SAND2017-8875C

DAGSENS: Directed Acyclic Graph Based Direct and Adjoint Transient Sensitivity Analysis for Event-Driven Objective Functions

Karthik V. Aadithya[‡], Eric Keiter, and Ting Mei
Sandia National Laboratories, Albuquerque, NM, USA
[‡]Corresponding author. Email: kvaadit@sandia.gov

***Abstract*—We present DAGSENS, a new approach to parametric transient sensitivity analysis of Differential Algebraic Equation systems (DAEs), such as SPICE-level circuits. The key ideas behind DAGSENS are, (1) to represent the entire sequence of computations from DAE parameters to the objective function (whose sensitivity is needed) as a Directed Acyclic Graph (DAG) called the "sensitivity DAG", and (2) to compute the required sensitivites efficiently by using dynamic programming techniques to traverse the DAG. DAGSENS is simple, elegant, and easy-to-understand compared to previous approaches; for example, in DAGSENS, one can switch between direct and adjoint sensitivities simply by reversing the direction of DAG traversal. Also, DAGSENS is more powerful than previous approaches because it works for a more general class of objective functions, including those based on "events" that occur during a transient simulation (*e.g.*, a node voltage crossing a threshold, a phase-locked loop (PLL) achieving lock, a circuit signal reaching its maximum/minimum value, *etc.*). In this paper, we demonstrate DAGSENS on several electronic and biological applications, including high-speed communication, statistical cell library characterization, and gene expression.**

I. Introduction

This paper is about a new, elegant, and powerful approach to computing transient sensitivities of dynamical systems. In integrated circuit design, such sensitivities are used for a variety of applications including optimization and tuning [1], yield estimation [2], performance modelling [3], statistical cell library characterization [4], *etc.* Indeed, as CMOS technology moves to progressively smaller feature sizes (7 nm and below), and with the advent of near-threshold and sub-threshold computing, such sensitivities are likely to become even more important in variability-aware circuit design, due to the increasingly significant role played by parameter variability in determining circuit performance (speed, power consumption, bandwidth, *etc.*), as well as yield [2–4].

Parametric sensitivities are also important for analyzing and designing biological systems; applications include identifying influential kinetic parameters and model order reduction of chemical reaction networks [5], improving model accuracy and separating biological mechanisms from mathematical artifacts [6, 7], calculating the relative importance of different parallel processes in biological phenomena such as gene expression [8], understanding the factors at work behind the robustness of biological systems to parameter variability [9, 10], *etc.*

Broadly speaking, there are two approaches to sensitivity analysis: "direct" and "adjoint". Direct methods [11] involve computing the sensitivities of *all intermediate quantities* with respect to *system parameters* (by repeatedly applying the chain rule of differentiation), until the sensitivity of the desired "objective function" is eventually found. Direct methods are easy to understand and implement, but scale poorly with the number of parameters. Adjoint methods [4, 12–16], on the other hand, work backwards; they compute the sensitivities of *the objective function* with respect to *all intermediate quantities*, until the sensitivity of the objective function with respect to the parameters is eventually found. Adjoint methods tend to be harder to understand and implement, but they scale well to problems with large numbers of parameters. In particular, if one is only interested in calculating the sensitivities of a small number of performance metrics with respect to a large number of parameters, adjoint methods allow one to take advantage of the small dimensionality of the performance space to significantly speed up sensitivity analysis. Modern circuit simulation problems often fit this description; in these cases, computing sensitivities is usually practical only using adjoint methods.

Despite their long history and continued importance, we believe that adjoint methods are not well understood. This is partly because existing descriptions in the literature use hard-to-understand concepts and esoteric constructs such as adjoint circuits [1, 17], integrating DAEs backwards in time [12, 13, 16], complicated mathematics involving feeding δ-function inputs to DAEs [12], *etc.*

Also, previous works on sensitivity analysis have lacked generality with respect to the objective functions supported; they have restricted themselves to the sensitivities of DAE solution variables at specific points in time (*i.e.*, "final point" objective functions, §II-E) [4, 14], or integrals of the solution over time (*i.e.*, "integral" objective functions, §II-E) [13, 16], or simple combinations thereof, such as the ratio of two integrals [12]. In practice, such simple objective functions are often inadequate. Circuit designers frequently need more advanced, "event-driven" metrics defined based on "events" that take place during a transient run. Examples include a signal crossing a user-specified threshold, or possibly reaching a maximum/minimum value. A more complicated objective function could be the amount of time taken by a phase-locked loop (PLL) to lock to a new input frequency. The "achievement of lock" by a PLL is a transient event, and so the "time to lock" is an event-driven objective function, whose sensitivities with respect to PLL parameters are of interest to designers. Indeed, many popular circuit simulators already provide ways to calculate such event-driven objectives (*e.g.*, the .MEASURE feature found in HSPICE® [18] and Xyce® [19]); we believe we are the first to address the problem of calculating the *sensitivities* of such objectives.

In this paper, we develop DAGSENS, a new theory for transient sensitivity analysis based on Directed Acyclic Graphs (DAGs). In this approach, we construct a DAG where each intermediate quantity computed during a transient run is represented as a node, starting from DAE parameters all the way up to the objective function (§II-D, §II-E). Once this DAG is constructed, all required sensitivities can be obtained by traversing it, using efficient techniques based on dynamic programming (§II-G) [20, 21]. A key advantage is the simplicity and elegance of the DAG approach, which we believe previous approaches lack; for example, in DAGSENS, to switch from direct to adjoint sensitivity analysis, one simply changes the direction of DAG traversal from topological order to reverse topological order (§II-G), much like forward and reverse mode automatic differentiation [22, 23]. Thus, DAGSENS eliminates the need for the hard-to-understand, esoteric mathematical constructs mentioned above, and can thus make the benefits of sensitivity analysis accessible to a wider audience, including device engineers, circuit designers, and students. Also, DAGSENS is more powerful than existing sensitivity analysis approaches because it can handle more general objective functions, including the event-driven objectives described above.

The rest of this paper is organized as follows. In §II, we provide some technical background (on DAEs, transient analysis, *etc.*), and then discuss the core techniques underlying DAGSENS. In §III, we apply DAGSENS to compute event-driven sensitivities in several electronic and biological applications, including high-speed communication

(§III-A), statistical cell library characterization (§III-B), and gene expression in *Drosophila* embryos (§III-C). We conclude in §IV.

II. Core Techniques and Algorithms for DAG-based Event-driven Sensitivity Analysis

A. DAE models of dynamical systems

Throughout this paper, we assume that the system we wish to analyze is a DAE of the form:

$$\frac{d}{dt}\vec{q}(\vec{x}(t), \vec{p}) + \vec{f}(\vec{x}(t), \vec{u}(t), \vec{p}) = 0, \tag{1}$$

where $\vec{x}$ is the system's state vector (*e.g.*, a list of voltages and currents), $\vec{p}$ is a vector of parameters with respect to which we need sensitivities (transistor dimensions, parasitic resistances/capacitances, *etc.*), $\vec{u}$ is a vector of inputs to the system, and t denotes time. We note that Eq. (1) is capable of modelling virtually any electronic circuit at the SPICE level, and many biological systems as well [24–26].

B. Transient analysis of DAEs

Given an initial condition $\vec{x}(t_0) = \vec{x}_0$, and time-varying inputs $\vec{u}(t)$ to the DAE of Eq. (1), *transient analysis* refers to the problem of solving for the time-varying DAE state $\vec{x}(t)$ over a time-interval $[t_0,\ t_f]$. This is accomplished by discretizing time into a sequence $\{t_0,\ t_1,\ \dots,\ t_{N-1}\}$ (where $t_{N-1} = t_f$), and then approximating the respective DAE states $\{\vec{x}_0,\ \vec{x}_1,\ \dots,\ \vec{x}_{N-1}\}$ by solving a sequence of "Linear Multi-Step" (LMS) equations of the form [24, 25, 27, 28]:

$$\sum_{j=0}^{m_i}\Big(\alpha_i(-j)\ \vec{q}\,(\vec{x}_{i-j},\, \vec{p}) + \beta_i(-j)\ \vec{f}\,(\vec{x}_{i-j},\, \vec{u}(t_{i-j}),\, \vec{p})\Big) = \vec{0}. \tag{2}$$

Thus, at each step i (where $1 \le i \le N-1$), one solves Eq. (2) (using techniques like Newton-Raphson iteration, homotopy, *etc.*) to determine $\vec{x}_i$, based on m_i previously calculated $\vec{x}$ values (from earlier steps). See [29] for the α and β coefficients used in Eq. (2) by several common LMS methods, including Forward Euler (`FE`), Backward Euler (`BE`), Trapezoidal (`TRAP`), and second-order Gear (`GEAR2`) [24, 27, 30, 31].

C. Transient sensitivity analysis of DAEs

Suppose we have a DAE in the form of Eq. (1), with nominal parameters $\vec{p}^\star$, and transient solution $\vec{x}^\star(t)$ over the interval $[t_0, t_f]$ (using the convention that starred quantities denote nominal values, *i.e.*, those evaluated at $\vec{p}^\star$). The question behind transient sensitivity analysis is: if we perturb the parameters slightly, how will the transient solution change? More precisely, suppose we change the parameters from $\vec{p}^\star$ to $\vec{p}^\star + \Delta\vec{p}$, and as a result, the transient solution changes from $\vec{x}^\star(t)$ to $\vec{x}^\star(t) + \Delta\vec{x}(t)$. The question is: what is the relationship between $\Delta\vec{x}(t)$ and $\Delta\vec{p}$, in the limit as $\Delta\vec{p} \to \vec{0}$? The answer is obtained by doing a perturbation analysis of Eq. (1) [12–14]:

$$\Delta\vec{x}(t) = S_x(t)\,\Delta\vec{p}, \text{ where} \tag{3}$$

$$\left[\frac{d}{dt}\left(J^\star_{qx}(t)S_x(t) + J^\star_{qp}(t)\right)\right] + \left[J^\star_{fx}(t)S_x(t) + J^\star_{fp}(t)\right] = \mathbf{0}_{|\vec{x}^\star|\times|\vec{p}^\star|}. \tag{4}$$

The J terms in Eq. (4) denote nominal time-varying Jacobians, *i.e.*,

$$J^\star_{qx}(t) \triangleq \left.\frac{\partial\vec{q}}{\partial\vec{x}}\right|_{\vec{x}^\star(t),\,\vec{p}^\star},\quad J^\star_{qp}(t) \triangleq \left.\frac{\partial\vec{q}}{\partial\vec{p}}\right|_{\vec{x}^\star(t),\,\vec{p}^\star},$$
$$J^\star_{fx}(t) \triangleq \left.\frac{\partial\vec{f}}{\partial\vec{x}}\right|_{\vec{x}^\star(t),\,\vec{u}(t),\,\vec{p}^\star},\ \text{and}\ J^\star_{fp}(t) \triangleq \left.\frac{\partial\vec{f}}{\partial\vec{p}}\right|_{\vec{x}^\star(t),\,\vec{u}(t),\,\vec{p}^\star}. \tag{5}$$

Since $\Delta\vec{x}(t)$ is obtained by multiplying $S_x(t)$ with $\Delta\vec{p}$ (Eq. 3), $S_x(t)$ is called the *sensitivity* of $\vec{x}(t)$ with respect to $\vec{p}$, evaluated at $\vec{p}^\star$. And Eq. (4), a matrix-valued DAE that tracks the evolution of the $|\vec{x}^\star| \times |\vec{p}^\star|$ matrix $S_x(t)$ over time, is called the "sensitivity DAE".

Note that the sensitivity DAE does not directly give us $S_x(t)$. Rather, it needs to be *solved* for $S_x(t)$ (see [29] for more details).

D. The sensitivity DAG

Each step of the transient analysis of §II-B builds on previously computed DAE states, to solve for a new DAE state. This sequence of computations fits naturally into a DAG structure (Fig. 1), much like DAGs used in automatic differentiation [23], or Boolean function representation [32].

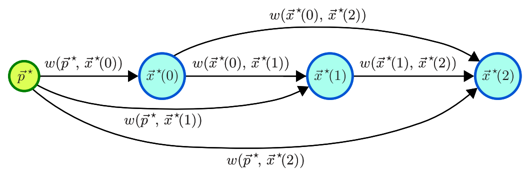

$w(\vec{x}^\star(0),\,\vec{x}^\star(2))$
$w(\vec{p}^\star,\,\vec{x}^\star(0))$
$w(\vec{x}^\star(0),\,\vec{x}^\star(1))$
$w(\vec{x}^\star(1),\,\vec{x}^\star(2))$
$\vec{p}^\star$
$\vec{x}^\star(0)$
$\vec{x}^\star(1)$
$\vec{x}^\star(2)$
$w(\vec{p}^\star,\,\vec{x}^\star(1))$
$w(\vec{p}^\star,\,\vec{x}^\star(2))$

Fig. 1. The DAG structure underlying a transient simulation.

The nodes of the "sensitivity DAG" in Fig. 1 represent the quantities that are computed during the transient simulation, and are labelled as such. The edges represent dependencies amongst these quantities. For example, Fig. 1 assumes that the initial condition $\vec{x}^\star(0)$ is computed from $\vec{p}^\star$ (*e.g.*, via DC analysis [24, 25]); so, there is an edge from the $\vec{p}^\star$ node to the $\vec{x}^\star(0)$ node. Similarly, $\vec{x}^\star(1)$ is assumed to be computed from $\vec{x}^\star(0)$ and $\vec{p}^\star$ by solving Eq. (2), using an LMS method with memory $m_1 = 1$, such as `FE`, `BE`, or `TRAP`. So, there are edges leading from both $\vec{p}^\star$ and $\vec{x}^\star(0)$ to $\vec{x}^\star(1)$. Finally, $\vec{x}^\star(2)$ is assumed to be computed via an LMS method like `GEAR2` that has memory $m_2 = 2$. Therefore, when Eq. (2) is solved to determine $\vec{x}^\star(2)$, both $\vec{x}^\star(0)$ and $\vec{x}^\star(1)$, as well as $\vec{p}^\star$, are used in the computation. So, there are edges from all these three nodes to $\vec{x}^\star(2)$.

While Fig. 1 stops at $\vec{x}^\star(2)$ for lack of space, the ideas behind the DAG construction can be extended to the entire length of the transient simulation. In general, if the simulation has N points, with indices $0 \le i \le N-1$, the corresponding sensitivity DAG will have $N+1$ nodes (one for $\vec{p}^\star$, and one for each $\vec{x}^\star(i)$). The $\vec{x}^\star(0)$ node will have exactly one incoming edge (from $\vec{p}^\star$). For all other $\vec{x}^\star(i)$, the number of incoming edges will be $1 + m_i$, where m_i is the memory of the LMS method used to compute $\vec{x}^\star(i)$ via Eq. (2). One of these edges will originate at $\vec{p}^\star$, while the others will originate at the m_i nodes prior to $\vec{x}^\star(i)$, *i.e.*, the nodes $\vec{x}^\star(i-j)$ for $1 \le j \le m_i$.

Also, each edge of the sensitivity DAG has a *weight* (Fig. 1). The weight of an edge from node u to node v, denoted $w(u, v)$, is equal to the partial derivative (or *sensitivity*) of v with respect to u; it measures how much a small perturbation in u will affect the value of v. The weight $w(\vec{p}^\star,\, \vec{x}^\star(0))$ is obtained by doing a DC perturbation analysis of Eq. (1) [24], while all other weights are obtained by doing a perturbation analysis of Eq. (2):

$$w(\vec{p}^\star,\, \vec{x}^\star(0)) = \frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star} = -J^\star_{fx}(0)^{-1}J^\star_{fp}(0), \tag{6}$$

$$w(\vec{p}^\star,\, \vec{x}^\star(i)) = \frac{\partial\vec{x}^\star(i)}{\partial\vec{p}^\star} = -\Big[\alpha_i(0)J^\star_{qx}(i) + \beta_i(0)J^\star_{fx}(i)\Big]^{-1}$$
$$\left[\sum_{j=0}^{m_i}\Big(\alpha_i(-j)J^\star_{qp}(i-j) + \beta_i(-j)J^\star_{fp}(i-j)\Big)\right],$$
$$\forall\ 1 \le i \le N-1, \text{ and} \tag{7}$$

$$w(\vec{x}^\star(i-j),\, \vec{x}^\star(i)) = \frac{\partial\vec{x}^\star(i)}{\partial\vec{x}^\star(i-j)} = -\Big[\alpha_i(0)J^\star_{qx}(i) + \beta_i(0)J^\star_{fx}(i)\Big]^{-1}$$
$$\Big[\alpha_i(-j)J^\star_{qx}(i-j) + \beta_i(-j)J^\star_{fx}(i-j)\Big],$$
$$\forall\ 1 \le i \le N-1,\ 1 \le j \le m_i. \tag{8}$$

E. Objective functions and the sensitivity DAG

In many applications, we are *not* directly interested in the sensitivities of $\vec{x}^\star(t)$, but would like to compute the sensitivities of important transient performance metrics (*i.e.*, "objective functions") *derived* from $\vec{x}^\star(t)$ (and denoted $\vec{\phi}^\star$). Below, we discuss two kinds of objective functions commonly found in the sensitivity analysis literature ("final point" and "integral" objectives), and show how to add these to the sensitivity DAG.

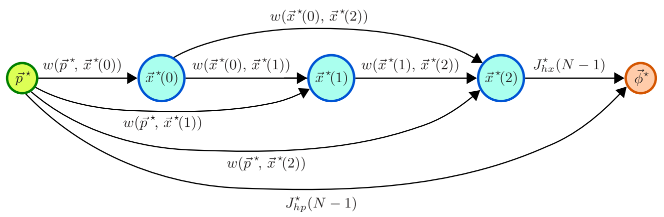
Final point objectives. These take the form [4, 14]:

$$\vec{\phi}^\star = \vec{h}(\vec{x}^\star(N-1),\, \vec{p}^\star), \tag{9}$$

where $\vec{x}^\star(N-1)$ is the final point in the transient simulation. Note that, if $\vec{h}(.,\,.)$ needs to be evaluated at multiple time-points, then each needs to be considered a separate objective, which will increase the dimension of the objective function. The *sensitivity* S_ϕ of a final point objective function, evaluated at $\vec{p}^\star$, is given by:

$$S_\phi = J^\star_{hx}(N-1)S_x(N-1) + J^\star_{hp}(N-1), \tag{10}$$

where the Jacobian symbols have their usual meanings.



$w(\vec{x}^\star(0),\, \vec{x}^\star(2))$
$w(\vec{p}^\star,\, \vec{x}^\star(0))$
$\vec{p}^\star$
$\vec{x}^\star(0)$
$w(\vec{x}^\star(0),\, \vec{x}^\star(1))$
$\vec{x}^\star(1)$
$w(\vec{x}^\star(1),\, \vec{x}^\star(2))$
$\vec{x}^\star(2)$
$J^\star_{hx}(N-1)$
$\vec{\phi}^\star$
$w(\vec{p}^\star,\, \vec{x}^\star(1))$
$w(\vec{p}^\star,\, \vec{x}^\star(2))$
$J^\star_{hp}(N-1)$

Fig. 2. Adding a final point objective function to the sensitivity DAG.

To add such an objective function to the sensitivity DAG, we add a new node $\vec{\phi}^\star$ with two incoming edges: one from $\vec{p}^\star$ with weight $J^\star_{hp}(N-1)$, and one from $\vec{x}^\star(N-1)$ with weight $J^\star_{hx}(N-1)$ (as shown in Fig. 2 for $N=3$).

Integral objectives. These take the form [13, 16]:

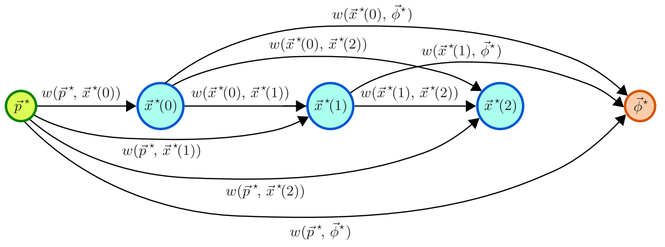
$$\vec{\phi}^\star = \int_{t=t_0}^{t_f} \vec{h}(t,\, \vec{x}^\star(t),\, \vec{p}^\star)\; dt. \tag{11}$$

Therefore, we have:

$$S_\phi = \int_{t=t_0}^{t_f} \left(J^\star_{hx}(t)S_x(t) + J^\star_{hp}(t)\right)\; dt. \tag{12}$$

In practice, the integral in Eq. (12) is approximated by a summation:

$$S_\phi \approx \sum_{i=0}^{N-2} \left[\left(J^\star_{hx}(i)S_x(i) + J^\star_{hp}(i)\right)(t_{i+1} - t_i)\right]. \tag{13}$$



$w(\vec{x}^\star(0),\, \vec{\phi}^\star)$
$w(\vec{x}^\star(0),\, \vec{x}^\star(2))$
$w(\vec{x}^\star(1),\, \vec{\phi}^\star)$
$w(\vec{p}^\star,\, \vec{x}^\star(0))$
$\vec{p}^\star$
$\vec{x}^\star(0)$
$w(\vec{x}^\star(0),\, \vec{x}^\star(1))$
$\vec{x}^\star(1)$
$w(\vec{x}^\star(1),\, \vec{x}^\star(2))$
$\vec{x}^\star(2)$
$\vec{\phi}^\star$
$w(\vec{p}^\star,\, \vec{x}^\star(1))$
$w(\vec{p}^\star,\, \vec{x}^\star(2))$
$w(\vec{p}^\star,\, \vec{\phi}^\star)$

Fig. 3. Adding an integral objective function to the sensitivity DAG.

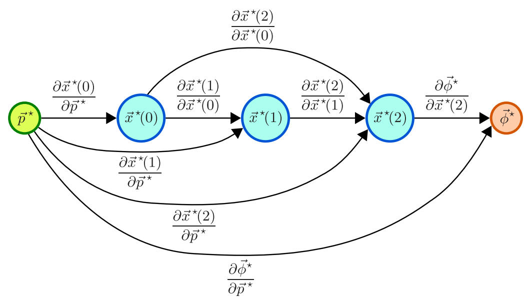
To add such an objective function to the sensitivity DAG, we add a new node $\vec{\phi}^\star$, with N incoming edges: one from $\vec{p}^\star$, and the rest from $\vec{x}^\star(i)$, where $0 \le i \le N-2$ (*i.e.*, from every point in the transient simulation except the last, as illustrated in Fig. 3 for $N=3$). The weights of these edges are:

$$w(\vec{p}^\star,\, \vec{\phi}^\star) = \sum_{i=0}^{N-2}\left(J^\star_{hp}(i)(t_{i+1}-t_i)\right), \text{ and} \tag{14}$$

$$w(\vec{x}^\star(i),\, \vec{\phi}^\star) = J^\star_{hx}(i)(t_{i+1}-t_i),\ \forall\ 0 \le i \le N-2. \tag{15}$$

With the introduction of a node representing the objective function, the sensitivity DAG is "complete": it now accurately represents all the intermediate computations involved in calculating the objective function starting from the DAE parameters. Moreover, the partial sensitivities of these computations are also available from the DAG's edge-weights. Thus, we now have all the information needed to do an end-to-end sensitivity analysis.

F. Sensitivity analysis = DAG path enumeration



$\frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(0)}$
$\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}$
$\frac{\partial\vec{x}^\star(1)}{\partial\vec{x}^\star(0)}$
$\frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}$
$\frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}$
$\vec{p}^\star$
$\vec{x}^\star(0)$
$\vec{x}^\star(1)$
$\vec{x}^\star(2)$
$\vec{\phi}^\star$
$\frac{\partial\vec{x}^\star(1)}{\partial\vec{p}^\star}$
$\frac{\partial\vec{x}^\star(2)}{\partial\vec{p}^\star}$
$\frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star}$

Fig. 4. The DAG of Fig. 2, with edge-weights denoted by partial derivatives.

Our goal is to compute the sensitivity of the objective function $\vec{\phi}$ with respect to the DAE parameters $\vec{p}$, evaluated at $\vec{p}^\star$. Let us take a closer look at this computation, through an example. Fig. 4 shows the sensitivity DAG for a 3-step transient simulation and a final point objective function. This is the same DAG from Fig. 2, except that we changed the edge-weight notation to make it clear that the edge-weights are, in fact, partial derivatives. Now, we repeatedly apply the chain rule of differentiation to find the sensitivity of the objective function:

$$\begin{aligned}
\begin{matrix}\text{Sensitivity}\\ \text{we need}\end{matrix} &= \left.\frac{d\vec{\phi}}{d\vec{p}}\right|_{\vec{p}^\star} = \underbrace{\frac{d\vec{\phi}^\star}{d\vec{p}^\star}}_{\text{Chain Rule}} = \frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star} + \frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\underbrace{\frac{d\vec{x}^\star(2)}{d\vec{p}^\star}}_{\text{Chain Rule}} \\
&= \frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star} + \frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\left(\frac{\partial\vec{x}^\star(2)}{\partial\vec{p}^\star} + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}\underbrace{\frac{d\vec{x}^\star(1)}{d\vec{p}^\star}}_{\text{Chain Rule}} + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(0)}\underbrace{\frac{d\vec{x}^\star(0)}{d\vec{p}^\star}}_{\text{Chain Rule}}\right) \\
&= \frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star} + \frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\left(\frac{\partial\vec{x}^\star(2)}{\partial\vec{p}^\star} + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}\left(\frac{\partial\vec{x}^\star(1)}{\partial\vec{p}^\star} + \frac{\partial\vec{x}^\star(1)}{\partial\vec{x}^\star(0)}\underbrace{\frac{d\vec{x}^\star(0)}{d\vec{p}^\star}}_{\text{Chain Rule}}\right) + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(0)}\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}\right) \\
&= \frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star} + \frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\left(\frac{\partial\vec{x}^\star(2)}{\partial\vec{p}^\star} + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}\left(\frac{\partial\vec{x}^\star(1)}{\partial\vec{p}^\star} + \frac{\partial\vec{x}^\star(1)}{\partial\vec{x}^\star(0)}\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}\right) + \frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(0)}\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}\right) \\
&= \Bigg[\underbrace{\frac{\partial\vec{\phi}^\star}{\partial\vec{p}^\star}}_{\text{Path: }\vec{p}^\star\to\vec{\phi}^\star} + \underbrace{\frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\frac{\partial\vec{x}^\star(2)}{\partial\vec{p}^\star}}_{\text{Path: }\vec{p}^\star\to\vec{x}^\star(2)\to\vec{\phi}^\star} + \underbrace{\frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}\frac{\partial\vec{x}^\star(1)}{\partial\vec{p}^\star}}_{\text{Path: }\vec{p}^\star\to\vec{x}^\star(1)\to\vec{x}^\star(2)\to\vec{\phi}^\star} \\
&\qquad + \underbrace{\frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(1)}\frac{\partial\vec{x}^\star(1)}{\partial\vec{x}^\star(0)}\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}}_{\text{Path: }\vec{p}^\star\to\vec{x}^\star(0)\to\vec{x}^\star(1)\to\vec{x}^\star(2)\to\vec{\phi}^\star} + \underbrace{\frac{\partial\vec{\phi}^\star}{\partial\vec{x}^\star(2)}\frac{\partial\vec{x}^\star(2)}{\partial\vec{x}^\star(0)}\frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star}}_{\text{Path: }\vec{p}^\star\to\vec{x}^\star(0)\to\vec{x}^\star(2)\to\vec{\phi}^\star}\Bigg] \\
&= \sum_{\substack{\pi\ \mid\ \pi \text{ is a path from}\\ \vec{p}^\star \text{ to } \vec{\phi}^\star \text{ in the sensitivity DAG}}} \Big(\text{Product of edge-weights of } \pi \text{ in reverse}\Big).
\end{aligned}$$

The derivation above shows that the sensitivity is a sum of terms, where each term corresponds to a unique path from $\vec{p}^\star$ to $\vec{\phi}^\star$ in the sensitivity DAG; more precisely, each term is a "product of edge-weights in reverse" of some path from $\vec{p}^\star$ to $\vec{\phi}^\star$. Thus, we have a key insight: *solving the sensitivity analysis problem is the same as enumerating paths in the sensitivity DAG.*

Taking a cue from this, we define the "weight of a path" in the sensitivity DAG to be the product of weights of all the edges along the path, in reverse. Also, given any two nodes u and v in the DAG, we define $\sigma(u, v)$ to be the sum of the weights of all the paths in the DAG that start at u and end at v. Thus, solving the sensitivity analysis problem is the same as computing $\sigma(\vec{p}^\star,\, \vec{\phi}^\star)$ in the sensitivity DAG.

G. Direct and Adjoint approaches to DAG path enumeration

We just reduced sensitivity analysis to the problem of adding up the weights of all paths from $\vec{p}^\star$ to $\vec{\phi}^\star$ in the sensitivity DAG. The brute-force approach to this, however, is computationally infeasible because the number of such paths grows exponentially as the size of the DAG [20, 21]. Therefore, we use dynamic programming techniques to efficiently enumerate DAG paths, and hence solve the sensitivity analysis problem in linear time in the size of the DAG [20, 21].

Fig. 5 illustrates the key idea that we exploit, which is that the problem of computing $\sigma(\vec{p}^\star, \vec{\phi}^\star)$ can be repeatedly broken down into smaller, simpler, sub-problems. There are 2 ways to do this: (1) the "direct" approach (Fig. 5a), where we keep the source $\vec{p}^\star$ constant, and express $\sigma(\vec{p}^\star, \vec{\phi}^\star)$ in terms of $\sigma(\vec{p}^\star, u)$, where u is one step closer to $\vec{p}^\star$ than $\vec{\phi}^\star$, or (2) the "adjoint" approach (Fig. 5b), where we keep the destination $\vec{\phi}^\star$ constant, and express $\sigma(\vec{p}^\star, \vec{\phi}^\star)$ in terms of $\sigma(v, \vec{\phi}^\star)$, where v is one step closer to $\vec{\phi}^\star$ than $\vec{p}^\star$. Both approaches give us optimal sub-structures for dynamic programming, as formalised in Algorithms 1 and 2 respectively.

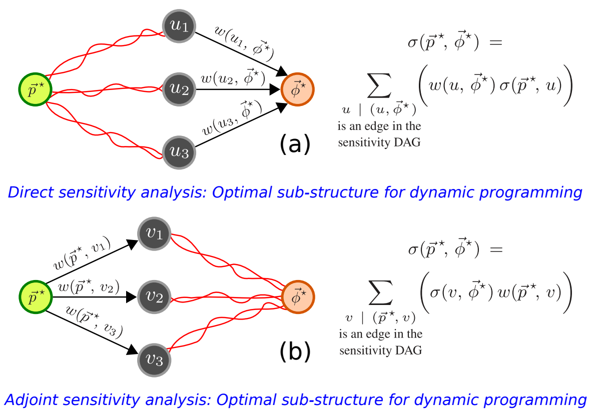

u_1
u_2
u_3
$\vec{p}^\star$
$\vec{\phi}^\star$
$w(u_1, \vec{\phi}^\star)$
$w(u_2, \vec{\phi}^\star)$
$w(u_3, \vec{\phi}^\star)$
$\sigma(\vec{p}^\star, \vec{\phi}^\star) = \sum_{u \mid (u, \vec{\phi}^\star) \text{ is an edge in the sensitivity DAG}} \left(w(u, \vec{\phi}^\star)\, \sigma(\vec{p}^\star, u) \right)$
(a)
Direct sensitivity analysis: Optimal sub-structure for dynamic programming
v_1
v_2
v_3
$w(\vec{p}^\star, v_1)$
$w(\vec{p}^\star, v_2)$
$w(\vec{p}^\star, v_3)$
$\sigma(\vec{p}^\star, \vec{\phi}^\star) = \sum_{v \mid (\vec{p}^\star, v) \text{ is an edge in the sensitivity DAG}} \left(\sigma(v, \vec{\phi}^\star)\, w(\vec{p}^\star, v) \right)$
(b)
Adjoint sensitivity analysis: Optimal sub-structure for dynamic programming

Fig. 5. The key ideas behind efficient direct and adjoint DAG path enumeration in DAGSENS.

Algorithm 1: Direct transient sensitivity analysis in DAGSENS

```
Input: The sensitivity DAG $G$, with nodes $\vec{p}^\star$ and $\vec{\phi}^\star$ representing the
       DAE parameters and the objective function respectively
Output: The sensitivity $\sigma(\vec{p}^\star, \vec{\phi}^\star)$, calculated via dynamic programming
        using the "direct" optimal sub-structure (Fig. 5a)
1 $\sigma(\vec{p}^\star, \vec{p}^\star) = \mathbf{I}_{|\vec{p}^\star| \times |\vec{p}^\star|}$   // identity matrix
2 order = topological_sort($G$)
3 for $u$ in order do
4     $\sigma(\vec{p}^\star, u) = \mathbf{0}_{|u| \times |\vec{p}^\star|}$
5     for $v$ such that $(v, u)$ is an edge in $G$ do
6         $\sigma(\vec{p}^\star, u)$ += $w(v, u)\ \sigma(\vec{p}^\star, v)$
7 return $\sigma(\vec{p}^\star, \vec{\phi}^\star)$
```

Algorithm 2: Adjoint transient sensitivity analysis in DAGSENS

```
Input: The sensitivity DAG $G$, with nodes $\vec{p}^\star$ and $\vec{\phi}^\star$ representing the
       DAE parameters and the objective function respectively
Output: The sensitivity $\sigma(\vec{p}^\star, \vec{\phi}^\star)$, calculated via dynamic programming
        using the "adjoint" optimal sub-structure (Fig. 5b)
1 $\sigma(\vec{\phi}^\star, \vec{\phi}^\star) = \mathbf{I}_{|\vec{\phi}^\star| \times |\vec{\phi}^\star|}$   // identity matrix
2 order = reversed(topological_sort($G$))
3 for $v$ in order do
4     $\sigma(v, \vec{\phi}^\star) = \mathbf{0}_{|\vec{\phi}^\star| \times |v|}$
5     for $u$ such that $(v, u)$ is an edge in $G$ do
6         $\sigma(v, \vec{\phi}^\star)$ += $\sigma(u, \vec{\phi}^\star)\ w(v, u)$
7 return $\sigma(\vec{p}^\star, \vec{\phi}^\star)$
```

Algorithms 1 and 2 both compute a topological ordering, *i.e.*, a permutation of the DAG nodes such that if (u, v) is a DAG edge, then u occurs before v in the permutation [20, 21]. But while Algorithm 1 traverses the nodes in topological order, Algorithm 2 traverses them in *reverse* topological order. At every such node u (v), Algorithm 1 (2) computes $\sigma(\vec{p}^\star, u)$ ($\sigma(v, \vec{\phi}^\star)$), making use of the optimal sub-structure logic in Fig. 5a (5b). This continues until, finally, the $\vec{\phi}^\star$ ($\vec{p}^\star$) node is reached, at which time the computed $\sigma(\vec{p}^\star, \vec{\phi}^\star)$ is returned as the required sensitivity. Thus, while Algorithm 1 involves computing the sensitivity of each intermediate DAG node with respect to the *DAE parameters*, Algorithm 2 involves computing the sensitivity of the *objective function* with respect to each intermediate DAG node. So, the former (latter) implements direct (adjoint) sensitivity analysis in DAGSENS [4, 12–16].

Finally, Line 6 of Algorithm 2 involves pre-multiplying the edge-weight $w(v, u)$ by the matrix $\sigma(u, \vec{\phi}^\star)$. Since most edge-weights are of the form $B^{-1}C$ (Eqs. 6, 7, and 8), this is a computation of the form $AB^{-1}C$, which can be done either as $A(B^{-1}C)$, or as $(AB^{-1})C = ((B^T)^{-1}A^T)^T C$ (where matrix/matrix solves are *sparse* in many applications of interest). If the number of rows of A is much smaller than the number of columns of C (*i.e.*, the dimension of the objective function is much smaller than that of the DAE parameter space), the latter is likely to be much more efficient than the former, which we exploit heavily in DAGSENS.

H. Event-driven objective functions

As mentioned in §I, we would like to define "events" that happen during a transient simulation (*e.g.*, a node voltage crossing a particular threshold, a PLL in a circuit achieving lock, *etc.*), and then compute sensitivities of objectives that are based on these events. For our purposes, an "event" *ev* is specified by the condition:

$$g_{ev}(\tau^\star_{ev}, \vec{x}^\star(\tau^\star_{ev}), \vec{p}^\star) = 0, \tag{16}$$

where $\tau^\star_{ev}$ is the time at which the event occurs during a transient simulation. In practice, we may need additional constraints to uniquely identify the event (such as limits on $\tau^\star_{ev}$, or a specification such as "the third time Eq. (16) is satisfied", *etc.*). But for sensitivity analysis, we can ignore these additional specifications because we only do perturbation analysis of Eq. (16) in a small neighbourhood around $\tau^\star_{ev}$. Also, we note that events corresponding to a signal reaching its maximum/minimum value are specified by adding a new DAE state variable representing the derivative of the signal in question, and by equating this variable to zero via Eq. (16) (see §III for examples).

Given a sequence of M events $1 \leq ev \leq M$, our event-driven objective function takes the form:

$$\vec{\phi}^\star = \vec{h}(\tau^\star_1, \tau^\star_2, \ldots, \tau^\star_M, \vec{x}^\star(\tau^\star_1), \vec{x}^\star(\tau^\star_2), \ldots, \vec{x}^\star(\tau^\star_M), \vec{p}^\star). \tag{17}$$

Thus, event-driven objective functions depend on the times of occurrence of a set of events, as well as the DAE states at these times, *both* of which change when the DAE parameters are perturbed.

I. Sensitivity analysis of event-driven objective functions

Let us denote by $S_{\tau_{ev}}$ and $S_{x_{ev}}$, where $1 \leq ev \leq M$, the sensitivities of our event times, and DAE states at these times, respectively. Note that $S_{x_{ev}} \neq S_x(\tau^\star_{ev})$; while $S_x(\tau^\star_{ev})$ only takes into account the sensitivity of the DAE state $\vec{x}$, $S_{x_{ev}}$ takes into account the sensitivities of *both* the DAE state $\vec{x}$ *and* the event time τ_{ev}. With this distinction in mind, perturbation analysis of Eq. (16) yields:

$$S_{\tau_{ev}} = -\frac{J^\star_{g_{ev}x} S_x(\tau^\star_{ev}) + J^\star_{g_{ev}p}}{J^\star_{g_{ev}x} \dot{\vec{x}}^\star(\tau^\star_{ev}) + J^\star_{g_{ev}\tau}}, \text{ and} \tag{18}$$

$$S_{x_{ev}} = S_x(\tau^\star_{ev}) + \dot{\vec{x}}^\star(\tau^\star_{ev}) S_{\tau_{ev}}, \text{ where} \tag{19}$$

$$\dot{\vec{x}}^\star(\tau^\star_{ev}) = \left. \frac{d}{dt} \vec{x}^\star(t) \right|_{\tau^\star_{ev}}, \ \forall\ 1 \leq ev \leq M, \tag{20}$$

and where Jacobian terms have their usual meanings, and are all evaluated at $(\tau^\star_{ev}, \vec{x}^\star(\tau^\star_{ev}), \vec{p}^\star)$. Therefore, we first need to *solve for the event*, *i.e.*, find $\tau^\star_{ev}$ and $\vec{x}^\star(\tau^\star_{ev})$, before we can compute $S_{\tau_{ev}}$ and $S_{x_{ev}}$. We do this by finding a time-point t_i of the transient simulation where the $g_{ev}(., ., .)$ function undergoes a sign change between t_i and t_{i+1}. We then solve the following "modified" LMS equations:

$$\left(\alpha_{ev}(0, \tau^\star_{ev})\ \vec{q}(\vec{x}^\star(\tau^\star_{ev}), \vec{p}^\star) + \beta_{ev}(0, \tau^\star_{ev})\ \vec{f}(\vec{x}^\star(\tau^\star_{ev}), \vec{u}(\tau^\star_{ev}), \vec{p}^\star) \right)$$

$$+ \sum_{j=1}^{m_{ev}} \left(\alpha_{ev}(-j, \tau^\star_{ev})\ \vec{q}(\vec{x}^\star(i - j + 1), \vec{p}^\star) \right.$$

$$+ \beta_{ev}(-j, \tau_{ev}^{\star})\ \vec{f}\,(\vec{x}^{\,\star}(i-j+1),\ \vec{u}(t_{i-j+1}),\ \vec{p}^{\,\star}) \Big) = \vec{0},\ \text{and} \tag{21}$$

$$g_{ev}(\tau_{ev}^{\star},\ \vec{x}^{\,\star}(\tau_{ev}^{\star}),\ \vec{p}^{\,\star}) = 0. \tag{22}$$

These equations are similar to Eq. (2). But here, we treat *both* the next time-point $\tau_{ev}^{\star}$ *and* the next DAE state $\vec{x}^{\,\star}(\tau_{ev}^{\star})$ as unknowns. So, the α and β coefficients become functions of the unknowns as well. Also, we use LMS approximations to calculate the $\dot{\vec{x}}^{\,\star}(\tau_{ev}^{\star})$ term in Eqs. (18) and (19).

Finally, the sensitivity of our event-driven objective function is given by:

$$S_{\phi} = J_{hp}^{\star} + \sum_{ev=1}^{M} \left(J_{h\tau_{ev}}^{\star} S_{\tau_{ev}} + J_{hx(\tau_{ev})}^{\star} S_{x_{ev}} \right). \tag{23}$$

J. Augmenting the sensitivity DAG for event-driven objective functions

Fig. 6 shows how to add an event-driven objective function to the sensitivity DAG. For each event $1 \le ev \le M$, we add three new nodes (Fig. 6a): a *partial* $\vec{x}^{\,\star}(\tau_{ev}^{\star})$ node whose sensitivity equals $S_x(\tau_{ev}^{\star})$ (which is created just like any other node in the transient simulation, following §II-D), and nodes corresponding to $\tau_{ev}^{\star}$ and $\vec{x}^{\,\star}(\tau_{ev}^{\star})$, which are created according to Eqs. (18) and (19) respectively. The edges associated with these nodes, and their weights, are shown in Fig. 6a.

Finally, we add a new node $\vec{\phi}^{\star}$ to capture the event-driven objective function. As shown in Fig. 6b, this node has incoming edges from $\vec{p}^{\,\star}$, as well as from all the $\tau_{ev}^{\star}$ and $\vec{x}^{\,\star}(\tau_{ev}^{\star})$ nodes above. The weights of these edges, as shown in the figure, follow Eq. (23).

K. DAGSENS: The overall flow for event-driven objective functions

Based on the preceding sections, Algorithm 3 outlines the overall flow that DAGSENS uses for computing direct and adjoint sensitivities of event-driven objective functions.

```
Algorithm 3: Event-driven sensitivity analysis in DAGSENS
  Input: A DAE $D$ in the form of Eq. (1), nominal DAE parameters $\vec{p}^{\,\star}$,
         DAE inputs $\vec{u}(t)$ over an interval $[t_0, t_f]$, events $1 \le ev \le M$ in
         the form of Eq. (16), and an event-driven objective function $\phi$ in
         the form of Eq. (17)
  Output: The sensitivity of the objective function with respect to the
          DAE parameters, evaluated at $\vec{p}^{\,\star}$
1 Do a transient analysis of $D$, using parameters $\vec{p}^{\,\star}$, with inputs $\vec{u}(t)$,
    over the time-interval $[t_0, t_f]$.
2 Record Jacobians $J_{qx}^{\star}(t)$, $J_{qp}^{\star}(t)$, $J_{fx}^{\star}(t)$, and $J_{fp}^{\star}(t)$ from the transient
    simulation.
3 Build a sensitivity DAG $G$, using information from the transient run and
    the Jacobians above, via Eqs. (6), (7), and (8).
4 for $1 \le ev \le M$ do
5     Solve for event $ev$, i.e., find $\tau_{ev}^{\star}$ and $\vec{x}^{\,\star}(\tau_{ev}^{\star})$, by constructing and
        solving Eqs. (21) and (22).
6     Augment the sensitivity DAG with nodes corresponding to $ev$, as
        outlined in §II-J.
7 Augment the sensitivity DAG with a $\vec{\phi}^{\star}$ node, as outlined in §II-J.
8 Traverse the sensitivity DAG using either Algorithm 1 (for direct
    sensitivities), or Algorithm 2 (for adjoint sensitivities).
9 Return the sensitivities computed above.
```

III. Results

We have developed a Python implementation of DAGSENS, which we now use to compute event-driven sensitivities in some electronic and biological applications, including high-speed communication, statistical cell library characterization, and gene expression in *Drosophila* embryos.

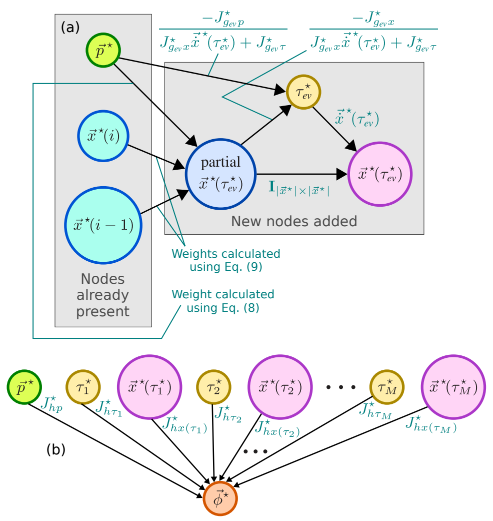

(a)
$\vec{p}^{\,\star}$
$\frac{-J_{g_{ev}p}^{\star}}{J_{g_{ev}x}^{\star}\dot{\vec{x}}^{\,\star}(\tau_{ev}^{\star}) + J_{g_{ev}\tau}^{\star}}$
$\frac{-J_{g_{ev}x}^{\star}}{J_{g_{ev}x}^{\star}\dot{\vec{x}}^{\,\star}(\tau_{ev}^{\star}) + J_{g_{ev}\tau}^{\star}}$
$\tau_{ev}^{\star}$
$\dot{\vec{x}}^{\,\star}(\tau_{ev}^{\star})$
$\vec{x}^{\,\star}(i)$
partial $\vec{x}^{\,\star}(\tau_{ev}^{\star})$
$\mathbf{I}_{|\vec{x}^{\star}|\times|\vec{x}^{\star}|}$
$\vec{x}^{\,\star}(\tau_{ev}^{\star})$
$\vec{x}^{\,\star}(i-1)$
New nodes added
Weights calculated using Eq. (9)
Nodes already present
Weight calculated using Eq. (8)
(b)
$\vec{p}^{\,\star}$ $\tau_1^{\star}$ $\vec{x}^{\,\star}(\tau_1^{\star})$ $\tau_2^{\star}$ $\vec{x}^{\,\star}(\tau_2^{\star})$ $\tau_M^{\star}$ $\vec{x}^{\,\star}(\tau_M^{\star})$
$J_{hp}^{\star}$ $J_{h\tau_1}^{\star}$ $J_{hx(\tau_1)}^{\star}$ $J_{h\tau_2}^{\star}$ $J_{hx(\tau_2)}^{\star}$ $J_{h\tau_M}^{\star}$ $J_{hx(\tau_M)}^{\star}$
$\vec{\phi}^{\star}$

Fig. 6. Adding (a) events, and (b) an event-driven objective, to the sensitivity DAG.

A. High-speed communication sub-systems

1) A "maximum crosstalk" example: In modern high-speed I/O links, "crosstalk" between parallel channels (*e.g.*, in a CPU/DRAM interface) often adversely impacts bandwidth [33–35]. When two signal-carrying lines lie physically close together on-chip, the bits transported in one of the lines (the *aggressor*) often interfere with those in the other line (the *victim*), via cross-coupled capacitances [33–35]. Fig. 7a shows the circuit that we designed to tease out the impact of such crosstalk. The aggressor and victim are both modelled as RC chains driving capacitive loads. The circuit has two sub-circuits: the right one where crosstalk is modelled via cross-coupled capacitances, and the left one without crosstalk. The difference between the victim's outputs in these two sub-circuits is a measure of crosstalk (Fig. 7a).

Our "event" of interest is when the crosstalk reaches its maximum value during a transient run. And our event-driven objective function ϕ is the value of this maximum crosstalk. Parts (b) and (c) of Fig. 7 depict these events during a transient simulation, where the aggressor and victim transmit their bits without and with pre/de-emphasis respectively. While pre/de-emphasis is a good strategy for boosting bandwidth by improving signal integrity at the receiver, it can have the drawback of increasing crosstalk [33–35].

	Parameter	Without Pre/De-Emphasis	With Pre/De-Emphasis	% impact of Pre/De-Emphasis
ϕ (V)		0.1183	0.1372	15.98%
Sens(ϕ)	Total R_{seg} $(k\Omega)$	0.6611	0.5219	−21.06%
	Total C_{seg} (pF)	0.7770	0.7851	1.04%
	Total C_{cross} (pF)	3.9643	4.7049	18.68%
	C_{load} (pF)	4.0547	4.7960	18.28%

Table 1. The impact of using pre/de-emphasis on the sensitivities of maximum crosstalk (ϕ), with respect to total segment resistance, total segment capacitance, total cross capacitance, and load capacitance.

Parts (d) to (i) of Fig. 7 show the results of applying DAGSENS to the system above, where the sensitivities of the maximum crosstalk with respect to each segment resistance, segment capacitance, and coupling capacitance are plotted as bar charts. It is interesting to see (parts d, e) that the maximum crosstalk is more sensitive to the first few segment resistances when pre/de-emphasis is employed. Also, it is interesting that the sensitivities with respect to segment capacitances rise in a *convex* manner (parts f, g), while those with respect to coupling capacitances rise in a *concave* manner (parts h, i). Table 1 shows the precise impact of using pre/de-emphasis on maximum crosstalk sensitivities with respect to various system and load parameters. Thus, event-driven DAGSENS can allow high-speed link engineers to obtain insights that would not be possible with existing sensitivity analysis tools.

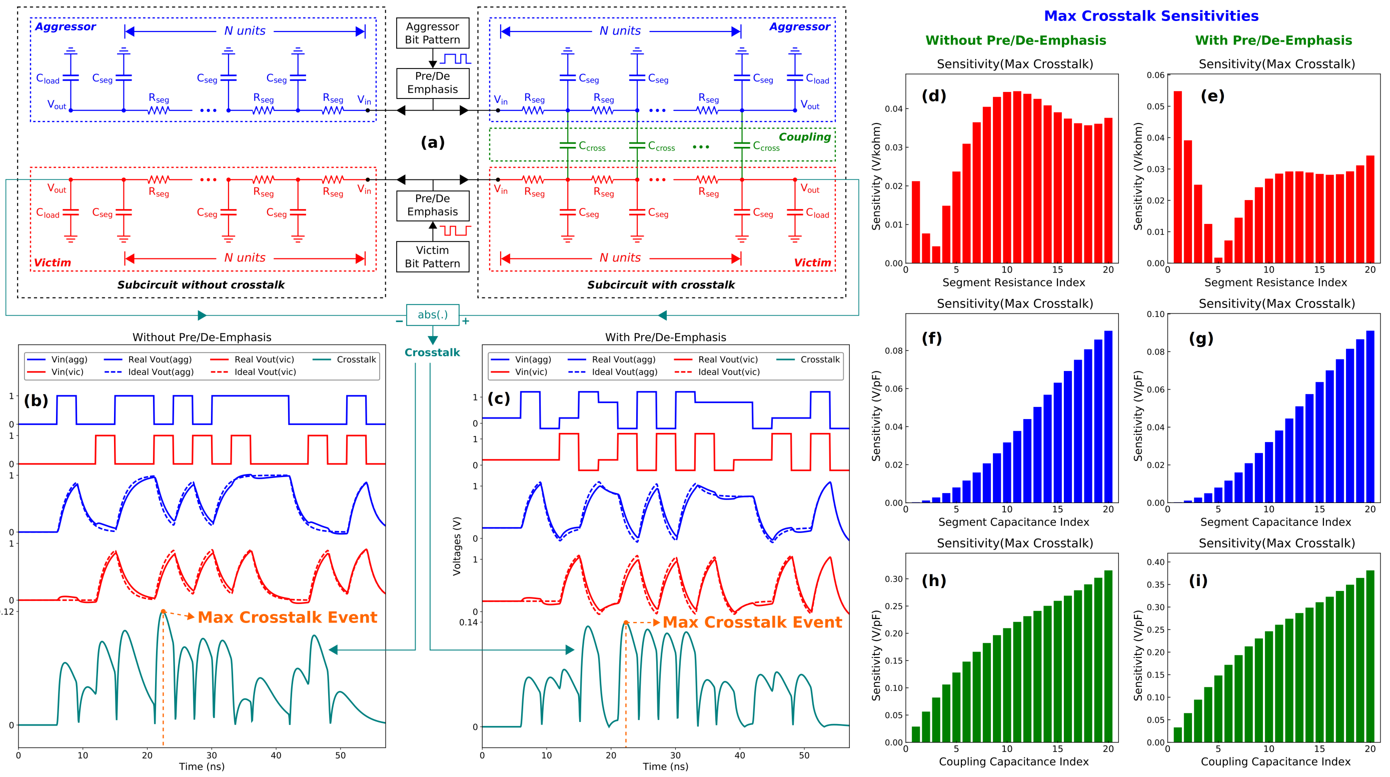


Fig. 7. (a) The circuit used to determine the magnitude of crosstalk induced by an aggressor on a victim. (b, c) Transient simulation of the circuit in (a) without and with pre/de-emphasis respectively, with the event corresponding to maximum crosstalk in each case. (d through i) Sensitivities of the maximum crosstalk with respect to each segment resistance (d, e), segment capacitance (f, g), and coupling capacitance (h, i), without (d, f, h) and with (e, g, i) pre/de-emphasis.

Also, in this example, our objective function has dimension 1 whereas the DAE parameter space has dimension $O(3N)$ (where N is the number of RC segments). So, this is a good test case to illustrate the benefits of adjoint over direct sensitivity analysis. We do not provide these results here due to space constraints, but refer the reader to [29] instead.

2) A PLL example: PLLs are widely used in high-speed communication sub-systems for frequency synthesis, clock and data recovery (CDR), *etc.* [33, 36, 37]. The lock time of a PLL, *i.e.*, how quickly a PLL can lock to a new input frequency, is critical in these applications. Since a PLL achieving lock is a transient event, we can use DAGSENS to calculate the sensitivities of a PLL's lock time with respect to its parameters.

		Low Bandwidth ($f_c = 0.11$GHz) Loop Filter		High Bandwidth ($f_c = 0.45$GHz) Loop Filter	
	Parameter	Lock time (ns)	V_{ctl} swing (mV)	Lock time (ns)	V_{ctl} swing (mV)
ϕ		14.85	45.41	1.68	182.94
Sens(ϕ)	K_{PFD} (V^{-1})	−1.95	45.76	−0.17	188.67
	R ($k\Omega$)	1.47	−32.61	0.27	−259.66
	C (pF)	2.06	−45.65	0.37	−363.53
	K_{VCO} (V^{-1}GHz)	−1.95	0.35	−0.17	5.72
	f_{VCO} (GHz)	−5.12	−0.01	−0.21	0.93

Table 2. Sensitivities of PLL lock times and peak-to-peak V_{ctl} swings at lock, with respect to various macromodel parameters, for low and high bandwidth loop filters.

Fig. 8a shows a high-level block-diagram for a PLL, and also the equations and parameters associated with each component [36, 37]. Parts (b) and (c) of Fig. 8 show transient simulations of two PLLs, one with a low-bandwidth loop filter (b) and the other with a high-bandwidth loop filter (c). In each case, the input waveform abruptly switches its frequency at $t = 50$ns, throwing the PLLs off lock. The PLLs eventually regain lock, as can be seen from the red bars that graph the time elapsed between the peaks of V_{in} (the PLL input) and the nearest peaks of V_{out} (the PLL output) in each case. Our event-driven objective functions are the respective PLL lock times, defined as the time taken for the respective V_{ctl} waveforms to settle into a narrow range around their final expected values, as well as the peak-to-peak swings in V_{ctl} at lock. If one used an ideal loop filter, V_{ctl} would settle to a DC value, so the swing in V_{ctl} is a measure of non-ideality in the PLL's response. While we would like PLLs to lock quickly and have small V_{ctl} swings, there is often a tradeoff between these: high (low) bandwidth PLLs lock quickly (slowly), but exhibit larger (smaller) V_{ctl} swings, as shown in parts (b) and (c) of Fig. 8. Table 2 shows the sensitivities of the event-driven objectives above (PLL lock times as well as V_{ctl} swings at lock), with respect to the PLL macromodel parameters shown in Fig. 8a. From the table, it is clear that when a high (low) bandwidth loop filter is used in the PLL, *both* the lock time *and* its sensitivities tend to be lower (higher), whereas *both* the V_{ctl} swing at lock *and* its sensitivities tend to be higher (lower).

B. Statistical cell library characterization

As we approach 7 nm CMOS, statistical characterization of cell libraries for digital design, taking into account the sensitivities of important performance metrics like speed and power consumption, with respect to device parameters, is crucial [4, 38, 39]. We now use DAGSENS to calculate the sensitivities of one such event-driven metric, namely, the 20% to 80% transition delay of a 22 nm CMOS NAND gate driving an RC load (Fig. 9), with respect to various NMOS, PMOS, and load parameters.

Fig. 10 shows 2 transitions of the NAND gate above (while there are 6 possible transitions that switch the output, we show only 2 to save space, although we analyze the sensitivities of all 6 in Table 3). Fig. 10 also shows the "20% complete" and "80% complete" events in each case, as well as our event-driven gate delay objective function, *i.e.*, the time elapsed between these two events. Table 3 shows the event-driven sensitivities of the NAND gate delay to various NMOS and PMOS parameters (including widths, lengths, threshold voltages, parasitic resistances and capacitances, *etc.*), as well as load parameters. It is interesting to see that, in most (although not all) cases, the gate delay is more sensitive to PMOS (NMOS) parameters during "pull up"

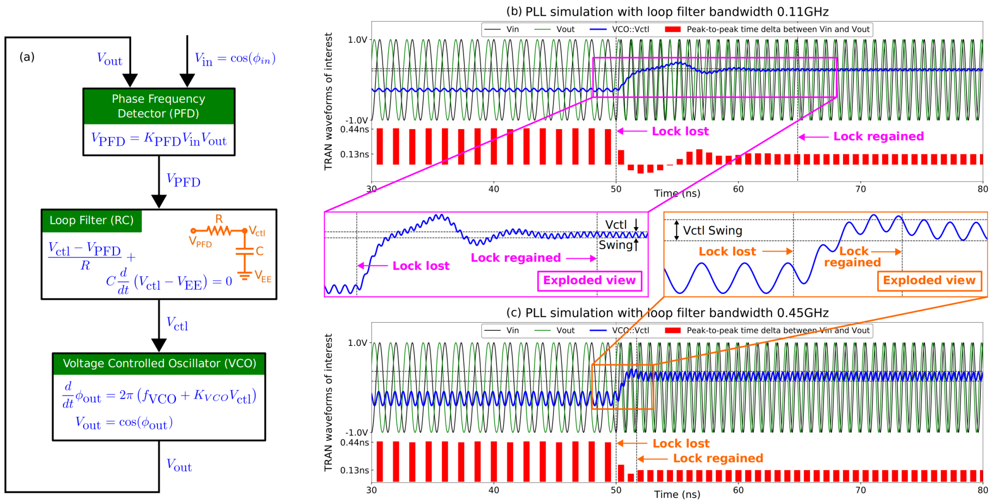


Fig. 8. (a) Block diagram of a PLL, with the underlying equations, (b, c) Transient simulation of low-bandwidth (b) and high-bandwidth (c) PLLs on an input waveform that abruptly changes frequency at $t = 50$ns. The high-bandwidth PLL regains lock more quickly, but features a larger peak-to-peak swing in V_{ctl} around its ideal DC value.

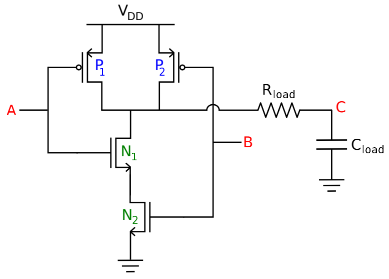


Fig. 9. A CMOS NAND gate driving an RC load.

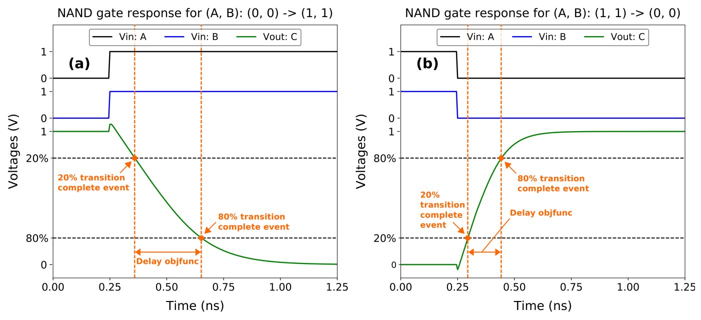


Fig. 10. Transient simulation of the CMOS NAND gate of Fig. 9 for two different input transitions, showing the 20% and 80% "transition complete" events, and the corresponding "gate delay" objective function in each case.

("pull down") transitions, as one would intuitively expect.

C. Biological applications

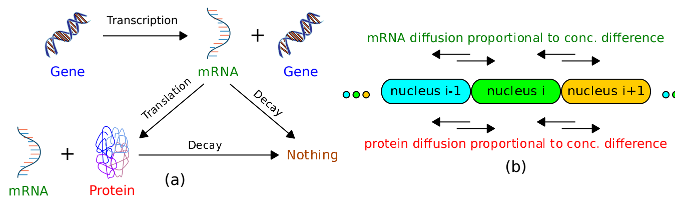


Fig. 11. A model for gene expression in a *Drosophila* embryo, featuring transcription, translation, and decay (part a), as well as diffusion across nuclei (part b).

We now apply DAGSENS to a biological example, *i.e.*, gene expression via transcription, translation, decay, and diffusion in *Drosophila* embryos (Fig. 11) [8, 40]. In this system, a *Drosophila* gene generates mRNA molecules via transcription, which in turn generate protein molecules via translation. In parallel, the mRNA and protein molecules also decay. This is all shown in Fig. 11a [8, 40]. Also, these reactions take place across multiple sites (called *nuclei*), and whenever there is an mRNA or protein imbalance between adjacent nuclei, molecules flow across the border to reduce the imbalance (Fig. 11b) [8, 40]. In our example, we have $N = 52$ nuclei, and each nucleus i (where $1 \leq i \leq N$) has an mRNA concentration $[\text{mRNA}]_i$, and a protein concentration $[\text{protein}]_i$. The system has a single exponentially decaying external input $u(t)$ that governs the rate of transcription. The equations for the system are:

$$\frac{d}{dt}[\text{mRNA}]_i = \underbrace{\sigma_{\text{mRNA}}\ u(t)}_{\text{Transcription}} + \underbrace{d_{\text{mRNA}}([\text{mRNA}]_{i-1} - [\text{mRNA}]_i)}_{\text{Diffusion from previous nucleus}} + \underbrace{d_{\text{mRNA}}([\text{mRNA}]_{i+1} - [\text{mRNA}]_i)}_{\text{Diffusion from next nucleus}} - \underbrace{\lambda_{\text{mRNA}}\ [\text{mRNA}]_i}_{\text{Decay}}, \text{ and} \tag{24}$$

$$\frac{d}{dt}[\text{protein}]_i = \underbrace{\sigma_{\text{protein}}\ [\text{mRNA}]_i}_{\text{Translation}} + \underbrace{d_{\text{protein}}([\text{protein}]_{i-1} - [\text{protein}]_i)}_{\text{Diffusion from previous nucleus}} + \underbrace{d_{\text{protein}}([\text{protein}]_{i+1} - [\text{protein}]_i)}_{\text{Diffusion from next nucleus}} - \underbrace{\lambda_{\text{protein}}\ [\text{protein}]_i}_{\text{Decay}}, \tag{25}$$

with the understanding that the "diffusion from previous (next) nucleus" term is 0 for the first (last) ($i = 1$ (N)) nucleus.

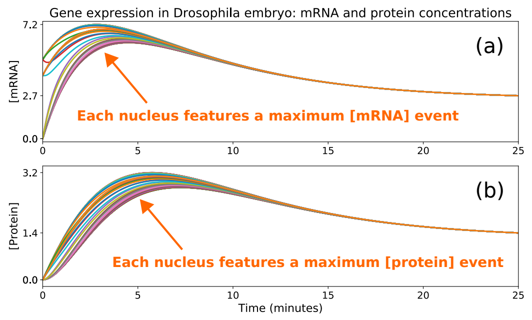


Fig. 12. Transient simulation of gene expression in a *Drosophila* embryo.

Fig. 12 shows a transient run of the system above; at each nucleus i, there is an instant when $[\text{mRNA}]_i$ peaks (before mRNA decay takes its toll), and a (slightly later) instant when $[\text{protein}]_i$ peaks (before protein decay takes its toll). These "peak concentration" events are of interest in many gene expression systems, so we set the times of these

	Parameter	Pull down transitions			Pull up transitions		
Input transition $(A,\ B)$		$(0,\ 0) \rightarrow (1,\ 1)$	$(0,\ 1) \rightarrow (1,\ 1)$	$(1,\ 0) \rightarrow (1,\ 1)$	$(1,\ 1) \rightarrow (1,\ 0)$	$(1,\ 1) \rightarrow (0,\ 1)$	$(1,\ 1) \rightarrow (0,\ 0)$
ϕ (ps)		292.70	292.89	292.85	302.92	293.93	147.38
Sens(ϕ) wrt PMOS parameters	W (nm)	7.87×10^{-6}	3.37×10^{-5}	2.11×10^{-5}	-4.96	-4.77	-2.37
	L (nm)	-2.36×10^{-5}	-1.01×10^{-4}	-6.32×10^{-5}	14.87	14.31	7.12
	V_{th} (V)	8.66×10^{-4}	3.71×10^{-3}	2.32×10^{-3}	-904.66	-867.64	-431.64
	R_d $(k\Omega)$	9.93×10^{-4}	9.78×10^{-4}	9.81×10^{-4}	0.68	0.66	0.31
	R_s $(k\Omega)$	-3.75×10^{-6}	-1.79×10^{-5}	-1.11×10^{-5}	2.88	2.76	1.38
	R_{ds} $(G\Omega)$	-0.15	-0.15	-0.15	0.15	0.14	0.04
	C_{gd} (fF)	572.50	560.58	562.92	625.30	620.19	326.68
	C_{gs} (fF)	3.05×10^{-7}	1.65×10^{-7}	1.73×10^{-7}	5.24×10^{-3}	5.10×10^{-3}	4.69×10^{-3}
	C_{db} (fF)	542.01	544.03	545.59	576.72	573.06	283.58
	C_{sb} (fF)	5.42×10^{-14}	5.72×10^{-14}	5.14×10^{-14}	4.96×10^{-7}	4.94×10^{-7}	5.04×10^{-7}
Sens(ϕ) wrt NMOS parameters	W (nm)	-6.79	-6.80	-6.82	1.32×10^{-3}	2.77×10^{-4}	-2.81×10^{-3}
	L (nm)	13.59	13.61	13.65	-2.65×10^{-3}	-5.54×10^{-4}	5.62×10^{-3}
	V_{th} (V)	813.20	814.42	816.31	-25.84	-0.02	-0.72
	R_d $(k\Omega)$	2.53	2.54	2.54	7.86×10^{-3}	3.31×10^{-4}	5.18×10^{-4}
	R_s $(k\Omega)$	4.51	4.50	4.52	5.73×10^{-4}	-2.04×10^{-4}	-4.64×10^{-5}
	R_{ds} $(G\Omega)$	0.03	0.03	0.03	-0.06	-0.08	-0.02
	C_{gd} (fF)	321.58	311.81	310.31	510.84	333.66	174.65
	C_{gs} (fF)	35.36	32.44	28.97	173.82	2.34×10^{-3}	11.30
	C_{db} (fF)	298.82	295.27	301.73	462.18	286.53	141.53
	C_{sb} (fF)	27.84	23.28	28.97	173.82	6.84×10^{-5}	-0.27
Sens(ϕ) wrt load parameters	R_{load} $(k\Omega)$	0.57	0.57	0.57	0.54	0.59	0.59
	C_{load} (fF)	272.14	273.15	273.93	289.45	287.71	142.98

Table 3. NAND gate delay sensitivities with respect to various NMOS, PMOS, and load parameters, for all input transitions that switch the output.

events, and the corresponding peak concentrations, to be our event-driven objectives.

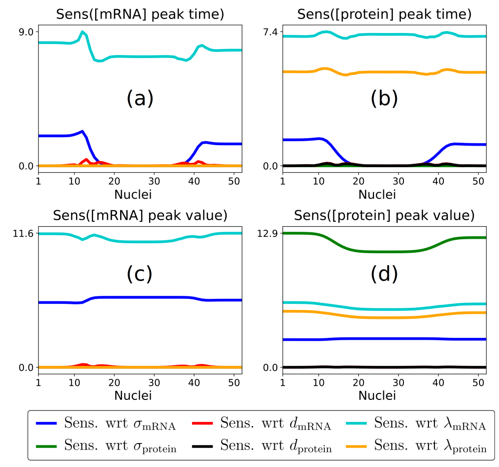


Fig. 13. Sensitivities of peak mRNA and protein concentrations, as well as the times at which these peak concentrations occur, across nuclei, for the *Drosophila* embryo gene expression system.

Fig. 13 shows a plot of these event-driven sensitivities, across nuclei, with respect to various system parameters. It is interesting to see that, while the peak mRNA and protein *event times*, as well as the peak mRNA *concentration value*, are all most sensitive to the mRNA decay constant λ_{mRNA}, the peak protein *concentration value* is most sensitive to the protein translation constant σ_{protein}, for all the nuclei.

IV. Summary

To summarise, we have developed and demonstrated DAGSENS, a simple, elegant, and powerful theory for transient sensitivity analysis based on directed acyclic graphs. We have shown how DAGSENS can be used to carry out direct and adjoint transient sensitivity analysis for an entirely new class of objective functions, defined based on events that happen during transient simulations. We have illustrated this on several real-world applications including high-speed communication (with I/O link and PLL examples), statistical cell library characterization, and gene expression in biological systems.

Acknowledgments: This work was sponsored by the Laboratory Directed Research and Development (LDRD) Program at Sandia National Laboratories. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

References

[1] A. R. Conn, P. K. Coulman, R. A. Haring, G. L. Morrill, C. Visweswariah, and C. W. Wu. JiffyTune: Circuit optimization using time-domain sensitivities. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(12):1292–1309, 1998.
[2] C. Gu and J. Roychowdhury. An efficient, fully non-linear, variability-aware non-Monte-Carlo yield estimation procedure with applications to SRAM cells and ring oscillators. In *ASPDAC '08: Proceedings of the* 13th *Asia and South Pacific Design Automation Conference*, pages 754–761, 2008.
[3] I. Stevanovic and . C. C. McAndrew. Quadratic backward propagation of variance for non-linear statistical circuit modelling. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(9):1428–1432, 2009.
[4] B. Gu, K. Gullapalli, Y. Zhang, and S. Sundareswaran. Faster statistical cell characterization using adjoint sensitivity analysis. In *CICC '08: Proceedings of the* 30th *Annual Custom Integrated Circuits Conference*, pages 229–232, 2008.
[5] T. Turányi. Sensitivity analysis in chemical kinetics. *International Journal of Chemical Kinetics*, 40(11):685–686, 2008.
[6] T. Ziehn and A. S. Tomlin. GUI–HDMR: A software tool for global sensitivity analysis of complex models. *Environmental Modelling & Software*, 24(7):775–785, 2009.
[7] J. M. Dresch, X. Liu, D. N. Arnosti, and A. Ay. Thermodynamic modelling of transcription: Sensitivity analysis differentiates biological mechanism from mathematical model-induced effects. *BMC Systems Biology*, 4(1):142, 2010.
[8] G. D. McCarthy, R. A. Drewell, and J. M. Dresch. Global sensitivity analysis of a dynamic model for gene expression in Drosophila embryos. *PeerJ*, 3:e1022, 2015.
[9] M. Morohashi, A. E. Winn, M. T. Borisuk, H. Bolouri, J. Doyle, and H. Kitano. Robustness as a measure of plausibility in models of biochemical networks. *Journal of Theoretical Biology*, 216(1):19–30, 2002.
[10] T. Eissing, F. Allgöwer, and E. Bullinger. Robustness properties of apoptosis models with respect to parameter variations and intrinsic noise. *Systems Biology*, 152(4):221–228, 2005.
[11] D. E. Hocevar, P. Yang, T. N. Trick, and B. D. Epler. Transient sensitivity computation for MOSFET circuits. *IEEE Transactions on Electron Devices*, 32(10):2165–2176, 1985.
[12] A. Meir and J. Roychowdhury. BLAST: Efficient computation of non-linear delay sensitivities in electronic and biological networks using barycentric Lagrange enabled transient adjoint analysis. In *DAC '12: Proceedings of the* 49th *Annual Design Automation Conference*, pages 301–310, 2012.
[13] Y. Cao, S. Li, L. Petzold, and R. Serban. Adjoint sensitivity analysis for differential-algebraic equations: The adjoint DAE system and its numerical solution. *SIAM Journal on Scientific Computing*, 24(3):1076–1089, 2003.
[14] F. Y. Liu and P. Feldmann. A time-unrolling method to compute sensitivity of dynamic systems. In *DAC '14: Proceedings of the* 51st *Annual Design Automation Conference*, 2014.
[15] R. M. Errico. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78(11):2577–2591, 1997.
[16] R. Bartlett. A derivation of forward and adjoint sensitivities for ODEs and DAEs. Technical Report SAND2007-6699, Sandia National Laboratories, Albuquerque, NM, USA, 2008.
[17] S. Director and R. Rohrer. The generalized adjoint network and network sensitivities. *IEEE Transactions on Circuit Theory*, 16(3):318–323, 1969.
[18] Synopsys. HSPICE® user guide: Simulation and analysis, 2010.
[19] E. R. Keiter, K. V. Aadithya, T. Mei, T. V. Russo, R. L. Schiek, P. E. Sholander, H. K. Thornquist, and J. C. Verley. Xyce® parallel electronic simulator (v6.6): User's guide. Technical Report SAND2016-11716, Sandia National Laboratories, Albuquerque, NM, USA, 2016.
[20] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education, 2006.
[21] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. *Introduction to algorithms*. The MIT Press, 2001.
[22] A. Griewank and A. Walther. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. SIAM, 2008.
[23] C. H. Bischof, P. D. Hovland, and B. Norris. On the implementation of automatic differentiation tools. *Higher-Order and Symbolic Computation*, 21(3):311–331, 2008.
[24] J. Roychowdhury. Numerical simulation and modelling of electronic and biochemical systems. *Foundations and Trends in Electronic Design Automation*, 3(2–3):97–303, 2009.
[25] L. W. Nagel. *SPICE2: A computer program to simulate semiconductor circuits*. PhD thesis, UC Berkeley, 1975.
[26] L. Edelstein-Keshet. *Mathematical models in biology*. SIAM, 2005.
[27] A. L. Sangiovanni-Vincentelli. *Computer Design Aids for VLSI Circuits*, chapter Circuit Simulation, pages 19–112. Springer, Netherlands, 1984.
[28] L. O. Chua and P. M. Lin. Computer-aided analysis of electronic circuits: Algorithms and computational techniques. 1975.
[29] K. V. Aadithya, E. R. Keiter, and T. Mei. DAGSENS: Directed acyclic graph based direct and adjoint transient sensitivity analysis for event-driven objective functions. Technical Report SAND2017-8569, Sandia National Laboratories, Albuquerque, NM, USA, 2017. https://xyce.sandia.gov/publications/_assets/documents/sand2017_8569.pdf.
[30] H. Shichman. Integration system of a non-linear transient network analysis program. *IEEE Transactions on Circuit Theory*, 17(3):378–386, 1970.
[31] C. W. Gear. The numerical integration of ordinary differential equations. *Mathematics of Computation*, 21(98):146–156, 1967.
[32] R. E. Bryant. Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys*, 24(3):293–318, 1992.
[33] G. Balamurugan, B. Casper, J. E. Jaussi, M. Mansuri, F. O'Mahony, and J. Kennedy. Modelling and analysis of high-speed I/O links. *IEEE Transactions on Advanced Packaging*, 32(2):237–247, 2009.
[34] P. K. Hanumolu, G. Y. Wei, and U. K. Moon. Equalizers for high-speed serial links. *International Journal of High Speed Electronics and Systems*, 15(2):429–458, 2005.
[35] J. A. Davis and J. D. Meindl. *Interconnect technology and design for gigascale integration*. Springer, Netherlands, 2003.
[36] B. Razavi. *Design of analog CMOS integrated circuits*. Tata McGraw-Hill Publishing Company Ltd., New Delhi, India, 2001.
[37] J. L. Stensby. *Phase-locked loops: Theory and applications*. CRC Press, 1997.
[38] A. Goel and S. Vrudhula. Statistical waveform and current source based standard cell models for accurate timing analysis. In *DAC '08: Proceedings of the* 45th *Annual Design Automation Conference*, pages 227–230, 2008.
[39] L. Yu, S. Saxena, C. Hess, I. M. Elfadel, D. Antoniadis, and D. Boning. Statistical library characterization using belief propagation across multiple technology nodes. In *DATE '15: Proceedings of the* 18th *Design, Automation & Test Conference in Europe*, pages 1383–1388, 2015.
[40] J. M. Dresch, M. A. Thompson, D. N. Arnosti, and C. Chiu. Two-layer mathematical modelling of gene expression: Incorporating DNA-level information and system dynamics. *SIAM Journal on Applied Mathematics*, 73(2):804–826, 2013.