ModSpec: An Open, Flexible Specification Framework for Multi-Domain Device Modelling

David Amsallem and Jaijeet Roychowdhury
University of California, Berkeley

***Abstract*—We describe ModSpec, a MATLAB/Octave based specification format suitable for modelling devices across a wide variety of application domains, including circuits, optics, fluidics and biology. The ModSpec format and associated API are centered around describing the nonlinear differential equations at the core of any device model. The format is open, general and easy to use, and is supported by toolchains that translate and automatically differentiate models, set up equations for systems of interacting devices, and provide simulation facilities. We illustrate the use of ModSpec for modelling semiconductor, photovoltaic, fluidic and neuronal devices and systems.**

I. Introduction

Specifying and incorporating "device models"[1] correctly and speedily is crucial for effective modelling and simulation. This is true not only in electronics, but in many other domains, including large-scale energy distribution systems and systems biology. Device models of interest can range from simple ones with a few equations relating inputs and outputs directly, to complex ones with thousands of lines of code describing equations with internal state and involving hundreds of device parameters [1], [2]. Simple, non-redundant specification formats, especially ones that make it easy to test and validate models incrementally, can greatly facilitate speedy and error-free device model development.

For microelectronic devices, the model coding API (in the C language) of the open-source simulator SPICE3 [3]–[5] has long served as a de-facto standard for specifying device models. While SPICE3's API and coding conventions have the great advantage of being open, their use for model specification suffers from serious disadvantages, including lack of modularity with respect to numerical algorithms[2], lack of automatic differentiation and suitability for only one simulator. Efforts over the past decade to overcome these limitations have resulted in the Verilog-A Compact Model Extension (VA-CME) standard [8] for model specification. VA-CME is supported by tools [9], [10] that translate specifications into source and executable code targeted for specific simulators.

While the VA-CME specification language is a great improvement over SPICE3 and is being adopted in the microelectronics industry, it has limitations in terms of its suitability for multiple physical domains, ease of use and generality:

1) VA-CME is designed for microelectronic devices. At its core, it relies on electrical network concepts (like nodes and branch currents), which may not be suited to devices from other domains – a general form for which are nonlinear differential equation systems that do not necessarily derive from network/graph structures. As such, VA-CME is not a natural choice for modelling devices in multi-domain systems (*e.g.*, systems that include biological and micro-electronic elements).
2) In many communities – including parts of the semiconductor modelling community – persons interested in writing device models are not familiar with Verilog-A, VA-CME or similar language constructs. The learning curve is a deterrent to the model development process.
3) Currently-available VA-CME toolchains are predominantly commercial and closed. Few, if any, truly open and functional translators are available, especially for open simulators. This has impeded the adoption of VA-CME to an extent, especially in academia.
4) VA-CME does not provide general support for stochastic modelling – *e.g.*, it is not currently possible to model non-stationary noise sources in models.

In this paper, we present **ModSpec**, a MATLAB/Octave based model specification format, together with associated tools for translation and simulation. Rather than being tied to any single application domain, **ModSpec** is centered around the *equations* that constitute the substance of any device model. The specification deals directly with the underlying *nonlinear differential equations* that prescribe relationships between the inputs and outputs of a device while accounting for internal state. **ModSpec** comes with "equation engines" that automatically set up equation systems for collections of interacting devices, and a MATLAB/Octave simulator for such equations. **ModSpec** allows implicit forms for differential equations, hence inputs and outputs do not have to be identified explicitly; this provides flexibility in modelling and in devising different types of equation engines. Any outputs that are available explicitly may be so specified, and can be used by equation engines to form more compact equation systems.

The **ModSpec** specification is free of redundancies; automatic differentiation (in MATLAB/Octave, using the `vecvalder` package developed as part of this work) is used to generate the derivatives needed by solution algorithms. A comprehensive MATLAB/Octave API, used for accessing the device, is an integral part of **ModSpec**. The API also provides facilities for device-specific aids to algorithms, such as initialization and limiting [11] for Newton-Raphson – these are cleanly separated from the device's differential equations and may be taken advantage of by any simulator.

The use of MATLAB/Octave for **ModSpec** provides ease-of-use and accessibility advantages, making it possible for users with minimal programming background, and from diverse communities, to develop device models quickly and correctly. The interactivity of the MATLAB/Octave environments is important in enabling incremental model development, refinement and validation. **ModSpec**'s API makes it possible to test and debug device models directly, without the need for invoking equation engines or simulators, which are layered modularly "on top of" the **ModSpec** specification. Being fully Octave compatible [12], **ModSpec** is immediately accessible to individuals and groups without the resources to buy MATLAB licenses. **ModSpec** and the equation and simulation engines we provide are open and freely usable and will be released publicly.

To illustrate its utility, we present example **ModSpec** descriptions of a semiconductor model, photovoltaic cells (which span microelectronics and optics), fluidic devices and compartmental neuronal structures. Using equation engines we have developed for electronic networks and neuronal structures, we build equation systems for networks of devices. We simulate these equations, finding nonlinear quiescent steady states and transient responses, using a MATLAB/Octave based simulator we have developed. The case of neuronal structures is especially interesting as it illustrates the multiphysics capabilities of the **ModSpec** framework: both electric potential propagation and dynamics of concentration of ions are handled.

[1]Examples of "devices" include the well-known BSIM [1] and PSP [2] MOS transistor models in microelectronics; beams and cantilevers in mechanics; and neurons (or portions of neurons) in systems biology.

[2]Algorithms such as the Newton-Raphson method and the Backward-Euler, Trapezoidal and Gear methods for numerical integration [6], [7], are hard-coded into each device model.

978-1-4577-1400-9/11/$26.00 ©2011 IEEE

The **ModSpec** framework is also a significant enabler for advanced simulation and macromodelling algorithms: we illustrate this by applying nonlinear macromodelling techniques to reduce **ModSpec** descriptions of compartmental neuronal structures. The resulting reduced models also conform to the **ModSpec** API, hence can be used as drop-in replacements for the original **ModSpec** device models. We present simulation results comparing such reduced models with the originals.

The remainder of the paper is organized as follows. In Section II, we describe the core of **ModSpec**, *i.e.*, how the nonlinear differential equations of device models are organized to enable model specification, and provide a simple illustrated example. We also describe **ModSpec**'s algorithm-specific aids. In Section III, we describe toolchains that take **ModSpec** device specifications, set up equations for systems of interacting devices, and simulate them. An important component of such toolchains is the `vecvalder` package, which enables automatic differentiation in MATLAB/Octave. In Section IV, we present examples of the use of **ModSpec** for photovoltaic, fluidic and neuronal devices, and illustrate its use for facilitating nonlinear reduced order modelling.

II. Model Specification

A. Framework

The presentation of the model specification (**ModSpec**) framework is deliberately made general and abstract in this section so as to handle the widest possible number of devices from multiple domains. In this paper, the usefulness of such an approach will be illustrated with multiple examples pertaining to circuit simulation, as well as neuron modeling.

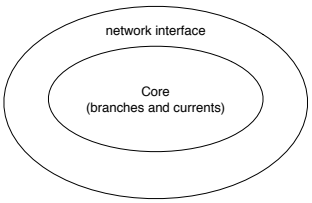


Fig. 1. Core and network interface layers for a given device.

As depicted in Fig. 1, we make a distinction between the "core" of the model and its "network interface" layer. At its core, the device is assumed to provide constitutive relationships between n_{ext} input/ouput quantities. In a resistor, for example, the I/O quantities of interest are its branch voltage v_b and branch current i_b, hence $n_{\text{ext}} = 2$. In general, for electrical devices, there will be $n_{\text{ext}} = 2n$ such quantities (voltages and currents) where n denotes the number of external branches, as depicted in Fig. 2[3]. In essence, the device specifies n equations that relate the external I/O quantities – *e.g.*, for a resistor, the equation $v_b - Ri_b = 0$.

Observe that a device's *branch* voltages/currents/charges/fluxes are not immediately related to network-level quantities such as node voltages. The network interface layer (Fig. 1) provides relationships that connect core I/O and network-level quantities – *e.g.*, for a resistor connected between nodes with node voltages v_+ and v_-, $v_b = v_+ - v_-$, $i_+ = i_b$, $i_- = -i_b$, with i_+ and i_- being the node currents due to the resistor. Separating the core and network interface layers is useful for cases where devices do not necessarily connect to networks. For example, two resistors augmented by *any* two other equations that lead to a well-formed square system of equations in the resistors' I/O variables can be modelled using **ModSpec**.

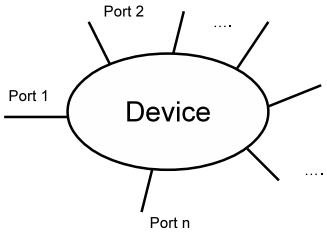


Fig. 2. Device with n external branches.

In the remainder of this section, electrical devices will be used for illustration; however, the framework is flexible enough to apply to many other domains (see Section IV). In general, there will be n differential-algebraic equations relating the $2n$ external quantities. There is no a priori assumption whatsoever in the **ModSpec** regarding which of these external quantities are inputs and which are outputs. We assume that $l \in [0,n]$ of the external I/O quantities are outputs, specified *explicitly* by l equations.

Those quantities are gathered in a vector $\mathbf{z} \in \mathbb{R}^l$ and the remaining $2n-l$ external unknowns stored in a vector $\mathbf{x} \in \mathbb{R}^{2n-l}$. In addition to these external quantities, $m \geq 0$ internal unknowns, stored in a vector $\mathbf{y} \in \mathbb{R}^m$ can be defined as well. Note that for some devices, it is possible to write their characteristic equation either as an explicit or implicit equation. For instance, for a resistor, its equation can be written explicitly as $I = \frac{V}{R}$ or implicitly as $f(V,I) = V - IR = 0$.

The device may also be parameterized by N_p parameters stored in $\boldsymbol{\mu} \in \mathbb{R}^{N_p}$. In the general case, there will be l explicit equations and $n_i = n - l + m$ implicit equations. In this paper, Differential Algebraic Equations (DAEs) are considered. These can be written in a general form as

$$\frac{d\mathbf{q}_e}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_e(\mathbf{x},\mathbf{y},t;\boldsymbol{\mu}) = \mathbf{z} \quad (1)$$

$$\frac{d\mathbf{q}_i}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_i(\mathbf{x},\mathbf{y},t;\boldsymbol{\mu}) = \mathbf{0}_{n_i}, \quad (2)$$

where the indices e and i respectively refer to terms appearing in the explicit and implicit equations.

In some cases, the time dependence in the functions $\mathbf{f}_i$ and $\mathbf{f}_e$ can be separated. One can then write the device DAEs as

$$\frac{d\mathbf{q}_e}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_e(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{b}_e(t;\boldsymbol{\mu}) = \mathbf{z} \quad (3)$$

$$\frac{d\mathbf{q}_i}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_i(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{b}_i(t;\boldsymbol{\mu}) = \mathbf{0}_{n_i}. \quad (4)$$

Noise terms may also be added to the DAE equations as follows:

$$\frac{d\mathbf{q}_e}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_e(\mathbf{x},\mathbf{y},t;\boldsymbol{\mu}) + \mathbf{M}_e(\mathbf{x},\mathbf{y};\boldsymbol{\mu})\mathbf{n}_e(t;\boldsymbol{\mu}) = \mathbf{z} \quad (5)$$

$$\frac{d\mathbf{q}_i}{dt}(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{f}_i(\mathbf{x},\mathbf{y},t;\boldsymbol{\mu}) + \mathbf{M}_i(\mathbf{x},\mathbf{y};\boldsymbol{\mu})\mathbf{n}_i(t;\boldsymbol{\mu}) = \mathbf{0}_{n_i}. \quad (6)$$

The matrices $\mathbf{M}_i$ and $\mathbf{M}_e$ denote the noise stationary component matrices and the vectors $\mathbf{n}_e$ and $\mathbf{n}_i$ are the stationary noise sources.

The model specification then defines the nonlinear operators $\mathbf{q}_e$, $\mathbf{q}_i$, $\mathbf{f}_i$, $\mathbf{f}_e$, $\mathbf{b}_i$, $\mathbf{b}_e$, $\mathbf{M}_e$, $\mathbf{M}_i$, $\mathbf{n}_e$ and $\mathbf{n}_i$. Appendix 1 presents a typical **ModSpec** class for an electronic device.

Example: To illustrate this framework, a simple example, made of a diode with a series resistor and depicted in Fig. 3, is considered. In this case the external and internal unknowns can be chosen as

$$\mathbf{z} = [I_{pn}], \quad \mathbf{x} = [V_{pn}], \quad \mathbf{y} = [V_{in}]. \quad (7)$$

The vector of parameters $\boldsymbol{\mu} = [R, I_s, T]^T$ contains the resistance value, the saturation current in the diode as well as the temperature of usage of the diode. The DAE operators are then

$$\mathbf{q}_e(\mathbf{y};\boldsymbol{\mu}) = \mathbf{q}_{\text{depletion}}(\mathbf{y};\boldsymbol{\mu}) + \mathbf{q}_{\text{diffusion}}(\mathbf{y};\boldsymbol{\mu}) \quad (8)$$

$$\mathbf{q}_i(\mathbf{y};\boldsymbol{\mu}) = -\mathbf{q}_{\text{depletion}}(\mathbf{y};\boldsymbol{\mu}) - \mathbf{q}_{\text{diffusion}}(\mathbf{y};\boldsymbol{\mu}) \quad (9)$$

where $\mathbf{q}_{\text{depletion}}$ and $\mathbf{q}_{\text{diffusion}}$ respectively denote the diffusion and depletion caps between the node i and n. The remaining nonlinear operators are

$$\mathbf{f}_e(\mathbf{y};\boldsymbol{\mu}) = \mathbf{diode}(\mathbf{y};\boldsymbol{\mu}), \quad \mathbf{f}_i(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) = \frac{\mathbf{x}-\mathbf{y}}{R} - \mathbf{diode}(\mathbf{y};\boldsymbol{\mu}), \quad (10)$$

where $\mathbf{diode}(\mathbf{y};\boldsymbol{\mu}) = I_d$ denotes the equation describing the current

[3]This is for the case of no charges/fluxes; when charges and fluxes are present, n_{ext} can be $> 2n$.

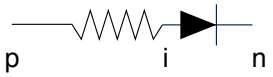


Fig. 3. Diode with series resistor.

I_d through the diode in function of the applied potential.

B. Additional Capabilities

1) Device-Specific Algorithm Support Features: In addition to the definition of the variables of interest and the DAE operators, the proposed framework also contains additional features needed in practice by simulators using the models. Algorithm aids for solving nonlinear sets of equations involving certain specific devices may be needed. Among those, in the case of the Newton-Raphson method, one can cite the computation of initial guesses as well as limiting routines for nonlinear functions containing exponential terms, such as the **diode** function.

2) Automatic Differentiation: The ability to compute functions derivatives and Jacobians is critical for sensitivity analyses as well as numerically solving nonlinear systems of equations using the Newton-Raphson method. Forward Automatic Differentiation (AD) is a powerful tool to achieve this goal. The recently developed **MATLAB** scalar AD `valder` class [13] presents an innovative AD approach through the use of classes. It is however limited by the fact that it has not been designed to support automatic differentiation of vector-valued function. This is an issue in the context of simulation, when variables and sets of equations originate from the discretization of continuous domains. In those cases, a manual definition of each variable as a `valder` object is inefficient and impractical.

This fact has motivated the authors of this paper to modify and extend the `valder` class to support automatic differentiation of vector valued functions. The resulting `vecvalder` class has been designed so that it is compatible with Octave as well. Appendix 2 presents some members of the `vecvalder` class. `vecvalder` is currently ready for public release.

III. Model Translation, Equation Engines and Simulation

A. Library of Models

Model compilation relies on the existence of a library of models in the **ModSpec** framework outlined in Section II. The current library is implemented in **MATLAB**/Octave as these are de facto widely used in the engineering community. It is expected that this library will be translated in C in the future, enabling, after compilation, the use of dynamically loaded libraries of models.

The models contain all the required information needed to build the global operators of interest needed to perform either simulations, model analysis or construction of reduced-order models. In the case of electric circuits, such a library may contain basic two-port components such as resistors, capacitors, diodes, generators as well as more complex multi-port components such as semiconductors. In the case of neuronal systems, the library contains specific models for equations characterizing the potential propagation as well as the ions kinetics in the neuronal cells. There are several hierarchical models describing such kinetics and as a result, a library containing many of these models is a useful tool for the comparison of such models when coupled with the potential propagation equations. Hence, this framework enables an inter-operable library of models from different domains, which is important in several fields such as bio-electronics.

B. Parser

A given network is defined as a netlist containing the connectivity of each component as well as its nature. By reading this netlist, the parser subsequently builds a data structure containing the graph connectivity of the network as well as the description of each device.

A key enabler to a variety of algorithm support is the construction of an Application Programming Interface (API) to the data structure. Such an API contains pointers to the location of the specific **ModSpec** files in the library for each network component, the graph connectivity, a function returning all the components in the data structure corresponding to a specific device, to name just a few. In the case of circuits, the API can return the list of all non-voltage controlled devices, which is very useful in Modified Nodal Analysis (MNA), for instance. For neuron modeling, the API can also return the properties of the distal and proximal compartments of each branch, such as the cable radius and the compartment length. These properties are then used when setting the equations to define the junction conditions in the network.

C. Equation Engine

As illustrated in Fig. 4, the equation engine builds the global DAE operators, that is at the network level, in the form of anonymous functions. This involves mapping the local variables at the devices level to global variables. The equations are then usually built by applying conservation laws such as Kirchhoff Current Law (KCL) for instance. For circuits, the equation engine may use Nodal Analysis (NA), Tableau Equations or Modified Nodal Analysis. One should notice that the proposed model specification is completely agnostic with respect to the method used to build the system of equations. In the case of neurons, the equations are built by enforcing potential continuity and KCL at the junction between branches as well as boundary conditions.

As described in the previous subsection, the API to the data structure is an essential tool for setting the equations. An abridged list of API routines is reported in Appendix 3 for the cases of electronic circuits and neuronal networks simulations. Some of the API are common to both cases and as a result, for each application, a derived class is defined from a common class. A full list of API routines will be published openly in the near future.

D. Utilization of the Compiled Model

1) Simulator: Simulators can use the compiled model to perform DC/QSS, AC, transient ODE/DAE time-stepping, periodic steady state simulations with noise terms or not, to name just a few. Since the DAEs are given in the form of anonymous functions, it is straightforward to extract the expressions for the residuals and Jacobians needed for each kind of simulation. In the case of transient simulations, the residuals and Jacobians depend on the time integrating scheme considered. However, because the **ModSpec** framework is agnostic with respect of the model usage, it is a very simple task to link it with any general ODE simulator.

2) Reduced-Order Model Constructor: Such a compiled model is also particularly useful in the context of model order reduction (MOR), where the reduced-order model constructor has to often repeatedly query solutions or operators of the underlying high-fidelity model. For instance, linear [14] and nonlinear [15] model reduction methods using Krylov-based moment matching techniques, as well as the Gappy POD method [16], require computing products of Jacobian matrices of the dynamical system with vectors. The Discrete Empirical Interpolation [17] and Gappy methods require evaluating parts of the nonlinear functions $\mathbf{f}_i$ and $\mathbf{f}_e$ for multiple state vectors and, as a result, relies heavily on an efficient framework for evaluating a few components of the high-fidelity model. The **ModSpec** framework can provide an efficient response to these demands for all of the methods enumerated above.

IV. Applications

A. BSIM3 Model

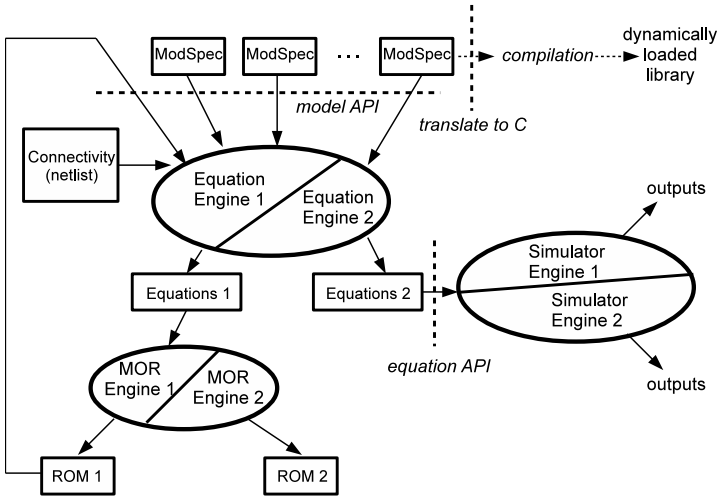


Fig. 4. Schematic description of the model compilation procedure.

The first example is an inverter electronic system as shown in Fig. 5. This circuit is composed of two MOSFETs, one which is n-type and the other p-type, as well as voltage sources. A single **ModSpec**, based on the BSIM3v3 model [1] handles both the nMOSFET and pMOSFET types. This **ModSpec** contains 8 external quantities among which 4 external unknowns and two internal unknowns. 406 parameters define the model equations. A sinusoidal input voltage V_{in} of amplitude 0.4 V, DC value 0.4 V and frequency 10 MHz is applied and the output voltage V_{out} of the model simulated. The results are reported in Fig. 6. The reader can note the effectiveness of the **ModSpec** framework to handle complex models such as BSIM3v3.

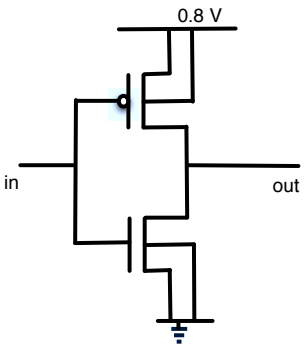


Fig. 5. Inverter circuit.

B. Solar Photovoltaic Cell

The second example considered in this section is a solar cell circuit. This small circuit, first described in [18] is composed of a current source in parallel to a diode as shown in Fig. 7. Three models are subsequently considered in the **ModSpec** format: one for the resistor, one for the photovoltaic current source and one for the diode. The diode **ModSpec** contains the PNJLIM [11] Newton limiting algorithm. For each given irradiance $G \in \{0.25, 0.5, 0.75, 1\}$ sun, varying the applied potential V_a leads to one of the *I-V* curve shown in Fig. 8. There is exact agreement with the results reported in [18]. Furthermore, the reader should note that unlike the code implemented in [18], specifying separately each device in the circuit outside the Newton-Raphson algorithm leads to a flexible and general implementation, enabling the use of these models in other contexts such as reliability assessment for instance [19]. Our approach contrasts with the one in SPICE3 where every algorithm (Newton-Raphson, trapezoideal rule, Gear 2, to name just a few) has to be recoded in every model.

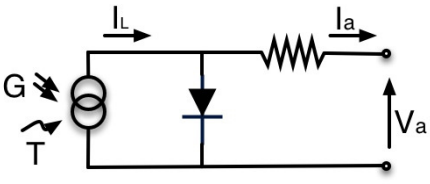


Fig. 7. Solar PV circuit.

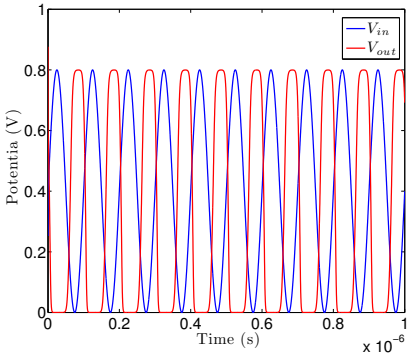


Fig. 6. Inverter circuit output under a sinusoidal input.

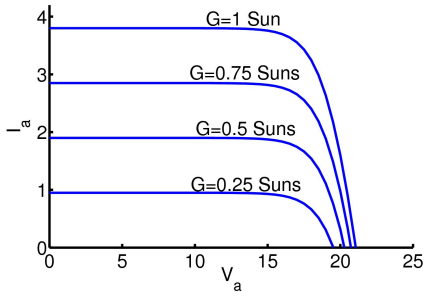


Fig. 8. Solar PV circuit operating at $T = 25°C$ under various irradiances.

C. Fluidic Device

The third example is a fluidic device modeled by the nonlinear Burger's equation. Burger's equation is a nonlinear PDE used to model traffic flow or the transport of a substance, such as a gas. Its inviscid form is

$$\frac{\partial U}{\partial t} + U\frac{\partial U}{\partial x} = g(x,t) \tag{11}$$

where $U(x,t)$ is the quantity of interest, and $g(x,t)$ is a source term. The PDE is defined in the domain $(x,t) \in \mathscr{D} = [0,L] \times [0,T_f]$. Initial and boundary conditions are also associated to the PDE:

$$U(x,0) = U_i(x) \tag{12}$$

and

$$U(0,t) = U_b(t), \tag{13}$$

assuming that $U(0,t) > 0,\ \forall t \geq 0$.

The PDE can then be discretized in space using Godunov's finite volumes method [20], [21]., resulting in the following set of nonlinear ODEs.

$$\frac{d\mathbf{u}}{dt} + \mathbf{f}(\mathbf{u}) = \mathbf{g} \tag{14}$$

where the state vector $\mathbf{u}(t) \in \mathbb{R}^N$ contains the values $u_i,\ i = 1,\cdots,N$ respectively approximating the values $U(x_i,t),\ \ i = 1,\cdots,N$ with $x_0 = 0 < x_1 < \cdots < x_N = L$. For simplicity, here, $x_{i+1} - x_i = \Delta x = \frac{L}{N}$. Similarly, $\mathbf{g}(t) \in \mathbb{R}^N$ contains the values $g_i(t) = g(x_i,t)$. The nonlinear function $\mathbf{f}$ is defined by the finite volumes approximation as

$$f_i = [\mathbf{f}(\mathbf{u})]_i = \frac{1}{\Delta x}(F_{i+\frac{1}{2}} - F_{i-\frac{1}{2}}) \tag{15}$$

where

$$F_{i+\frac{1}{2}} = \begin{cases} f(u_i) & \text{if } \ u_i > -u_{i+1} \\ f(u_{i+1}) & \text{if } \ u_i \leq -u_{i+1}. \end{cases} \tag{16}$$

There are 2 external quantities in the **ModSpec**, namely $u_{inlet} = u_0$ and $u_{outlet} = u_N$. Hence $2n = 2$. Since there are no explicit outputs, $l = 0$ and there are $2n - l = 2$ external unknowns. As a result, there are $m = N - 1$ internal unknowns, namely $u_i,\ i = 1,\cdots,N-1$. One can then check that there are indeed $n - l + m = 1 - 0 + N - 1 = N$ implicit equations and no explicit equations.

Two parameters define the system: the length of the domain L as well as the number of points N in the discretization of the equations. The following vector of parameters is then defined

$$\boldsymbol{\mu} = [L,N]^T. \tag{17}$$

Using the notations from Section II, here,

$$\mathbf{x} = \begin{bmatrix} u_0 \\ u_N \end{bmatrix}, \quad \mathbf{y} = \begin{bmatrix} u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}, \quad \mathbf{z} = []. \tag{18}$$

The implicit set of equations is

$$\frac{d\mathbf{q}_i}{dt}(\mathbf{x},\mathbf{y}) + \mathbf{f}_i(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) + \mathbf{b}_i(t;\boldsymbol{\mu}) = 0, \tag{19}$$

where

$$\mathbf{q}_i(\mathbf{x},\mathbf{y}) = \begin{bmatrix} \mathbf{y} \\ \mathbf{x}_2 \end{bmatrix}, \tag{20}$$

and

$$\mathbf{f}_i(\mathbf{x},\mathbf{y};\boldsymbol{\mu}) = \frac{\mu_2}{\mu_1}\Delta\mathbf{F}(\mathbf{x},\mathbf{y}) = \frac{\mu_2}{\mu_1}\left[F_{\frac{3}{2}} - F_{\frac{1}{2}},\cdots,f(\mathbf{x}_2) - F_{N-\frac{1}{2}}\right]^T \tag{21}$$

with

$$F_{\frac{1}{2}} = f(\mathbf{y}_1) + \mathbf{1}_{(\mathbf{x}_1 > -\mathbf{y}_1)}(f(\mathbf{x}_1) - f(\mathbf{y}_1)) \tag{22}$$

$$F_{i+\frac{1}{2}} = f(\mathbf{y}_{i+1}) + \mathbf{1}_{(\mathbf{y}_i > -\mathbf{y}_{i+1})}(f(\mathbf{y}_i) - f(\mathbf{y}_{i+1})),\ \forall i \geq 1 \tag{23}$$

and

$$\mathbf{b}_i(t;\boldsymbol{\mu}) = -\left[g\left(\frac{\mu_1}{\mu_2},t\right),\cdots,g\left(N\frac{\mu_1}{\mu_2},t\right)\right]^T. \tag{24}$$

Fig. 9 reports snapshots of the the state vector $\mathbf{u}(t)$ at various times t, as simulated using **ModSpec** together with `vecvalder` to compute the Jacobians needed in the nonlinear implicit time-stepping. Very good agreements can be seen when compared with the results presented in [21].

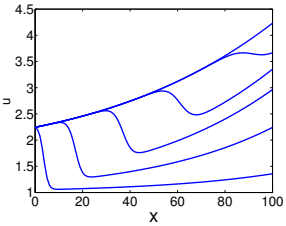


Fig. 9. Snapshots of the evolution of $\mathbf{u}(t)$ at various times t_i with Burger's equation.

D. Neuron Modeling

1) Generalities: Neurons are generally (but not always) composed of three parts; a cell body, also called soma, an axon and dendrites attached to the soma, as illustrated in Fig. 10 [22]. The soma integrates the input currents injected in its dendrites by other neurons, and, under certain conditions, will itself produce an electric current that propagates in the axon and eventually will serve as an input for other neurons. This motivates, as done in the remainder of this section, studying a network made of the dendritic tree together with the soma. Depending on the potential integration at the soma, the potential then propagates or not in the axon.

2) The Hodgkin-Huxley Model: The Hodgkin-Huxley neuron model [23] is a multi-compartmental model composed of a set of coupled nonlinear partial differential equations describing the propagation of current inside the dendrites, soma and axon. The first set of equations, also called cable equation, describes the temporal and spatial potential variation in the neuron elongated structures (dendrites and axon)

$$\begin{aligned} \frac{d}{2R_i}\frac{\partial^2 V}{\partial x^2} &= C_m\frac{\partial V}{\partial t} + G_{Na}m^3h(V - E_{Na}) \\ &\quad + G_K n^4(V - E_K) + G_L(V - E_L) + I(x,t). \end{aligned} \tag{25}$$

Boundary conditions are associated with the potential in the soma, considered to be uniform at a given time:

$$\begin{aligned} \frac{d^2\pi}{r_{\text{soma}}R_i}\frac{\partial V_{\text{soma}}}{\partial x} &= C_m\frac{\partial V_{\text{soma}}}{\partial t} + G_{Na}m^3_{\text{soma}}h_{\text{soma}}(V_{\text{soma}} - E_{Na}) \\ &\quad + G_K n^4_{\text{soma}}(V_{\text{soma}} - E_K) + G_L(V_{\text{soma}} - E_L) + I_{\text{soma}}(t). \end{aligned} \tag{26}$$

The second set of equations models the dynamics of concentration of Na^+, K^+ and Cl^- (leak) ions across the cell membrane

$$\frac{\partial m}{\partial t} = \alpha_m(V)(1-m) - \beta_m(V)m \tag{27}$$

$$\frac{\partial h}{\partial t} = \alpha_h(V)(1-h) - \beta_h(V)h \tag{28}$$

$$\frac{\partial n}{\partial t} = \alpha_n(V)(1-n) - \beta_n(V)n. \tag{29}$$

$V(x,t)$ denotes the local membrane potential, $V_{\text{soma}}(t)$ the potential at the soma, $I(x,t)$ and $I_{\text{soma}}(t)$ the injected synaptic current, $m(x,t)$, $h(x,t)$ and $n(x,t)$ are gating variables for the ion channels corresponding to respectively the Na^+ ions (m and h) and K^+ ions (n). The expressions for $\alpha_i(V)$ and $\beta_i(V)$, $i \in \{m,h,n\}$ as derived originally by Hodgkin and Huxley can be found in [23]. In this section, the physical quantities of interest are the ones found experimentally by Hodgkin and Huxley, namely a specific membrane capacitance $C_m = 10^{-2}$ F/m^2, an axial resistivity $R_i = 0.354$ ohm.m, maximal conductances $G_{Na} = 1.2\ 10^3$ ohm^{-1}m^{-2}, $G_K = 3.6\ 10^2$ ohm^{-1}m^{-2} and $G_L = 3$ ohm^{-1}m^{-2}, and equilibrium potentials $E_{Na} = 0.115$

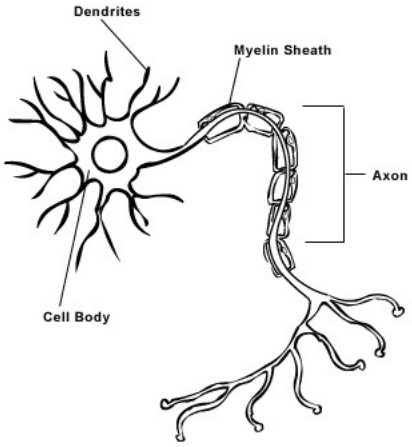


Fig. 10. Schematic representation of a neuron structure.

TABLE I
COMPARISON OF CPU TIMINGS WITH AND WITHOUT THE **MODSPEC** FRAMEWORK

	with the **ModSpec** framework	without the **ModSpec** framework
CPU Time	160 s	138 s

V, $E_K = -0.012$ V and $E_L = 0.010613$ V. Dendrites with radius $d = 0.476\ 10^{-3}$ m and length $L = 5\ 10^{-2}$ m is here considered. The soma has a radius $r_{\text{soma}} = 2\ 10^{-3}$ m. The space and time derivative are approximated using the approach first developed in [24]. Every dendrite is discretized in space using 1400 compartments, resulting in a dynamical system having a total of 5600 degrees of freedom per dendrite.

The underlying equations as well as the parameters of interest are here again all defined using the **ModSpec** framework. This enables changing these parameters in a straightforward fashion. The cable equation model is defined by one **ModSpec** while the equations relative to the ions concentration dynamics are defined in a different **ModSpec**. This can allow easily switching between models of different fidelities. In the present case, the ions model could be straightforwardly replaced by a more complex model incorporating more complex ions kinetics such as the A-type K^+ current developed by Connor and Stevens [25] for instance.

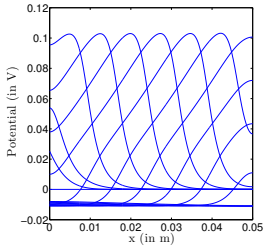
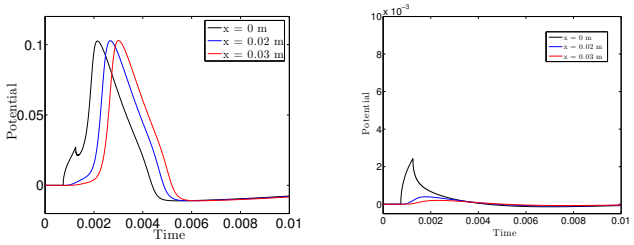


Fig. 11. Several spatial snapshots of the potential propagation in a single dendrite.

To illustrate the model's predictive capabilities, an input pulse of amplitude $I_{max} = 8.418\ 10^{-3}$ A is applied at one end of the dendrite between times $t_i = 0.75\ 10^{-3}$ s and $t_f = 1.25\ 10^{-3}$ s. The propagation of the action potential is illustrated in Fig. 11 where multiple snapshots of the spatial distribution of the potential are depicted. The time histories of the potentials at three different locations of the dendrite are plotted in Fig. 12(a). The black curve represents the time history of the potential at the location of the current injection. If the amplitude of the applied current is below a certain threshold, then no propagation occurs as illustrated in Fig. 12(b). In this case $I_{max} = 10^{-3}$ A, all other parameters being equal. This illustrates the nonlinear nature of the Hodgkin-Huxley model.



(a) Propagation (b) No propagation

Fig. 12. Transient potential propagation in a dendrite

In order to assess the efficiency of the **ModSpec** framework, the CPU timings are compared when the framework is not used to perform the simulations. As reported in Table I, using the framework adds 16% extra computational time to the simulation for the benefit of generality and utilization of the framework with any arbitrary network. It is also expected that this overhead cost will disappear when the framework is ported into C, allowing the use of pre-compiled **ModSpec** libraries.

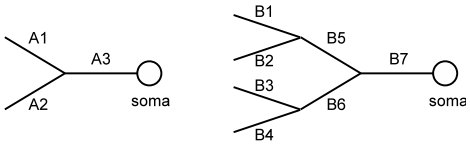


Fig. 13. Dendritic trees considered.

3) Neuron Networks with Multiple Branches: Neuronal networks made of multiple branches are the focus of the remainder of this paper. More specifically, networks constituted of a dendritic tree and a soma will be considered. At the junctions between 2 branches or more, potential continuity as well as current conservation are enforced. For extremity branches, sealed end boundary conditions are imposed. Fig. 13 presents the two dendritic networks studied in this paper, made of 3 and 7 branches respectively. Snapshots of the propagation of the action potential inside each tree at $t = 0.007$ and $t = 0.011$ s respectively are reported in Figs. 14 and 15. In both cases, the current is injected at the end of the top left-most branch as represented in Fig. 13 (branches A1 and B1).

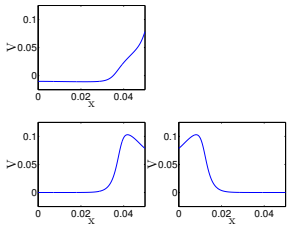


Fig. 14. Spatial snapshot of the action potential propagation at $t = 0.007$ s in a dendritic tree with 3 branches. From top to bottom and then left to right, potential in branches A1, A2 and A3.

In the case of these dendritic trees, the equation engine takes into account the network connectivity properties in order to set the equations preserving potential continuity and conservation of current at the junctions.

4) Model Reduction: The proposed model specification framework is useful for conveniently and efficiently building reduced-order models in networks. The main idea is to first build a reduced-order model (ROM) for a single branch of the network and then assemble as many such ROMs as there are branches in the network. An alternative is to build a single ROM for the entire network, as done in [26]. A main

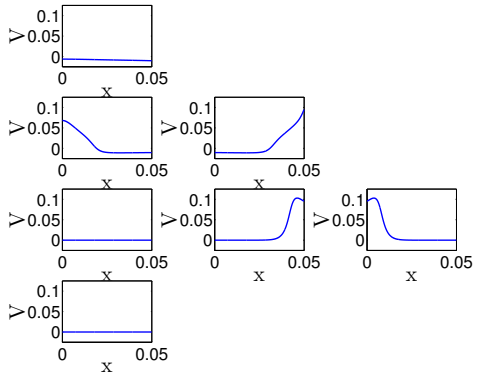


Fig. 15. Spatial snapshot of the action potential propagation at $t = 0.011$ s in a dendritic tree with 7 branches. From top to bottom and then left to right, potential in branches B1, B2, B3, B4, B5, B6 and B7.

advantage of the new procedure presented in this paper is the ability of building ROMs for networks of arbitrary topologies. Furthermore, conservation and continuity laws at junctions can be enforced in a similar fashion as in the original equation setting. Variations in the dendrites geometrical and physical parameters can be subsequently handled using the techniques recently developed in [27].

A reduced-order basis of dimension $n_{\mathrm{ROM}} = 30$ is first built for the potential satisfying the cable equations. This basis is built by Proper Orthogonal Decomposition (POD) using the method of snapshots [28]. As illustrated in Fig. 16, such a ROM captures very well the behavior of the underlying high-fidelity model, which is of size 1400.

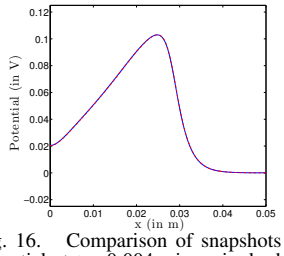


Fig. 16. Comparison of snapshots of potential at $t = 0.004$ s in a single dendrite obtained with the H-H model and the ROM. In blue: high-fidelity model, in red: reduced-order model.

After this training phase, arbitrary dendritic trees with n_{b} branches is considered. Global network ROMs of dimension $n_{\mathrm{ROM}} \times n_{\mathrm{b}} = 30 n_{\mathrm{b}}$ are built by assembling single branch ROMs together. The **ModSpec** framework is here a valuable support since the reduced-order basis can be added as an additional field of the model. Figs. 17 and 18 illustrate the very good agreements between the potential propagation predicted by the ROM and the high-fidelity model in both networks.

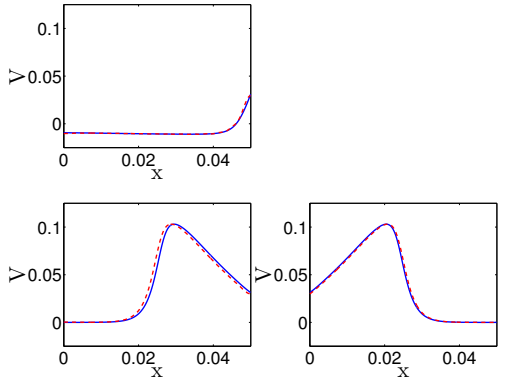


Fig. 17. Comparison of snapshots of potential at $t = 0.008$ s in a dendritic tree with 3 branches obtained with the H-H model and the ROM. In blue: high-fidelity model, in red: reduced-order model. From top to bottom and then left to right, potential in branches A1, A2 and A3.

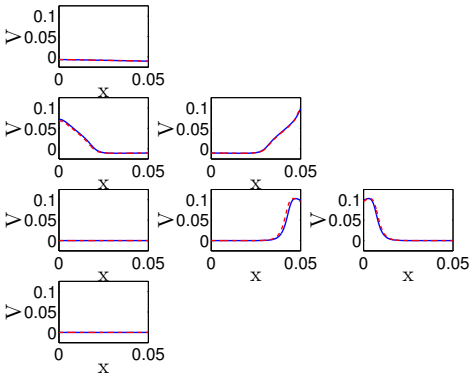


Fig. 18. Comparison of snapshots of potential at $t = 0.011$ s in a dendritic tree with 7 branches obtained with the H-H model and the ROM. In blue: high-fidelity model, in red: reduced-order model. From top to bottom and then left to right, potential in branches B1, B2, B3, B4, B5, B6 and B7.

The **ModSpec** framework can be also very useful in the context of hierarchical modeling where several types of neurons are connected to each other as in the case of the *Drosophila melanogaster* depicted in Fig. 19 [29].

V. Conclusions

We have developed a new framework for device model specification. This framework is general enough to be applicable to devices stemming from multiple domains as illustrated by electronic and neuronal devices in this paper. Furthermore, this new framework is flexible enough to allow its use within several simulation and modeling settings and with complex networks.

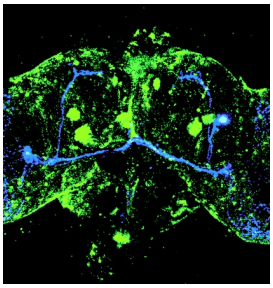


Fig. 19. Ventral lateral neurons in the *Drosophila melanogaster* [29].

Future work will focus on the use of compiled models within this framework, allowing the definition of dynamically loaded libraries with an associated API in C, as well as constructing translators for specific simulators including commercial ones. Another important characteristic to be added is hierarchical specification of devices. This is particularly important for enabling multilevel solution algorithms. Noise source specification will be enabled within **ModSpec**, with support for non-stationary noise sources. Finally, translation to and from VA-CME will be supported as well as all its features.

VI. Appendix 1: Model Specification for an Electronic Device

```
class ModSpec{
 %sizes
 DM.nExternalQuantities;     DM.nExternalUnknowns;
 DM.nExplicitOutputs;        DM.nInternalUnknowns;
 DM.nImplicitEquations;      DM.nExplicitOutputEquations;
```

```
 DM.nNoiseSources;
 % parameter support
 DM.nParms;                  DM.Parmnames;
 DM.Parmdefaults;            DM.Getparms;
 DM.Setparms;                DM.parms;
 %device properties
 DM.IsVoltageControlledDevice;
 % names
 DM.modelName;               DM.ExternalQuantityNames;
 DM.ExternalUnknownNames;    DM.InternalUnknownNames;
 DM.ExplicitOutputNames;
 % DAE functions
 DM.fi;                      DM.fe;
 DM.qi;                      DM.qe;
 DM.bi;                      DM.be;
 % Newton-Raphson limiting support
 DM.NRLimiting;
 % noise support
 DAE.nNoiseSources;          DAE.NoiseSourceNames;
 DAE.NoiseStationaryComponentPSDmatrix;
 DAE.NoiseModulationMatrix;
}
```

This is an example of class member (for a diode):

```
function out = NRLimiting(dx,x,DM)
  out = pnjlim_dx(dx, x, 0.025, 0.6145);
%end NRLimiting
```

VII. Appendix 2: vecvalder: a vectorial automatic differentiation class in **MATLAB**/Octave

Some members of the vecvalder class are presented below as well as utility subroutines. many other routines are omitted for lack of space but will be published openly.

```
function h = minus(u,v)
% minus is overloaded
  if ~isa(u,'vecvalder')
     h = vecvalder(u-val2mat(v), -der2mat(v));
  elseif ~isa(v,'vecvalder')
     h = vecvalder(val2mat(u)-v, der2mat(u));
   else
     h = vecvalder(val2mat(u)-val2mat(v),
       der2mat(u)-der2mat(v));
   end
end

function h = sqrt(u)
% sqrt is overloaded
  n = size(u(1).der,2);
  h = vecvalder(sqrt(val2mat(u)),
  der2mat(u)./repmat(2*sqrt(val2mat(u)),[1 n]));
end

function A = val2mat(u)
% Transforms the val fields into a matrix
  A = zeros(size(u));
  A(:) = cell2mat({u.val});
end

function A = der2mat(u)
% Transforms the der fields  into a matrix
  A = zeros(size(u(1).der,2),size(u,1));
  A(:) = cell2mat({u.der});
  A=A';
end
```

VIII. Appendix 3: API to the equation engine

```
class EquationEngine{
 %API functions
 DS.SearchDevice(deviceName);
 DS.AllDevicesModSpecs();
 DS.DisplayAllDevicesModSpecs();
 DS.GraphConnectivity();
 DS.IncidenceMatrix();
 DS.nDevices();
 DS.nImplicitEqInStruct(devStruct);
 DS.deviceParameters(iDevice);
}
```

Additional members in the derived class for circuits:

```
class EquationEngineCircuits{
 DS.NonVoltageContrDevices();
 DS.VoltageContrDevices();
}
```

Additional members in the derived class for neurons:

```
class EquationEngineNeurons{
 DS.DistalCompartmentsProperties();
 DS.ProximalCompartmentsProperties();
 DS.SomaLocation();
}
```

References

[1] W. Liu, X. Jin, J. Chen, M-C. Jeng, Z. Liu, Y. Cheng, K. Chen, M. Chan, K. Hui, J. Huang, R. Tu, P.K. Ko, and Chenming Hu. Bsim 3v3.2 mosfet model users' manual. Technical Report UCB/ERL M98/51, EECS Department, University of California, Berkeley, 1998.

[2] Xin Li, Colin C. McAndrew, Weimin Wu, Samir Chaudhry, James Victory, and Gennady Gildenblat. Statistical modeling with the psp mosfet model. *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems*, 29:599–606, 2010.

[3] T.L. Quarles. *Analysis of Performance and Convergence Issues for Circuit Simulation*. PhD thesis, April 1989. Memorandum no. UCB/ERL M89/42.

[4] ngSPICE circuit simulator. http://ngspice.sourceforge.net.

[5] Wikipedia. SPICE. http://en.wikipedia.org/wiki/SPICE.

[6] W.H. Press, S.A. Teukolsky, W.T. Vetterling, and B.P. Flannery. *Numerical Recipes – The Art of Scientific Computing*. Cambridge University Press, 1989.

[7] L.O. Chua and P-M. Lin. *Computer-aided analysis of electronic circuits : algorithms and computational techniques*. Prentice-Hall, Englewood Cliffs, N.J., 1975.

[8] Verilog-A Compact Model Extensions. http://www.vhdl.org/verilog-ams/htmlpages/public-docs/Verilog-A_Compact_Model_Extensions.pdf.

[9] Tiburon Design Automation. Tiburon DA products and documentation. http://www.tiburon-da.com/.

[10] L. LeMaitre, W. Grabinski, and C. McAndrew. Compact Device Modeling Using Verilog-aMS and ADMS. 2003.

[11] Jaijeet Roychowdhury. Numerical simulation and modelling of electronic and biochemical systems. *Foundations and Trends in Electronic Design Automation*, 3(2-3):97–303, December 2009.

[12] J. Eaton et. al. Octave. http://www.gnu.org/software/octave/index.html.

[13] R.D Neidinger. Introduction to automatic differentiation and matlab object-oriented programming. *SIAM review*, 52(3):545–563, 2010.

[14] DS Weile, E Michielssen, E Grimme, and K Gallivan. A method for generating rational interpolant reduced order models of two-parameter linear systems. *Applied Mathematics Letters*, 12(5):93–102, 1999.

[15] C Gu and J Roychowdhury. Model reduction via projection onto nonlinear manifolds, with applications to analog circuits and biochemical systems. *Proceedings of the 2008 IEEE/ACM International Conference on Computer-Aided Design*, pages 85–92, 2008.

[16] K Carlberg, C Bou-Mosleh, and C Farhat. Efficient nonlinear model reduction via a leastsquares petrov–galerkin projection and compressive tensor approximations. *International Journal for Numerical Methods in Engineering*, 86(2):155–181, 2011.

[17] S Chaturantabut and DC Sorensen. Nonlinear model reduction via discrete empirical interpolation. *SIAM Journal on Scientific Computing*, 32:2737, 2010.

[18] G Walker. Evaluating mppt converter topologies using a matlab pv model. *Journal of Electrical & Electronics Engineering, Australia*, 21(1):49–56, 2001.

[19] S.V. Dhople, A.Davoudi, A.D Dominguez-Garcia, and P.L Chapman. Unified approach to reliability assessment of multiphase dc-dc converters in photovoltaic energy-conversion systems. *IEEE Transactions on Power Electronics*, 2010.

[20] Randy J. LeVeque. *Finite volume methods for hyperbolic problems*. Cambridge University Press, 2002.

[21] M Rewienski and J White. Model order reduction for nonlinear dynamical systems based on trajectory piecewise-linear approximations. *Linear Algebra and its Applications*, 415(2-3):426–454, 2006.

[22] Neuron structure. http://creationwiki.org/pool/images/1/13/NEURON2.gif.

[23] E.M. Izhikevich. *Dynamical Systems in Neuroscience - The Geometry of Excitability and Bursting*. MIT Press, Cambridge, Massachusetts, 2007.

[24] M Hines. Efficient computation of branched nerve equations. *International journal of bio-medical computing*, 15(1):69, 1984.

[25] J.A Connor and C.F Stevens. Inward and delayed outward membrane currents in isolated neural somata under voltage clamp. *The Journal of Physiology*, 213:1–19, Jan 1971.

[26] AR Kellems, S Chaturantabut, DC Sorensen, and SJ Cox. Morphologically accurate reduced order modeling of spiking neurons. *Journal of computational neuroscience*, pages 1–18, 2010.

[27] D Amsallem and C Farhat. Interpolation method for adapting reduced-order models and application to aeroelasticity. *AIAA Journal-American Institute of Aeronautics and Astronautics*, 46(7):1803–1813, 2008.

[28] L Sirovich. Turbulence and the dynamics of coherent structures. part i: Coherent structures. *Quarterly of applied mathematics*, 45(3):561–571, 1987.

[29] Drosophila melanogaster neuron network. http://www.pnas.org/content/105/50.cover-expansion.