t) = [-\delta^{m-1} y(t) \ldots - \delta^0 y(t) \quad \delta^n u(t) \ldots \delta^0 u(t)]$$

Using a similar approach to LSE in the discrete-time transform, the parameter vector is obtained using the delta operator in (25):

$$\theta(t) = \left[\frac{1}{N} \sum_{t=1}^{N} \varphi(t) \varphi^T(t) \right]^{-1} \left[\frac{1}{N} \sum_{t=1}^{N} \varphi(t) \delta^m y(t) \right] \tag{25}$$

RLS is also obtained in (26), as shown that it is similar to equation (18), the difference is that the vectors including θ, ε, y have been deltarised.

$$\begin{aligned}
\theta(t) &= \theta(t-1) + L(t)\varepsilon(t) \\
\varepsilon(t) &= \delta^m y(t) - \varphi^T(t)\theta(t-1) \\
L(t) &= \frac{P(t-1)\varphi(t)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \\
P(t) &= \frac{1}{\lambda(t)} \left[P(t-1) - \frac{P(t-1)\varphi(t)\varphi^T(t)P(t-1)}{\lambda(t) + \varphi^T(t)P(t-1)\varphi(t)} \right]
\end{aligned} \tag{26}$$

However, the approach in (Wilkinson, Roberts et al. 1991) is only available to single-input single-output (SISO) systems.

3.2 System development using delta transfer function

In this section a novel AMG approach named multiple model gradation system using delta transfer operator (MMGSD) is developed. The concept of process is shown in Fig. 4. The MMGSD generates macromodels by observing the variation in output voltage error against input range. The advantage is that the estimated signal can be adjusted recursively in time to handle nonlinearity. It consists of two parts: the automated model estimator (AME) and automated model predictor (AMP). The AME implements the model generation algorithm, and the AMP uses these models from AME to predict signals in the simulation with different types of stimuli. The system is based on a set of models n. The location of each model is decided by the thresholds seen in u(t).

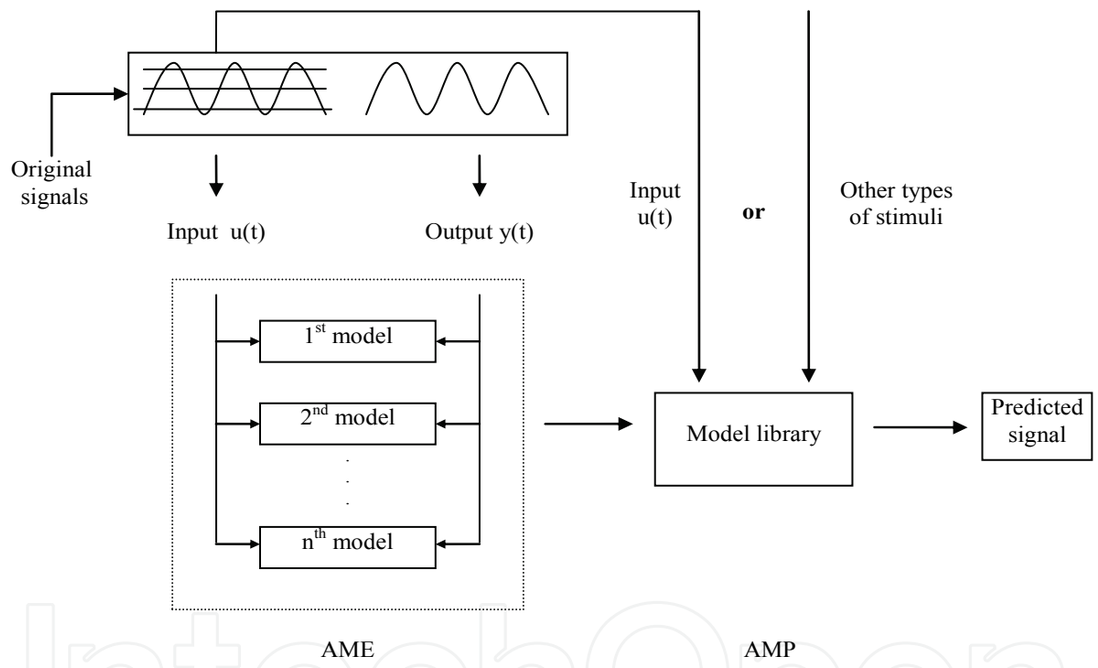


Fig. 4. Schematics for the procedure of MMGSD

The AME comprises three stages: the pre-analysis, estimator and post-analysis. The general structure is shown in Fig. 5.

Pre-analysis is mainly to set up conditions such as input range and the number of intervals for model creation and is only performed once; the estimator is used to determine the quality of output data; post-analysis is the critical step because procedures for creating models are implemented here. This process terminates when no new model is created.

Pre-analysis is mainly to set up conditions such as input range. In the whole algorithm, this stage is only run once. The Estimator process starts by running through all samples using the *for* loop in MATLAB. The indices for creating the threshold are found with a *find* statement. A statement *min* is used to guarantee that only the smallest index is selected, and then the new model pointed by this index is generated. Parameters (th) and the covariance

matrix (p) in each model need to be created and updated. The innovation error (epsi) and residual error (epsilon) are all calculated. Moreover, the prefilter needs also to be updated. The estimation is not over until all samples finish (Ljung 1999).

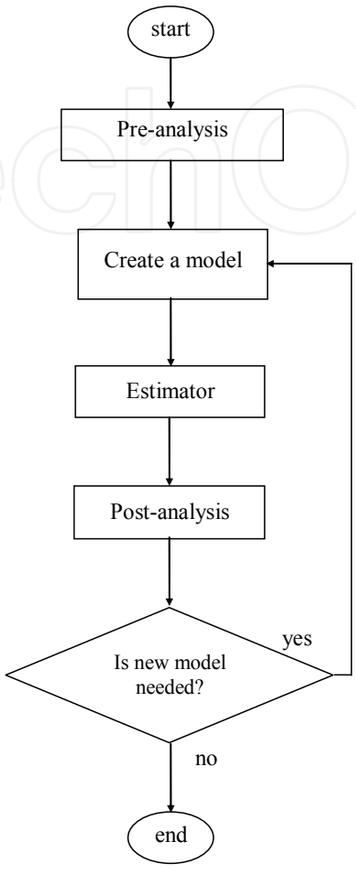


Fig. 5. The flowchart for the AME

Post-analysis is the critical step because procedures for creating models are run here. The workflow is described in Fig. 6. The decision to add a new model to an interval of input voltage is based on (27), where mediumRange is half of the difference between the maximum amplitude of the error (highInterval) and the minimum amplitude of the error (lowInterval) for the interval. criticalRange is the equivalent summation. criteria calculated for the interval results from the comparison of these measures and that of the central interval of the simulation (mediumRange(central)).

$$\begin{aligned} mediumRange &= (highInterval - lowInterval)/2 \\ criticalRange &= (highInterval + lowInterval)/2 \\ criteria &= [mediumRange - mediumRange(central)] - criticalRange \end{aligned} \tag{27}$$

If the difference between two mediumRange is greater than the criticalRange, one model is added within the jth interval (if there are j intervals), otherwise no action is taken. If j is greater than a central point, the threshold will be set at the lower range, otherwise it is set at the higher range in order to obtain the position close to the central point. In order to increase

simulation speed a shift mechanism is used to delete equivalent models. Finally the new threshold array is sorted into monotonic order. Only one model is created per iteration, because the error profile is recalculated whenever a model is added.
The AMP is used to verify the AME system. It loads models generated by the AME to predict output responses.

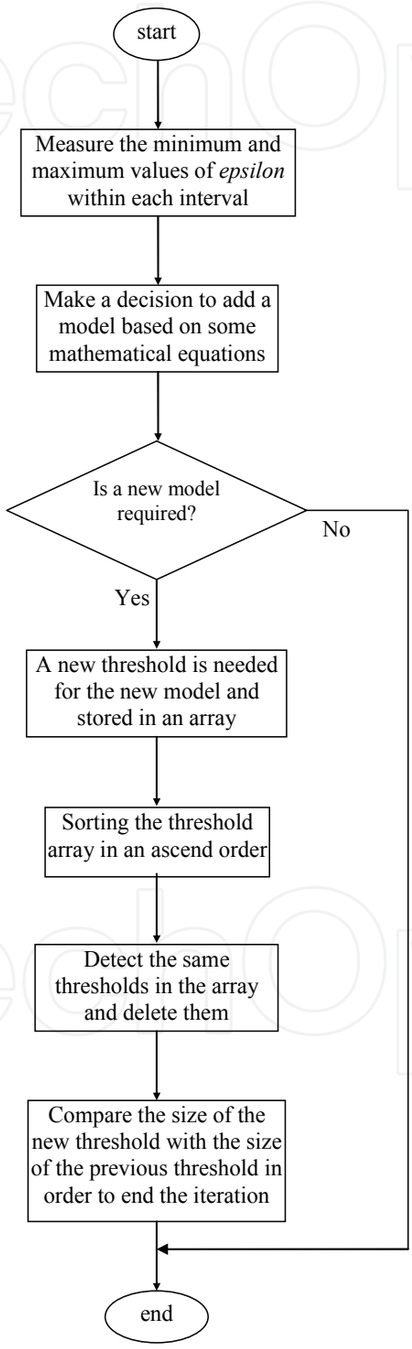


Fig. 6. The algorithm for post-analysis

The model structure is based on the RARMAX system (Ljung 1999) but with modification since RARMAX is under the discrete-time transform, whereas the MMGSD is based on delta transform. Therefore, during simulation (estimation) some quantities in the system need to be either deltarised or undeltarised, for example, the residual error epsilon in the AME and AMP is already deltarised, but during the vector update the undeltarised value is required. Therefore, we create two functions in the MMGSD: the Deltarise function and Undeltarise function. The former is to generate derivative vectors based on original vectors. The undeltarise function requires original data during the estimation. These two functions are used in different places in the MMGSD.

3.2.1 The deltarise function

The deltarise function is used to find the deltarised value using the delta operator given in (28), where delta (δ) is related to both the present and future values, Ts is the sampling rate, q is the forward shift operator used to describe discrete models, which is shown in (29).

$$\delta = \frac{q-1}{T_s} \cong \frac{d}{dt} \tag{28}$$

$$qx_k = x_{k+1} \tag{29}$$

The equivalent form of (29) is obtained in (30), the relationship between δ and q is a simple linear function, so δ can offer the same flexibility in the modeling of discrete-time systems as q does.

$$\delta x_k = \frac{x_{k+1} - x_k}{T_s} = \frac{x(kT_s + T_s) - x(kT_s)}{T_s} \cong \frac{dx}{dt} \tag{30}$$

The use of delta operator and its relationship is illustrated in the following example. It is a discrete-time model, but only output vectors are displayed in (31(a)). Initially each vector is subtracted from the one next to it, as seen in (31(b)), and is then divided by Ts, so deltarised value is obtained, as seen in (31(c)). However, the last one highlighted by the rectangle is not involved in the calculation.

$$\mathrm{y}(\mathrm{t}) \quad \mathrm{y}(\mathrm{t}-1) \quad \mathrm{y}(\mathrm{t}-2) \quad \boxed{\mathrm{y}(\mathrm{t}-3)} \tag{31(a)}$$

$$\mathrm{y}(\mathrm{t}-1) \quad \mathrm{y}(\mathrm{t}-2) \quad \mathrm{y}(\mathrm{t}-3) \tag{31(b)}$$

$$\delta\mathrm{y}(\mathrm{t}-1) \quad \delta\mathrm{y}(\mathrm{t}-2) \quad \boxed{\delta\mathrm{y}(\mathrm{t}-3)} \tag{31(c)}$$

$$\delta\mathrm{y}(\mathrm{t}-2) \quad \delta\mathrm{y}(\mathrm{t}-3) \tag{31(d)}$$

To achieve δ^2y(t-3), equation (31(c)) is subtracted from (31(d)), and then divided by Ts. The procedure is used to obtain δ^3y(t-3) seen in (31(g)).

$$\delta^2\mathrm{y}(\mathrm{t}-2) \quad \boxed{\delta^2\mathrm{y}(\mathrm{t}-3)} \tag{31(e)}$$

$$\delta^2 y(t-3) \tag{31(f)}$$

$$\delta^3 y(t-3) \tag{31(g))}$$

Thus, the deltarised version of (31(a)) is obtained shown in (32).

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \tag{32}$$

The same procedure is also used for other vectors such as the inputs vectors u, e and the noise vector c. Delay is not included here. However, there is some difference such that in the input vector the current deltarised values (u(t), v(t)) are not required.

3.2.2 The undeltarise function

This function is based on (28) but with the modification, q = δTs+1, in order to model at the current time. An example is also used to demonstrate how this reverse algorithm works. It is a model in delta transform, but only the output vectors y are shown in (33(a)). Firstly each vector, except for the last one, highlighted by the rectangle because it is already undeltarised, is multiplied by Ts in (33(b)). We then add the output vectors as shown in (33(b)) and (33(c)), so undeltarised vectors are obtained in (33(d)), i.e., y(t-2) is obtained.

$$\delta^3 y(t-3) \quad \delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \boxed{\delta^0 y(t-3)} \tag{33(a))}$$

$$T_s\delta^3 y(t-3) \quad T_s\delta^2 y(t-3) \quad T_s\delta^1 y(t-3) \tag{33(b))}$$

$$+ \qquad + \qquad +$$

$$\delta^2 y(t-3) \quad \delta^1 y(t-3) \quad \delta^0 y(t-3) \tag{33(c))}$$

$$\| \qquad \| \qquad \|$$

$$\delta^2 y(t-2) \quad \delta^1 y(t-2) \quad \boxed{y(t-2)} \tag{33(d))}$$

To achieve y(t-1), equation (33(d)) is multiplied by Ts, and then we add the vectors shown in (33(e))- (33(g)).

$$T_s\delta^2 y(t-2) \quad T_s\delta^1 y(t-2) \tag{33(e))}$$

$$+ \qquad +$$

$$\delta^1 y(t-2) \quad \delta^0 y(t-2) \tag{33(f))}$$

$$\| \qquad \|$$

$$\delta^1 y(t-1) \quad \boxed{y(t-1)} \tag{33(g))}$$

Finally y(t) is obtained using the same procedure as above.

$$T_s\delta^1 y(t-1) \tag{33(h))}$$

$$+$$

$$y(t-1) \tag{33(i))}$$

$$\|$$

$$\boxed{y(t)} \tag{33(j))}$$

Therefore, the undeltarised version of (33(a)) is achieved shown in (34).

$$\mathrm{y}(t) \quad \mathrm{y}(t-1) \quad \mathrm{y}(t-2) \quad \mathrm{y}(t-3) \tag{34}$$

The number of iterations depends on a variable called *numb*, the reason to use the variable is that during undeltarising, vectors such as output vector need to be undeltarised once to obtain the value at next time, but during the prefilter update, it needs to be fully undeltarsied. If a full undeltarisation is required, the variable is set to 0, otherwise an integer is selected. If the number is greater than the size of the vector array an error message is produced.

3.2.3 Two functions utility in MMGSD

It is known that the delta operator is a very high gain system because of the sampling interval T_s (10us in this case), so it is important not to put a vector or a variable in the wrong place during the manipulation, otherwise, the whole process may numerically explode very quickly.

In this subsection some key modifications in the MMGSD based on the functions defined above are described in the following subsections.

3.2.3.1 The AME

In order to obtain the deltarised output data dy at current time and the deltarised vector array dphi, the vector array phi (φ) and the original output data y at current time are needed. The deltarise function is employed in (35).

$$dphi4\mathrm{y} \;=\; deltarise\left(\left[\mathrm{y}\; phi(iiia)\right],\; Ts\right) \tag{35}$$

where iiia indexes the array for the output vector in phi. Ts is the sampling interval, dphi4y is the deltarised vector array for output, in which the first element is dy, and all other elements are assigned to dphi(iiia).

Similarly input vectors u and e, and the noise vector c are deltarised values for dphi. However, their deltarised values at the current time are not required.

Secondly in the prefilter ztil in RML, the relationship between psi (ψ) and phi (φ) in z transform is expressed as: phi(t) = c(z)*psi(t), or phi(t) = psi(t)+$c_1$psi(t-1)+…+c_{nc}psi(t-n_c), where c is the polynomial coefficients [1, c_1, …, c_{nc}] for noises to improve the property of psi so that the estimator converges more reliable. It is seen that phi(t) is related to psi at both current and previous time. The relationship between psi and phi in delta (δ) transform is expressed as in (36), where the c polynomial is a deltarised version of the coefficients,

$$phi(t) = c(\delta)\cdot psi(t) \tag{36}$$

or its full expression in (37).

$$\delta^{nc-1}phi(t-nc) = \delta^{nc-1}psi(t-nc) + c_1\delta^{nc-2}psi(t-nc) + ... + c_{nc}psi(t-nc) \tag{37}$$

To achieve deltarised psi at current time, this equation is manipulated as shown in (38). It is a two-dimensional array, the number of rows is equal to the size of vectors in phi and the number of columns is equal to the number of terms in the c polynomial.

$$\delta^{nc-1}psi(t-nc) \;=\; \delta^{nc-1}phi(t-nc) \;-\; c_1\delta^{nc-2}psi(t-nc) \;-\ldots-\; c_{nc}psi(t-nc) \tag{38}$$

When using the z transform, (Ljung 1999) makes use of the fact that past values of psi and phi are readily available in the estimator, so that psi(t) can be obtained easily from available data vectors in the estimator. This is because the nature of the data does not change with storage position in the data vector. However, when using the delta transform $\delta^{nc\text{-}1}psi(t\text{ - }nc)$ cannot be obtained using the same procedure, because samples in the data vector are different orders of δ. All these data vectors have to be refilled at each sampling interval.
The vectors in $\delta^{nc-1}psi(t-nc)$ are shown in (39), if, for example, the coefficients array nn is [3 4 2 1 4].

$$-\delta^2 y(t-3)\ldots-\delta^0 y(t-3),\delta^3 u(t-4)\ldots\delta^0 u(t-4),\delta^1\overline{\varepsilon}(t-2)\ \ \delta^0\overline{\varepsilon}(t-2),1,\delta^3 v(t-4)\ldots\delta^0 v(t-4) \quad (39)$$

$-\delta^2 y(t-3)\ldots-\delta^0 y(t-3)$ are obtained by deltarising $-y(t-1)\ldots-y(t-3)$ using deltarise function. The undeltarise function in 0 is also required to firstly fully undeltarise each row of dpsi at previous time to achieve the current time psi(t), e.g., $-y(t-1)\ldots-y(t-3)$ is achieved by fully undeltarising $-\delta^2 y(t-3)\ldots-\delta^0 y(t-3)$. The undeltarise function is employed again but only for a single iteration (*numb* = 1) to obtain dpsi the next time, so this matrix is shifted forward once. The last term (δ^0psi) in the array is then thrown away, so δ^1psi becomes δ^0psi and so on in order to add the new array in front and keep the algorithm consistent.
Finally the vector array phi is updated with the new estimation including the noise vector that is updated by residual error epsilon. We must keep in mind that depsilon is the deltarised version of epsilon, in this case we only have depsilon at current time, thus the undeltarise function is needed for epsilon, as shown in (40).

$$epsilon = undeltarise([depsilon\ dphi(iiic)], Ts, 0) \quad (40)$$

where *dphi*(*iiic*) includes noise vectors at previous time, *iiic* is the index array for noise vectors in *dphi*, Ts is the sampling rate, 0 indicates the full undeltarisation as has been discussed above.

3.2.3.1 The AMP

Similar to the AME both the deltarise and undeltarise functions are required through the system. Unlike the AME, the predicted value y is used for updating the vector array phi, whereas in the AME inputs u, e and output y are obtained from the training data.
To obtain the output data *y*, *dy* is fully undeltarised by employing the undeltarise function shown in (41):

$$y = undeltarise([dy\ \ -dphi(iiia)], Ts, 0) \quad (41)$$

where *dphi*(*iiia*) includes the previous deltarised output vector, *iiia* is the array for the outputs in *dphi*, Ts is the sampling rate, 0 indicates the full undeltarisation is utilized.

4. Results and discussions

In this subsection the MMGSD is evaluated based on two experiments. The data obtained from a two-stage CMOS operational amplifier (op amp) as shown in Fig. 7. The op amp is used in an open-loop configuration.

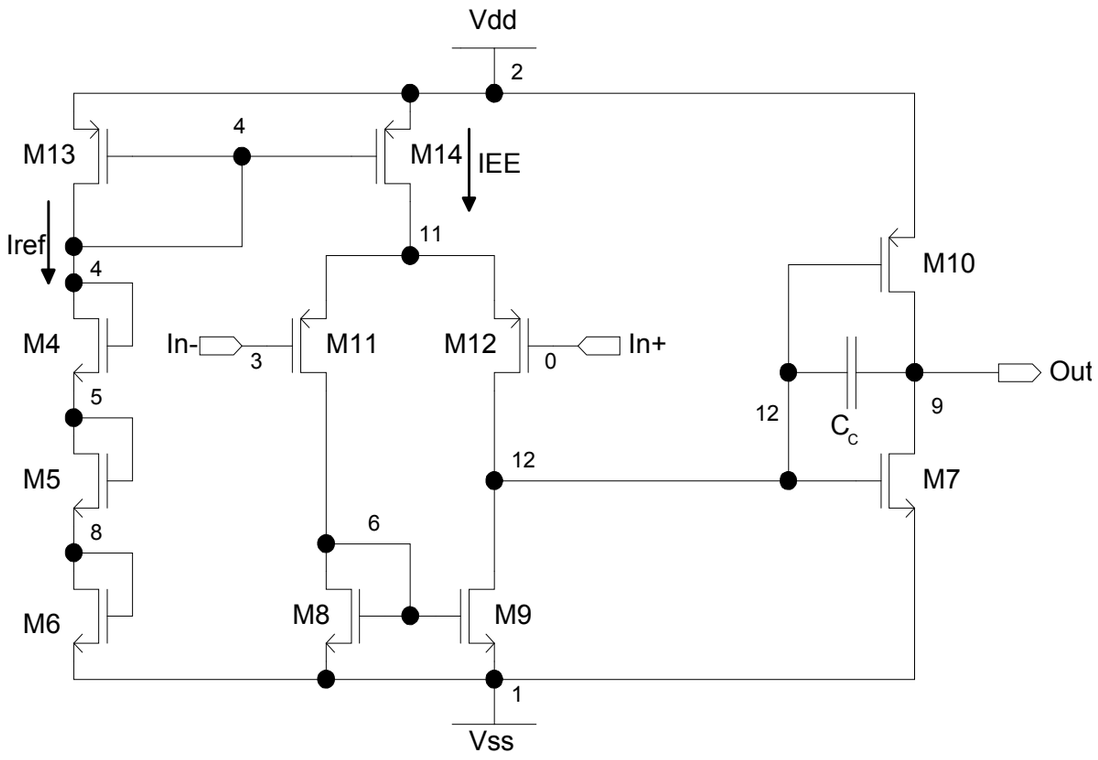


Fig. 7. Schematic of the two-stage CMOS operational amplifier

4.1 A single model detection

The aim of the experiment is to prove that it is able to hunt for known models and converges well. The process follows two steps:

1. The AMP system is applied to a known linear model. Both input data and output data are stored in a text file.
2. The AME generates the model based on these data.

The reason to work in the opposite way is that the AMP is less complicated than the AME and it is easier to find out whether or not the delta operator works well in the MMGSD. The system used in this example is a linear model given in (42).

$$V_o = \frac{-(20s+500)V_{in} + (10s+250)V_{ip} + 250V_{offset}}{s^2 + 20s + 500} \tag{42}$$

Two types of training data are generated from the PRBSG for the MISO AMP: one is a 0.6V, 50Hz square waveform with a 0.12V, 100kHz PRBS superimposed on it for the inverting input, a similar signal but with lower amplitude and frequency is applied to the noninverting input with 14,000 samples. Another training waveform is a 0.2V, 100Hz triangle waveform with a 0.05V, 100kHz PRBS superimposed on it for the inverting input, the second input is a similar signal but with lower amplitude and frequency for the noninverting input with 14,000 samples.

The AME is employed to generate the model seen in (43) with Ts of 10us. It is seen that two models can be matched referring to their coefficients.

$$V_o = \frac{-(20s+500)V_{in} + (10s+250)V_{ip} + 250.02V_{offset}}{s^2 + 20s + 500} \tag{43}$$

4.2 High level fault modeling (HLFM)

In this subsection HLFM is performed to evaluate the models generated using AMG in MATLAB. The training stimulus is a 2.5V, 83.33Hz triangle waveform with a 0.5V, 100kHz PRBS superimposed on it and connects to the inverting input of the open-loop op amp. A similar signal but with lower amplitude and frequency is applied to the noninverting input. The MMGSD generated five models to cover both fault-free and faulty situations. The model thresholds were -2.5V, -1.5V -0.5V, 0.5, 1.5V and 2.5V and number of training samples used for these models were 2263, 2010, 2267, 2452 and 3048, respectively. The generated models are then used to perform a fault simulation of a circuit built from these op amps.

A standard quadratic low-pass filter, shown in Fig. 8, was used to investigate fault simulation with the generated model. The input signal was a 2.0V, 20Hz sinusoid. Transient analysis using SystemVision from Mentor Graphics results from 60ms to 200ms with a step of 0.1ms where used to compare output waveforms.

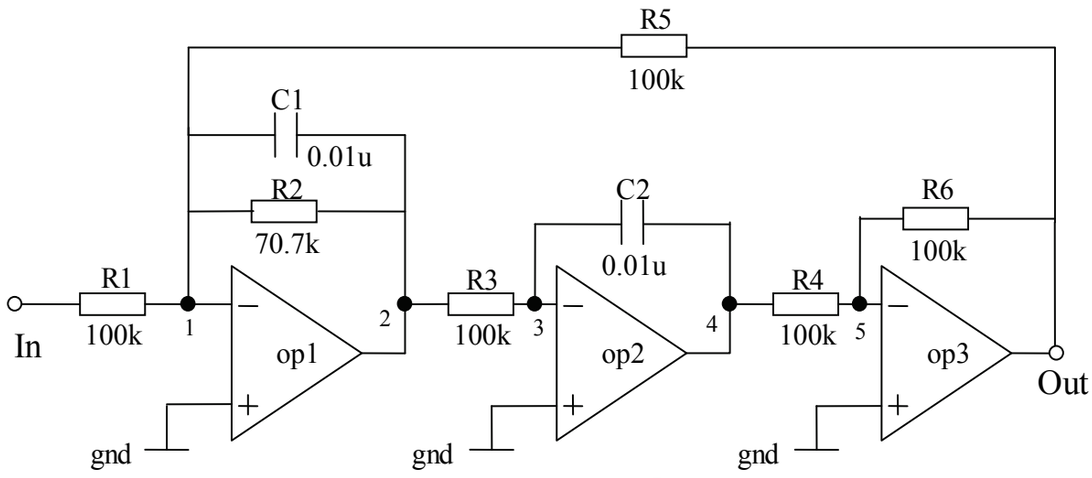


Fig. 8. The quadratic low-pass filter.

Simulation of fault M9_dss_1[1] is shown in Fig. 9. Again the signal becomes nonlinear compared with the fault free case and in this instance the TLFS and HLFS are well matched throughout. TLFS takes 1.297s to complete simulation, and HLFS requires 2.543s

5. Conclusion

In this chapter automated model generation (AMG) techniques using MATLAB were outlined. The models generated were able to generate either SISO or MISO models from transistor level SPICE simulations. They showed the advantage and ability to perform high level fault modeling (HLFM) with the reasonable accuracy compared with transistor level fault simulation (TLFS).

[1] short between drain and source on transistor 9 at op1

In section 1, various model structures were introduced, such as linear-time invariant (LTI) models, or nonlinear-time varying models.

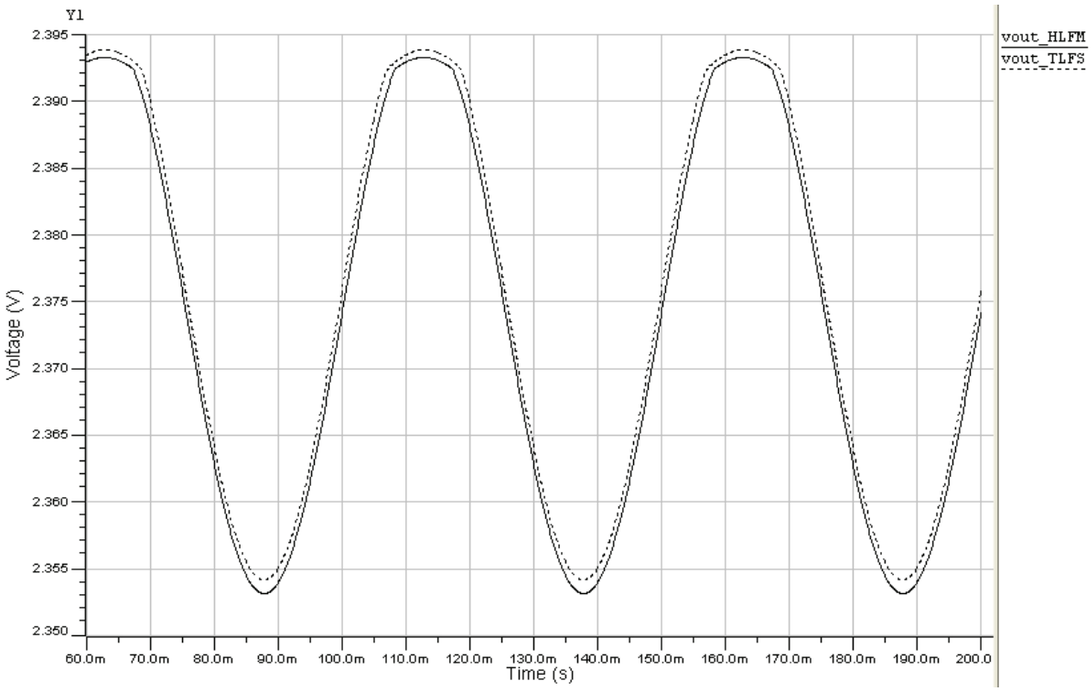


Fig. 9. The output signal from fault M9_dss_1

In section 2, various estimation methodologies were concluded, which consisted of regression, table lookup, neural networks and so on. Particularly regression approaches were focused in this case.

In section 3, an example of AMG termed multiple model generation system using delta operator (MMGSD) was introduced under MATLAB environment. We demonstrated how the delta operator was converted from the discrete-time operator, and how they could be used in the MMGSD.

In section 4, two experiments were implemented. The first one was to demonstrate that the MMGSD was capable of detecting an existing model accurately. The second experiment proved that it could handle the low-pass filter and model nonlinear behaviors accurately.

In summary, AMG approaches using MATLAB are efficient to support high level modeling and simulation, especially useful for high level fault modeling and simulation because of their accuracy.

6. Acknowledgment

The author would like to thank Universiti Teknologi PETRONAS for funding this project. The author wishes to give most special gratitude to his wonderful parents for their encouragement, tolerance and unconditional love during these years in China, United Kingdom and Malaysia. The author shows his appreciation to Miss Zahraa Osman for her contribution on the work. The authors also would like to thank Dr. Ian M. Bell and Dr.

Antony J. Wilkinson from Hull University, United Kingdom for their support and useful discussion throughout the work. And last but not least, the author wishes to thank his friends around of the world.

7. References

Breiman, L. (1996). "*Stacked Regressions*." Machine Learning 24(1): 49-64.

Burden, R. and J. Faires (1985). *Numerical analysis*, Prindle, Weber & Schmidt.

Davalo, É. and P. Naïm (1991). *Neural networks*, Macmillan Education.

Dong, N. and J. Roychowdhury (2003). Piecewise polynomial nonlinear model reduction. *Design Automation Conference Proceedings:* 484-489.

Dong, N. and J. Roychowdhury (2004). Automated extraction of broadly applicable nonlinear analog macromodels from SPICE-level descriptions. *Proceedings of the IEEE Custom Integrated Circuits Conference (CICC2004)*: 117-120.

Dong, N. and J. Roychowdhury (2005). Automated nonlinear macromodeling of output buffers for high-speed digital applications. *Proceedings of the 42nd Design Automation Conference (DAC 2005)*: 51-56.

Feldmann, P. and R. W. Freund (1995). "Efficient linear circuit analysis by Pade approximation via the Lanczos process." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14(5): 639-649.

Grimme, E. J. (1997). *Krylov Projection Methods for Model Reduction*. Electrical engineering department Urbana, Illinios, University Illinios. Doctor of philosophy

Hong, X., P. M. Sharkey, et al. (2003). "A robust nonlinear identification algorithm using PRESS statistic and forward regression." *IEEE Transactions on Neural Networks* 14(2): 454-458.

Li, P. and L. T. Pileggi (2003). NORM: compact model order reduction of weakly nonlinear systems. *Design Automation Conference Proceedings (DAC 2003):* 472-47.

Li, P. and L. T. Pileggi (2005). "Compact reduced-order modeling of weakly nonlinear analog and RF circuits." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(2): 184-203.

Ljung, L. (1999). *System identification: theory for the user*, Prentice Hall PTR.

McConaghy, T., T. Eeckelaert, et al. (2005). CAFFEINE: template-free symbolic model generation of analog circuits via canonical form functions and genetic programming. *Proceedings of Design, Automation and Test in Europe Conference* 2: 1082-1087.

McConaghy, T. and G. Gielen (2005). Analysis of simulation-driven numerical performance modeling techniques for application to analog circuit optimization. *IEEE International Symposium on Circuits and Systems (ISCAS 2005).* 2: 1298-1301

Middleton, R. H. and G. C. Goodwin (1990). Digital control and estimation: a unified approach, Prentice Hall.

Mutnury, B., M. Swaminathan, et al. (2003). Macro-modeling of non-linear I/O drivers using spline functions and finite time difference approximation. Electrical Performance of Electronic Packaging.

Nayfeh, A. H. and B. Balachandran (1995). *Applied nonlinear dynamics: analytical, computational, and experimental methods*, Wiley.

Phillips, J. R. (1998). Model reduction of time-varying linear systems using approximate multipoint Krylov-subspace projectors. *IEEE/ACM International Conference on Computer-Aided Design, ICCAD 98. Digest of Technical Papers.* Santa Clara, CA, USA: 96-102.

Pillage, L. T. and R. A. Rohrer (1990). "Asymptotic waveform evaluation for timing analysis." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9(4): 352-366.

Rewienski, M. and J. White (2001). A trajectory piecewise-linear approach to model order reduction and fast simulation of nonlinear circuits and micromachined devices. *IEEE/ACM International Conference on Computer Aided Design (ICCAD 2001):* 252-257.

Roychowdhury, J. (1999). "Reduced-order modeling of time-varying systems." IEEE *Transactions on Circuits and Systems II: Analog and Digital Signal Processing* 46(10): 1273-1288.

Roychowdhury, J. (2003). Automated macromodel generation for electronic systems. *Proceedings of the 2003 International Workshop on Behavioral Modeling and Simulation (BMAS 2003)*: 11-16.

Roychowdhury, J. (2004). Algorithmic macromodeling methods for mixed-signal systems. *Proceedings of17th International Conference on VLSI Design (VLSID):* 141-147.

Schetzen, M. (1980). *The Volterra and Wiener theories of nonlinear systems*, Wiley.

Simeu, E. and S. Mir (2005). Parameter identification based diagnosis in linear and nonlinear mixed-signal systems. *11th International Mixed-Signals Testing Workshop*, France, TIMA laboratory

Uppal, F. J. and R. J. Patton (2005). "Neuro-fuzzy uncertainty de-coupling: a multiple-model paradigm for fault detection and isolation." *International Journal of Adaptive Control and Signal Processing* 19(4): 281-304.

Vasilyev, D., M. Rewienski, et al. (2003). A TBR-based trajectory piecewise-linear algorithm for generating accurate low-order models for nonlinear analog circuits and MEMS. *Proceedings of Design Automation Conference, (DAC 2003)*: 490-495.

Verbruggen, H. B. and R. Babuška (1999). *Fuzzy logic control: advances in applications,* World Scientific.

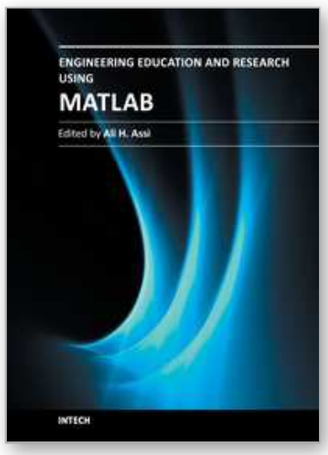
Volterra. (2005). "Volterra Series and Volterra Kernel." Retrieved 06/08, from http://ctas.east.asu.edu/chnam/ASE_Book/Volterra%20Theory.htm

Wilkinson, A. J., S. Roberts, et al. (1991). Real time plant monitoring using recursive identification. *Proceedings of COMADEM 91: The Third International Congress on Condition Monitoring and Diagnostic Engineering Management,* Southampton Institute, Adam Hilger.

Xia, L., I. M. Bell, et al. (2010). "Automated Model Generation Algorithm for High-Level Fault Modeling." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)* 29(7): 1140-1145.

Yang, B. and B. McGaughy (2004). An essentially non-oscillatory (ENO) high-order accurate adaptive table model for device modeling. P*roceedings of the 41st Design Automation Conference* (DAC 2003), San Diego, CA, USA 864-867

Zhang, Q. J. and K. C. Gupta (2000). *Neural networks for RF and microwave design*, Boston: Artech House.



Engineering Education and Research Using MATLAB

Edited by Dr. Ali Assi

ISBN 978-953-307-656-0

Hard cover, 480 pages

Publisher InTech

Published online 10, October, 2011

Published in print edition October, 2011

MATLAB is a software package used primarily in the field of engineering for signal processing, numerical data analysis, modeling, programming, simulation, and computer graphic visualization. In the last few years, it has become widely accepted as an efficient tool, and, therefore, its use has significantly increased in scientific communities and academic institutions. This book consists of 20 chapters presenting research works using MATLAB tools. Chapters include techniques for programming and developing Graphical User Interfaces (GUIs), dynamic systems, electric machines, signal and image processing, power electronics, mixed signal circuits, genetic programming, digital watermarking, control systems, time-series regression modeling, and artificial neural networks.

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen