Towards Sensitivity Analysis of Hybrid Systems Using Simulink

Zhi Han
MathWorks
zhi.han@mathworks.com

Pieter J. Mosterman
MathWorks
pieter.mosterman@mathworks.com

ABSTRACT

In the design of engineered systems two types of models are used: (i) *analysis models* and (ii) *system models*. The system models are primary deliverables between design stages whereas analysis models are employed within a design stage. Sensitivity analysis studies the behavior of the system under small parameter variations which proves to be useful in design. To enable sensitivity analysis in verification of hybrid dynamic systems that model industry-size problems, support for simulation-based methods is desired. The computational semantics for simulation of corresponding analysis models must then be consistent with the computational semantics of the system models. A method is presented that enables direct sensitivity analysis on system models via an implementation in the Simulink® software. The approach relies on the existing ordinary differential equation solver of Simulink and the block-by-block analytic Jacobian computation to provide the analytic Jacobian for solving the sensitivity equations. Results of a prototype implementation show that sensitivity analysis can be applied to moderate size Simulink models of continuous and hybrid systems.

Categories and Subject Descriptors

I.6.3 [**Computing Methodologies**]: Simulation and ModelingApplications

Keywords

Sensitivity analysis; simulation; verification

1. INTRODUCTION

Design of complex technical systems such as automobiles, airplanes, satellites, etc., relies on the principles of (i) separation of concerns and (ii) divide and rule [15, 29]. Where the latter is the motivation to partition and modularize an overall system, the former presages the utility of various abstractions such as linearized models, architecture models, performance models, requirement models, trade-off models, etc. Generally, these models can be classified in (i) *system models*, which are included as deliverables in specifications between design stages and (ii) *analysis models*, which are utilized within design stages to deliver a specification based on incoming requirements.

With the advent of personal computing, the documents storing the various models increasingly became electronic by using office applications such as spreadsheet software, word processing software, graphics software, etc. The availability in an electronic modality then unlocked the potential to automate the generation of a representation amenable to computational simulation. While initially it eliminated the task performed by 'simulationists' to transform a model into computer code, such automation proceeded to enable sophisticated and domain specific formalism with semantics defined by their corresponding computational representation. By employing the same computational semantics for system models as they progress throughout the various design stages, in due time, the formal meaning of models became defined by the computational semantics [22].

As a consequence, for models that include continuous-time dynamics, such as *hybrid dynamic systems*, the approximations made by the resulting computational semantics (e.g., numerical integration solvers, algebraic equation solvers, root-finding solvers, etc.) have effectively come to define the meaning of the models. Though to an extent because of uninformed use of numerical methods, engineers working on the development of complex technical systems have come to accept and even rely on the computational approximations introduced by their modeling tool of choice. To these engineers, consistency of computational results between design stages supersedes correctness as a measure of computational closeness to the underlying theoretically analytic solutions that is typically not practically available. For tool developers, when developing features for computational models it is imperative to preserve the precise computational semantics even in the face of their approximative fidelity [23].

For verification of hybrid systems, the necessity to preserve computational semantics is critical in simulation-based approaches. Such approaches are developed to tackle scalability issues in reachability analysis for formal verification (*e.g.* [9, 11, 28]) as based on a combination of methods from discrete systems, such as model checking and algorithmic abstractions, and computational methods for continuous systems, such as computational geometry and numerical simulations. Here, for industry problems, the state space that must be analyzed often becomes prohibitively large and solu-

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
HSCC'13, April 8–11, 2013, Philadelphia, Pennsylvania, USA.
Copyright 2013 ACM 978-1-4503-1567-8/13/04 ...$15.00.

tions resort to simulation-based reachability analysis, which uses sensitivity analysis of the simulated trajectories as an intermediate result (*e.g.* [1, 4, 6, 7, 13]). Thus solving industry problems renders it important to support sensitivity analyses and so computing sensitivities as part of the *analysis models* must preserve the computational semantics and not affect the simulation results of the *system models*.

Sensitivity analysis (SA) is the study of how variation in the output of a model can be apportioned to different sources of variation, and of how the given model depends upon the information input into it. SA has been used in engineering prior to the use in reach set computation, for instance, to find the most influential parameters in a model for optimization and parameter tuning [8]. In the hybrid systems domain, SA methods have been used to analyze hybrid dynamic systems and applied in computing critical parameter values of electric power networks [14, 24]. Although success cases have been reported in the literature, there is still a lack of tool support to enable SA in the software tools for the design of engineered system. For example, as a popular industry tool, Simulink® [20] provides a modeling environment for control system design engineers to build complex, hybrid systems using hierarchical block diagrams. Although SA functionality has been implemented in ordinary differential equation (ODE) solvers (*e.g.*, [26]), this functionality cannot be integrated in Simulink by simply replacing the Simulink ODE solver because the computational semantics of Simulink models include various additional numerical algorithms to support not only continuous-time behavior (*e.g.*, an algebraic equation solver) but, perhaps more importantly, also discrete and hybrid behavior (*e.g.*, root-finding).

Ideally one would like to have the capability to perform SA in analysis models without affecting the numerical computations performed in simulation of the corresponding system models. If implemented in a separate tool, this would require capturing the complete computational semantics of industry-strength simulation software such as Simulink, which presents a fundamental language engineering challenge that is yet to be solved [21]. Instead, the approach taken in this paper *embeds* SA directly into the existing simulation engine and relies on the existing ODE solver combined with the block-by-block analytic Jacobian computation of Simulink to solve the sensitivity equations. By implementing the majority of the SA capability in a block based on the standard Simulink block interface (S-function) API, a model can be easily configured for SA by including the block in the model diagram without diagrammatic complications. The prototype SA for Simulink hybrid system models is based on the research results of [8] and [14] and applied to a small set of test models. Results show that without affecting the computational semantics, SA can be performed on moderate size Simulink models with hundreds of blocks and over 60 continuous states, illustrating the scalability of the approach.

2. SENSITIVITY EQUATIONS

Most of the discussion in this section can be found in [8] and [14] with a slightly different formulation.

Consider an ODE system given in the following form $\dot{x} = f(x, p, t)$, $x(0) = x_0$ where $x \in \mathbb{R}^n$ is the state vector, $p \in \mathbb{R}^m$ is a vector of constant parameters, and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a function that is continuously differentiable with respect to p, x, and t. Also assume that the system has a unique solution for any given initial value x_0 and parameter value p. Denote the unique solution by $\xi(t)$ and call it the *trajectory* of the system. The trajectory sensitivity of the system is

$$s_p^x \equiv \left.\frac{\partial \xi}{\partial p}\right|_{p,x_0}(t). \tag{1}$$

where $s_p^x \in \mathbb{R}^{n\times m}$ is the trajectory sensitivity matrix. The trajectory sensitivity can be used as a linear approximation for a *perturbed* trajectory $\xi|_{p+\Delta p,x_0}(t) \approx \xi|_{p,x_0}(t) + s_p^x|_{p,x_0}(t)\Delta p$, where $\Delta p \in \mathbb{R}^m$ and $\|\Delta p\| \ll \|p\|$.

The dynamics of s satisfies the *sensitivity equations* [8]:

$$\dot{s}_p^x = \frac{\partial f}{\partial x} s_p^x + \frac{\partial f}{\partial p}, \text{ and } s_p^x(0) = \frac{\partial x_0}{\partial p} \tag{2}$$

In formal verification of hybrid systems, a useful intermediate step is to compute the sensitivity matrix with respect to the initial state, where the sensitivity equation is: ([6])

$$\dot{s}_{x_0}^x = \frac{\partial f}{\partial x} s_{x_0}^x,\ s_{x_0}^x(0) = I_n. \tag{3}$$

Solving sensitivity equations for continuous systems involves solving the ODEs in Eq. (2) or Eq. (3). The sensitivity equations are time-varying linear ODE systems, which can be solved by general-purpose nonlinear ODE solvers or exponential ODE solvers [25]. Note that since $\frac{\partial f}{\partial x}$ must be evaluated, it is necessary to solve the system equations in Eq. (1) simultaneously, which results in a problem that consists of $n \times m + n$ continuous states.

For hybrid dynamic systems, special attention must be paid to the discrete events that are triggered by the continuous dynamics. Since the time of a discrete event t_e is dependent on the parameter values, the sensitivity of t_e must be calculated with respect to p [14]. To find t_e, Simulink employs a root-finding mechanism called zero-crossing detection [30]. Now define a continuously differentiable function $g : \mathbb{R}^n \to \mathbb{R}$ and call it a *zero-crossing* function. Let t_e denote the time of the *zero-crossing event*, that is, the instance of time where the function g changes its sign. Under the same assumptions provided in [14], the sensitivity of the zero-crossing event time can be derived as follows. Since the zero-crossing event is described by $y = g(\xi(t_e)) = 0$ it follows that

$$\frac{\partial y}{\partial t_e}\Delta t + \frac{\partial y}{\partial p}\Delta p = 0$$

With some manipulation, the sensitivity of the time of zero-crossing event can be derived as

$$s_p^{t_e} \equiv \frac{\partial t_e}{\partial p} = -\frac{1}{\frac{\partial g}{\partial x}\cdot f(x,p,t)}\frac{\partial g}{\partial x} s_p^x(t_e) \tag{4}$$

Assuming that there is no state reset after the zero-crossing event, and let f^+ denote the right-hand side of the system equations after the zero-crossing event, the trajectory sensitivity matrix is reset at the zero-crossing event according to the following *jump condition* [14]:

$$s_p^x(t_e^+) = s_p^x(t_e^-) + \left(f^-(x,p,t_e) - f^+(x,p,t_e)\right) s_p^{t_e}. \tag{5}$$

3. IMPLEMENTATION

Numerical solvers for ODEs are necessary to solve Eq. (2) and Eq. (3) along with means to evaluate the partial derivative terms (Jacobians) in those equations. The evaluation of Jacobians can be performed either analytically or by numerical approximations such as finite differences. Analytic

computation of the Jacobian is usually preferred as it can be more accurate and efficient using algorithmic differentiation (AD) tools [12]. In Simulink, block-by-block analytic Jacobian evaluations have been used in tools such as Simulink® Control Design™ [19] to evaluate the model Jacobian.

3.1 Simulation loop in Simulink®

The simulation of a continuous dynamic system model in Simulink consists of a model compilation phase, a link phase, and a simulation loop phase [20]. In the compilation phase, Simulink models are compiled into an operational form. In the linking phase, the resources necessary for simulation are allocated. The simulation loop phase depicted in Fig. 1 is the step where the dynamic equations of the model are solved by computing state and output signal values at each consecutive time step. At each time step, Simulink first invokes the `Output` method to compute the output signals of each block, then Simulink invokes the *ODE solver* to compute the next time step and the state values at the next time step. For zero-crossing functions Simulink then checks if the signs of the zero-crossing functions have changed for the new state values. If so, Simulink invokes the *zero-crossing detector* to further reduce the time step to locate the time instances of the zero-crossing events. Once an appropriate time step has been found and the corresponding state vector values are computed, Simulink advances to the next time step. This loop is repeated till the simulation is completed.

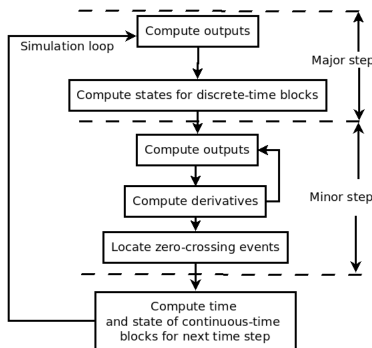


Figure 1: Simulation loop of Simulink®

The ODE solver and zero-crossing detector of Simulink invoke the `Output` and `Derivative` methods of the model to compute signal values and the state derivative vector. In variable step solvers, the `Output` method and `Derivative` method may be invoked multiple times in one step. And some results from these method calls may be discarded based on estimations of the approximation error. To distinguish the `Output` calls from the simulation loop and the calls originating from the solver and the zero-crossing detector, the former is referred to as the *major* step and the latter is called the *minor* step [20].

After the `Output` method completes, Simulink may optionally compute the analytic model Jacobian before invoking the ODE solvers for the sensitivity analysis. The computation of the model Jacobian consists of three steps. First, the Jacobian of each block is computed by invoking their `Jacobian` method and a set of matrices is computed to represent the linear relationship corresponding to block connections. Second, the analytic Jacobians of all blocks are concatenated into a set of Jacobian matrices along with the matrices for block connections combined into the *open-loop model Jacobian* data structure. Third, the linearization algorithm performs *derivative accumulation*: it uses the open-loop model Jacobian to compute the closed-loop model Jacobian $\frac{\partial f}{\partial x}$ [12]. The derivative accumulation of Simulink is implemented as linear fractional transformations (LFT) [31].

3.2 The sensitivity blocks

The SA for continuous-time Simulink blocks is implemented in a number of S-function blocks. To perform SA with respect to initial conditions, the `Sensitivity` block must be included in the root level of a Simulink model. If a block contains parameters for which the sensitivity is computed, changes must also be made to the block. The solution of the sensitivity vector is computed in the callback function of the block. At the major `Output` steps, the `Sensitivity` block notifies Simulink that the block-by-block analytic Jacobian of the model must be computed. After completing the `Output` of all blocks, Simulink computes the open-loop model Jacobian and invokes the block callback function with the open-loop model Jacobian as an argument.

Figure 2 shows the control flow of the simulation loop with the computation of the sensitivity. Once control has reached the block callback function, it uses the open-loop model Jacobian to compute the closed-loop model Jacobian. Then the state value of the model is used to compute the sensitivity matrix. Note that since the block callback function is invoked at the end of the current step (*i.e.*, the solver has not yet computed the next time step), the sensitivity is computed for the current time step. To handle sensitivity of the zero-crossing event time, another block that is implemented as an S-function and only invoked when a zero-crossing event is detected computes the jump conditions.

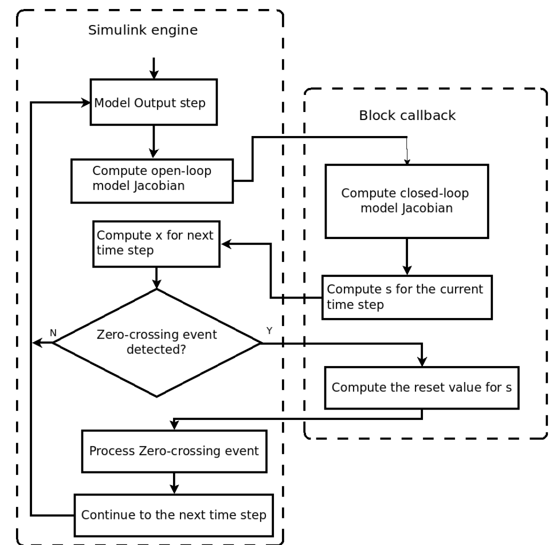


Figure 2: Block callback for SA

To compute the sensitivity with respect to a block parameter the *linear analysis point* specification of Simulink [19] is utilized. When a signal y is marked as a linear analysis point, the Simulink engine treats the output port of the signal as a special port and marks the location of the signal in the open-loop model Jacobian.

For example, suppose the block implements $y = \phi(p, i)$ where y is the output signal of the block, p is a constant parameter, and i is the input signal of the block. Now suppose

a state-less block provides the input to a block with a derivative function to form the system equation $\dot{x} = f(x, \phi(p, i), t)$. With u the input to the connected blocks the following relation holds $\frac{\partial f}{\partial p} = \frac{\partial f}{\partial y}\frac{\partial \phi(i,p)}{\partial p}$. To obtain $\frac{\partial f}{\partial p}$, the signal y is marked as a linear analysis point, which enables Simulink to compute the partial derivative $\frac{\partial f}{\partial y}$ from the open-loop model Jacobian using LFT. Also $\frac{\partial \phi}{\partial u}$ can be computed from the function ϕ using u and p. For the `Gain` blocks used in this section, and for which $\phi(u) = Ku$, the partial derivative is $\frac{\phi}{K} = u$. For blocks with complicated algorithms, the source code of the block must be modified to implement the partial derivative.

Figure 3 illustrates this by the Van der Pol example augmented with the blocks to perform SA. The changes to the model are highlighted in green. It has a masked `Gain` block that specifies the gain value as the parameter sensitivity input. It also has a `Sensitivity` block that implements the block callback function for SA. The masked Gain block is a subsystem that has two blocks as its children: one block computes the output and the other block computes $\frac{\partial \phi}{\partial p}$ and passes the value to the `Sensitivity` block.

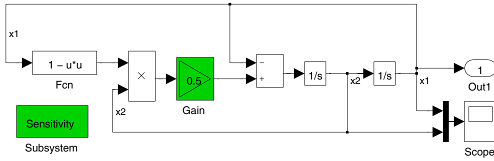
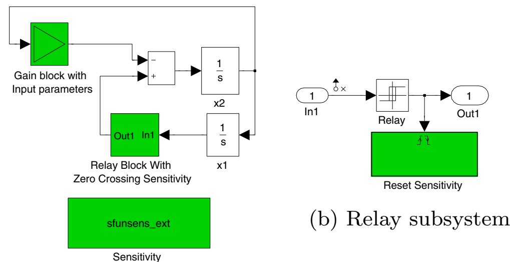


Figure 3: A Simulink® model with SA

Sensitivity for systems with zero-crossing events is implemented in a few Simulink blocks. The `Reset Sensitivity` block and `Sensitivity` block share data by reading and writing to a shared workspace. Figure 4 shows the block diagram used for the computation of sensitivity for a model with zero-crossing events. Figure 4b shows the content of the masked subsystem `Relay Block With Zero-crossing Sensitivity` in Fig. 4a. The subsystem consists of a `Relay` block and a triggered subsystem block to compute the jump condition at the zero crossing event. The computation of the zero-crossing event requires computing $\frac{\partial g}{\partial x}$ in Eq. (4), which can be computed from the open-loop model Jacobian once the signal corresponding to $g(x)$ is marked for analytic Jacobian as linear analysis points. The upward arrow icon on the output port of the `Inport` block in Fig. 4b shows that indeed this signal is marked as linear analysis point. The `Sensitivity` block performs the computation to solve the sensitivity equations.

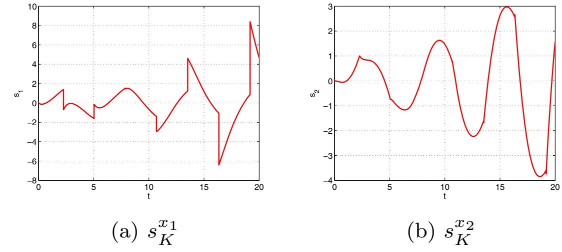


(a) System model

(b) Relay subsystem

Figure 4: SA for models with zero-crossing events

Figure 5a shows the SA results for the model with zero-crossing events in Fig. 4 where the sensitivity of the state trajectory is computed with respect to the parameter value of the `Gain` block. Note that the state sensitivity has discontinuities for both states because of the jump conditions at the zero-crossing events.

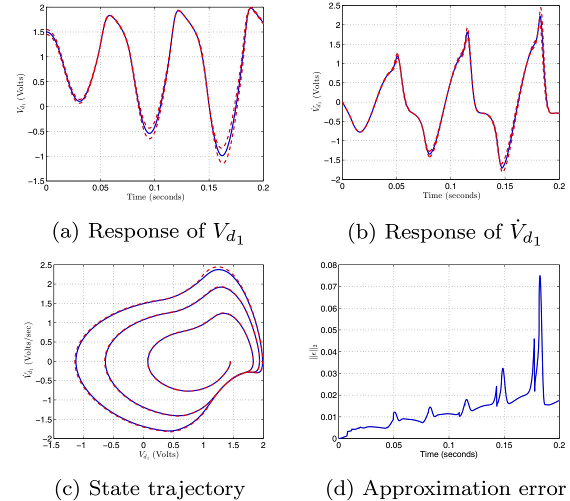


(a) $s_K^{x_1}$ (b) $s_K^{x_2}$

Figure 5: SA for systems with zero-crossing events

4. COMPUTATION RESULTS

The prototype implementation is tested on a number of Simulink models of continuous-time dynamic systems. The first is a Simulink model of a voltage controlled oscillator (VCO) taken from [6, 10]. The initial values of the simulation are chosen to be $V_{d_1} = 1.5$ Volts, $V_{d_2} = 0.5$ Volts and $\dot{V}_{d_1} = \dot{V}_{d_2} = 0$ Volts/s. Using a variable-step solver `ODE45`, the simulation of a Simulink model of the VCO system takes 253 steps to simulate from 0 to 0.2 seconds. The time steps and state trajectories with SA are identical to the results without SA which confirms that the computational semantics are preserved. Figure 6 shows the simulation result of the VCO model together with the approximation of the perturbed responses. The simulation results are shown in solid lines and the approximation of the perturbed trajectories for $\pm\Delta V_{d_1}$ using SA are shown as the two dashed curves.



(a) Response of V_{d_1} (b) Response of $\dot{V}_{d_1}$

(c) State trajectory (d) Approximation error

Figure 6: Simulation results of VCO model

To evaluate the approximation errors, the linear approximation using SA is compared with a simulation of the perturbed state trajectory. Figure 6c shows the approximation of a perturbed simulation for $V_{d_1} = 1.45$ Volts overlaid with the actual simulation results of the perturbed initial state. The approximation using SA is shown as dashed lines and the actual simulation result is shown in solid lines.

The approximation error for state trajectory using SA is

$$\epsilon(t) = \xi|_{p+\Delta p, x_0}(t) - \xi|_{p,x_0}(t) - s_p^x|_{p,x_0}(t)\Delta p.$$

Figure 6d shows the two-norm of the approximation error for the perturbed state trajectory. The approximation error

is small compared to the state values. The approximation error accumulates as the simulation progresses, leading to noticeable approximation errors in the state trajectory.

The prototype implementation is further used to compute trajectory sensitivity for a number of Simulink models. All SA are performed for sensitivity to initial conditions. The experiments are all performed on a computer with Intel® Xeon® CPU X5650 at 2.67GHz and 24 GiB memory running Debian Linux 6. Table 1 shows the computational results for a number of Simulink example models and Table 2 shows the computational results for a number of mechanical system models using SimMechanics™ [18]. The models are listed in the order of number of blocks in the block diagram. The results show that SA can be applied to Simulink models of continuous dynamic systems with moderate complexity. For example, for mechanical system Model 6, where the system consists of 268 blocks and 61 continuous states, the total computation time for SA is slightly more than twice that of the computation time for a simulation, even though the combined sensitivity equation and system equation comprises an ODE with 3782 continuous state variables.

Test Model	# of blocks	# of states	Simulation time (seconds)	Time for SA (seconds)
`sldemo_ dblcart1`	20	8	0.50	19.73
`sldemo_ f14`	75	10	0.54	18.01
`sldemo_ engine`	85	4	1.67	20.53
`sldemo_ hydrod`	128	6	1.75	4.52
`sldemo_ hydcyl4`	136	4	2.94	6.30

Table 1: Results for Simulink® example models

Test Model	# of blocks	# of states	Simulation time (seconds)	Time for SA (seconds)
Scotch yoke	47	8	1.15	6.56
Double pendulum	49	4	2.54	5.42
Flexible four bar mechanism	56	8	1.38	6.55
Cross slider	66	8	10.48	118.75
Crank slider	79	8	1.93	2.24
Offset slider	117	16	14.75	216.75
Stewart platform	268	61	5.49	11.8
Walking gait mechanism	414	32	12.83	86.05

Table 2: Results for SimMechanics™ models

There are a number of cases where the computational overhead of SA is significant. For example, the total time of SA for the double-cart example model is around 40 times the simulation time. Profiling the computation reveals that most of the computation time is spent in derivative accumulation and numerical integration of the sensitivity equations.

5. DISCUSSION

Because of their popularity in embedded systems design, Simulink models have attracted attention from researchers in the hybrid dynamic system domain. There have been numerous efforts to apply research results in hybrid dynamic systems to Simulink products, particularly in the area of formal verification (*e.g.*, [2, 4, 16, 27]). Most methods limit the scope to a fairly restricted subset of Simulink blocks. Some methods rely heavily on translation mechanisms that translate Simulink models into equivalent models that are amenable to analysis with a developed method, so the analysis can be performed externally on the translated model.

An existing software package that provides SA directly for Simulink is `DiffEdge` [3]. The software provides SA capability by augmenting the system model, that is, by creating new block diagrams on top of the existing block diagram model. The newly created block diagram models include the sensitivity equations, which can be simulated to provide SA results. The approach has a number of shortcomings, though. First, the newly created block diagram model has significantly increased complexity as it consists of at least twice the number of blocks and twice the number of signal connections compared to the original block diagram model. Second, the method creates a new model from an original model for sensitivity analysis and it is unclear how close the trajectories of the new model and those of the original model are. Third, the method relies on analyzing the structure of a given model and creating blocks, subsystems, and adding connections to create the new model. During model compilation the Simulink software may transform the model by removing blocks, inserting blocks, and rerouting signals [20] and it is unclear whether the method is compatible with the transformations performed by Simulink.

The approach taken in this paper is direct: it does not require intrusive changes to the original block diagram for simulation. It uses the existing computation procedures in Simulink. Although the implementation is still a prototype, it demonstrates that the functionality required by SA is available in the Simulink software and the method holds promise to handle industry models. The computational results show that SA for hybrid systems can be a feasible and useful enhancement to the Simulink software.

The software tool `Breach` performs SA and parameter synthesis [5]. It uses Real-Time Workshop® [17] to generate C code for Simulink models and uses CVODES for solving ODEs and computing trajectory sensitivity [26]. The approach in this paper is different because it applies to the Simulink execution engine and solver directly, which does not require the additional step of code generation.

The prototype implementation also reveals some limitations of the approach. Preliminary results showed that the computational overhead involved in the SA can be large. One of the computational steps that require a significant amount of time is the computation of the closed-loop Jacobians. Because for SA the model Jacobian must be evaluated many times during a simulation, it is crucial to improve the efficiency to compute Jacobians and some work in progress in this direction has been made.

Another limitation of the method is that it does not consider discrete-time systems. A straightforward manner to support them is to consider discrete-time dynamic systems described as difference equations with fixed sampling rate and extend the sensitivity equations to difference systems.

$$y_k = g(x_k, u_k) \tag{6}$$
$$x_{k+1} = f(x_k, u_k) \tag{7}$$

This method is adopted by Simulink Control Design and used in its linearization functionality [19]. When multiple sampling rates are encountered, a system with one sampling rate is converted to an equivalent system with another sampling rate. Since the sampling rate has changed, the converted system is not semantically identical to the original. The method is not used in the implementation of SA because, first, it does not support Simulink capabilities such

as sample time offsets and, second, it does not consider how rate transitions are handled in Simulink.

Another difficulty with difference equations is the compatibility with implementations of discrete-time blocks. A Simulink block must have Eq. (6) and Eq. (7) implemented separately for the method to apply. When modeling discrete-time dynamic systems, the block author is allowed to implement an `Output` method with the combined effects of Eq. (6) and Eq. (7). To compute the Jacobian of Eq. (6) and Eq. (7) the block author must first analyze the implementation of the dynamics and extract the equations for Eq. (6) and Eq. (7) and re-implement these methods for the block.

Finally, in addition to discrete-time dynamic systems, the method developed in this paper does not support state reset functions because of their implementation in Simulink. In Simulink, state reset is modeled by configuring a state port, initial condition port, and reset port for blocks with continuous states. The implementation makes it difficult to isolate and extract the parts that contribute to the reset function.

6. CONCLUSION

This paper presents a prototype implementation of sensitivity analysis in Simulink that preserves the computational semantics between analysis models and systems models. The implementation employs existing Simulink algorithms for solving ODEs and sensitivity equations. Computational results using the implementation showed that it can be applied to Simulink models with hundreds of blocks and less than 100 continuous states with reasonable computational overhead. Limitations of the current implementation have been discussed along with reference to recent progress on improving the efficiency.

Acknowledgments

We thank Dr. Fu Zhang at MathWorks® for the numerous discussions and contributions to the ideas that are published in this paper. We thank professor Dr. Goran Frehse for the original implementation of the VCO model.

References

[1] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *IEEE Conference on Decision and Control (CDC)*, pages 4042–4048, 2008.

[2] R. Alur, A. Kanade, K. C. Shashidhar, and S. Ramesh. Generating and analyzing symbolic traces of Simuink/Stateflow models. In *International Conference on Computer-Aided Verification*, pages 430–445, 2009.

[3] AppEdge Consulting and Engineering. DiffEdge: Differentiation, sensitivity analysis and identification of hybrid models under simulink. http://pagesperso-orange.fr/appedge/diffedge_intro.htm.

[4] T. Dang, A. Donzé, O. Maler, and N. Shalev. Sensitive state-space exploration. In *Proceedings of the IEEE Conference on Decision and Control*, pages 4049 –4054, 2008.

[5] A. Donzé, B. Krogh, and A. Rajhams. Parameter synthesis for hybrid systems with an application to simulink models. In *HSCC*, pages 165 – 179, 2009.

[6] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In *Hybrid Systems: Computation and Control (HSCC 2007)*, pages 174–189, 2007.

[7] A. Donzé and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal modeling and analysis of timed systems*, pages 92 – 106, 2010.

[8] P. M. Frank. *Introduction to System Sensitivity Theory*. Academic Press Inc., 1978.

[9] G. Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Hybrid Systems: Computation and Control (HSCC'05)*, pages 258–273, 2005.

[10] G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *Design, Automation and Test in Europe (DATE)*, pages 257 – 262, 2006.

[11] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control (HSCC'05)*, pages 291–305, March 2005.

[12] A. Griewank and A. Walther. *Evaluating Derivatives: Principle and Techniques of Algorithmic Differentiation*. SIAM, 2008.

[13] Z. Han and B. H. Krogh. Reachability analysis of nonlinear systems using trajectory piecewise linearized models. In *American Control Conference*, pages 1505–1510, 2006.

[14] I. A. Hiskens and M. A. Pai. Trajectory sensitivity analysis of hybrid systems. *IEEE Transaction on Circuits and Systems*, 47(2):204 – 220, 2000.

[15] M. Jackson. Some complexities in computer-based systems and their implications for system development. In *Proceedings of the 1990 IEEE International Conference on Computer Systems and Software Engineering*, pages 344–351, Tel-Aviv, Israel, May 1990.

[16] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from Simuink/Stateflow models. In *Hybrid systems computation and control (HSCC)*, pages 317–318, 2011.

[17] MathWorks. *Real-Time Workshop*. MathWorks, Inc., Natick, MA, Mar. 2009.

[18] MathWorks. *SimMechanics*. MathWorks, Inc., Natick, MA, Sept. 2012.

[19] MathWorks. *Simulink Control Design*. MathWorks, Inc., Natick, MA, Sept. 2012.

[20] MathWorks. *Simulink User's Guide*. The MathWorks, Inc., Natick, MA, Sept. 2012.

[21] P. J. Mosterman and H. Vangheluwe. Computer automated multi-paradigm modeling: An introduction. *SIMULATION: Transactions of The Society for Modeling and Simulation International*, 80(9):433–450, Sept. 2004.

[22] P. J. Mosterman and J. Zander. Advancing model-based design by modeling approximations of computational semantics. In *Proceedings of the 4th International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools*, pages 3–7, Zürich, Switzerland, Sept. 2011.

[23] P. J. Mosterman, J. Zander, G. Hamon, and B. Denckla. A computational model of time for stiff hybrid systems applied to control synthesis. *Control Engineering Practice*, 20(1):2–13, 2012.

[24] T. B. Nguyen, M. A. Pai, and I. A. Hiskens. Computation of critical values of parameters in power systems using trajectory sensitivities. In *Power System Computation Conference (PSCC)*, pages 24 – 28, 2002.

[25] D. A. Pope. An exponential method of numerical integration of ordinary differential equations. *Communications of the ACM*, 6(8):491–493, 1963.

[26] R. Serban and A. C. Hindmarsh. CVODES: An ODE solver with sensitivity analysis capabilities. Technical Report UCRL-JP-200039, LLNL, 2003.

[27] B. I. Silva, K. Richeson, B. H. Krogh, and A. Chutinan. Modeling and verifying hybrid dynamic systems using CheckMate. In *ADPM 2000 Conference Proceedings: The 4th International Conference on Automation of Mixed Processes - Hybrid Dynamic Systems*, 2000.

[28] P. Varaiya. Reach set computation using optimal control. Available online at http://paleale.eecs.berkeley.edu/ ∼varaiya/ papers_ps.dir/reachset.ps, 1998.

[29] K. Wijbrans. *Twente Hierarchical Embedded Systems Implementation by Simulation: a structured method for controller realization*. PhD dissertation, University of Twente, Enschede, The Netherlands, 1993. ISBN 90-9005933-4.

[30] F. Zhang, M. Yeddanapudi, and P. J. Mosterman. Zero-crossing location and detection algorithm for hybrid system simulation. In *IFAC World Congress*, pages 7967 – 7972, 2008.

[31] K. Zhou and J. C. Doyle. *Essentials of Robust Control*. Prentice Hall, 1998.