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

SAND2017-4319C

***Abstract*—We present DAGSENS, a new theory for transient sensitivity analysis of Differential Algebraic Equation systems (DAEs), such as SPICE-level circuit equations. The key ideas behind DAGSENS are, (1) to represent the entire sequence of computations, starting from DAE parameters, all the way up to the objective function whose sensitivity we need to compute, as a Directed Acyclic Graph (DAG) called the "sensitivity DAG", and (2) to compute the required sensitivites efficiently (with time complexity linear in the size of the sensitivity DAG) by leveraging dynamic programming techniques to traverse the DAG. DAGSENS is simple, elegant, and easy-to-understand compared to existing sensitivity analysis approaches: for example, in DAGSENS, one can switch between direct and adjoint transient sensitivities just by changing the direction of DAG traversal (*i.e.*, topological order *vs.* reverse topological order). Also, DAGSENS is significantly more powerful than existing sensitivity analysis approaches because it allows one to compute the sensitivities of a much more general class of objective functions, including those defined based on "events" that occur during a transient simulation (*e.g.*, a node voltage crossing a particular threshold, a phase-locked loop (PLL) achieving lock, a signal reaching its maximum/minimum value during the simulation, *etc.*). In this paper, we apply DAGSENS to compute the sensitivities of important event-driven performance metrics in several real-world electronic and biological applications, including high-speed communication (featuring sub-systems such as I/O links and PLLs), standard cell characterization, and gene expression in *Drosophila* embryos.**

I. Introduction

This paper is about a new flexible and elegant approach to computing transient parameter sensitivities. In recent years there has been a lot of interest in transient sensitivity analysis for large-scale dynamical systems, represented by linear/nonlinear differential-algebraic equations (DAEs) or partial differential equations (PDEs). Within engineering and scientific communities, it is well understood that performance sensitivities with respect to model parameters or system input excitations are valuable for a variety of purposes, including design optimization [1], uncertainty quantification [2], stability analysis [3] and transient global error control [4].

In the field of integrated circuit design, transient sensitivities have historically been very important [5], [6]. Specific circuit design applications of transient adjoint parametric sensitivities have included automated circuit tuning [7] and yield estimation [8], among others. The importance of sensitivites in circuit design is likely to increase, particularly with the move to progressively smaller process nodes. As the industry approaches the 14nm and 7nm technology nodes, parameter variability is becoming ever more pronounced, due to less control over the manufacturing process.

In general, there are two types of sensitivity algorithms: direct sensitivities and adjoint sensitivities. Direct methods [6] are relatively easy to understand and to implement, but scale poorly for problems with large numbers of parameters. Adjoint methods [3], [9]–[13] are generally harder to understand and implement, and have thus received much more recent attention in the literature. Unlike direct methods, they scale very well to very large numbers of parameters. Modern simulation problems can frequently consist of millions, or even billions of degrees of freedom, and a similarly large number of input parameters of interest. For such problems, computing sensitivities can only be practically accomplished using an adjoint method.

Despite the long history and continued importance, transient adjoint methods are often not well understood. Previous descriptions have lacked generality, and it has been unclear how to apply adjoint methods efficiently to many common objective functions of interest. Many papers, particularly early ones [5], [7] have been based around the construction of an "adjoint circuit". Most recent papers are based around a more mathematical description, but have restricted themselves to the sensitivity of solution variables at specific points in time [9], [11], or integrals with respect to time of the solution [10], [13]. Throughout this paper, we describe such objective functions as "final point" and "integral" objective functions, respectively.

In practice, such objective functions are often inadequate, as objectives of interest often don't directly correspond to either a "final point" or an "integral" style of function. Circuit designers frequently make use of metrics we describe as "event-driven" objectives. Such objectives are defined as the point in time at which a particular event happens. Examples include when a signal exceeds a user-specified value, or possibly reaches a maximum value. For a more complicated example, the objective function could be the amount of time it takes for a PLL to lock to a new input frequency. The "achievement of lock" is a transient event, and so the "time to lock" is an event-driven objective function. If the PLL parameters change, so does the time to lock (as well as the state of the PLL when lock is finally achieved). These types of objectives are common in circuit analysis, and are exemplified by the `.MEASURE` feature that is found in popular circuit simulators such as HSPICE [14] and Xyce [15].

In this paper, we develop DAGSENS, a new theory for transient sensitivity analysis based on directed acyclic graphs (DAGs), related to some of the ideas found in the automatic differentiation literature [16], [17]. In this approach, each computation and intermediate quantity is represented in a DAG, starting from the parameters all the way to the objective function. Once such a DAG is constructed, required sensitivities can be obtained by traversing it. A key advantage is the simplicity and elegance of the DAG approach: switching from a direct approach to an adjoint approach is as simple as switching the direction of DAG traversal from forward topological order to reverse topological order We also define and develop the notion of event-driven transient sensitivity analysis, where the objective function is defined based on events that happen during a transient run. We show how the DAG formulation can enable straightforward extension for computing direct and adjoint sensitivities of event-driven objective functions.

The remainder of this paper is organized as follows. In sections §II, general background related to the solution of DAEs and sensitivity analysis are presented. In sections §II-D a description of the DAG theory of sensitivity analysis is described. In section §II-I the extension of this method to event-based objective functions is discribed. Results for several representative circuit problems and also a biological example are given in section §III.

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

We now discuss the key ideas and core concepts behind DAGSENS.

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 wish to compute sensitivities (*e.g.*, transistor widths and lengths, parasitic resistances and capacitances), $\vec{u}$ 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 [18]–[20].

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 $\{t_0,\ t_1,\ \ldots,\ t_{N-1}\}$ (where $t_{N-1} = t_f$), and then approximating the respective DAE states $\{\vec{x}_0,\ \vec{x}_1,\ \ldots,\ \vec{x}_{N-1}\}$ by solving a sequence of "Linear Multi-Step" (LMS) equations of the form [18], [19], [21], [22]:

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

Method	m_i	Coefficient	$j=2$	$j=1$	$j=0$
FE	1	$\alpha_i(-j)$	–	$-\frac{1}{t_i-t_{i-1}}$	$\frac{1}{t_i-t_{i-1}}$
		$\beta_i(-j)$	–	1	0
BE	1	$\alpha_i(-j)$	–	$-\frac{1}{t_i-t_{i-1}}$	$\frac{1}{t_i-t_{i-1}}$
		$\beta_i(-j)$	–	0	1
TRAP	1	$\alpha_i(-j)$	–	$-\frac{1}{t_i-t_{i-1}}$	$\frac{1}{t_i-t_{i-1}}$
		$\beta_i(-j)$	–	0.5	0.5
GEAR2	2	$\alpha_i(-j)$	$\frac{1}{t_{i-1}-t_{i-2}}$ $-\frac{1}{t_i-t_{i-2}}$	$-\frac{1}{t_i-t_{i-1}}$ $-\frac{1}{t_{i-1}-t_{i-2}}$	$\frac{1}{t_i-t_{i-1}}$ $+\frac{1}{t_i-t_{i-2}}$
		$\beta_i(-j)$	0	0	1

Table 1. Coefficients used by several common LMS methods in Eq. (2).

Thus, at each step i (where $1 \leq i \leq N-1$), 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). Table 1 lists the α and β coefficients used by several common LMS methods, including Forward Euler (FE), Backward Euler (BE), Trapezoidal (TRAP), and second-order Gear (GEAR2), in the construction of Eq. (2) [18], [21], [23], [24].

C. Transient sensitivity analysis of DAEs

Suppose we have a DAE in the form of Eq. (1), with parameters $\vec{p}^\star$, and transient solution $\vec{x}^\star(t)$ over the interval $[t_0, t_f]$. 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 $\vec{p}^\star + \Delta\vec{p}$. As a result, let us say that 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, which is obtained by doing a perturbation analysis of Eq. (1), is given by [9]–[11]:

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

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

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

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

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 the solution $\vec{x}(t)$ with respect to the parameters $\vec{p}$, 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 an expression for $S_x(t)$. Rather, it needs to be *solved* for $S_x(t)$. The "direct" approach for this involves two rounds of transient analysis. In the first round, the original DAE (Eq. 1) is solved using the LMS techniques of §II-B. This yields a sequence of discretized time-points $\{t_i\}$ (where $0 \leq i \leq N-1$ and $t_{N-1} = t_f$), as well as a corresponding sequence of DAE states $\{\vec{x}^\star(i)\}$ (§II-B). These, in turn, are used to compute the sequences of Jacobians $\{J^\star_{qx}(i)\}$, $\{J^\star_{qp}(i)\}$, $\{J^\star_{fx}(i)\}$, and $\{J^\star_{fp}(i)\}$.

In the second round, the sensitivity DAE (Eq. 4) is solved using the same LMS techniques, and the same sequence of LMS methods (FE, BE, *etc.*) used for the first round. The LMS equations that are solved (similar to Eq. 2) are:

$$\sum_{j=0}^{m_i}\Big[\alpha_i(-j)\;\left[J^\star_{qx}(i-j)S_x(i-j) + J^\star_{qp}(i-j)\right] + \beta_i(-j)\;\left[J^\star_{fx}(i-j)S_x(i-j) + J^\star_{fp}(i-j)\right]\Big] = \mathbf{0}_{|\vec{x}^\star|\times|\vec{p}^\star|}, \tag{6}$$

where the α and β coefficients, as before, are looked up from Table 1.

Solving the LMS equations above yields the required sensitivities $S_x(i)$ for $0 \leq i \leq N$, discretized over the time-interval $[t_0, t_f]$. Note that the initial condition $S_x(0)$ (which is the sensitivity of the initial DAE state $\vec{x}^\star(0)$) needs to be specified to start the chain of LMS solves above; in practice, this is usually found by DC sensitivity analysis [10], [18].

D. The sensitivity DAG

Each step of the transient analysis of §II-B builds on previously computed DAE state vectors, to solve for a new DAE state vector. This sequence of computations fits naturally into a Directed Acyclic Graph (DAG) structure, as shown in Fig. 1.

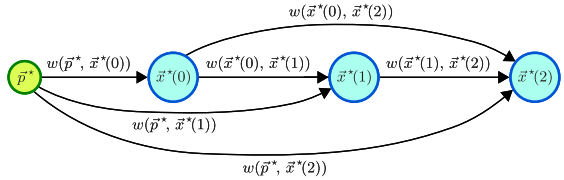


Fig. 1. The DAG structure underlying a transient simulation.

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 [18], [19]); so, there is an edge that leads 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 (Table 1). So, there are edges leading from both the $\vec{p}^\star$ node and the $\vec{x}^\star(0)$ node to the $\vec{x}^\star(1)$ node. Finally, $\vec{x}^\star(2)$ is assumed to be computed via an LMS method like GEAR2 that has memory $m_2 = 2$ (Table 1). 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 leading from all these three nodes to the $\vec{x}^\star(2)$ node.

While Fig. 1 stops at $\vec{x}^\star(2)$ for lack of space, it is easy to see that the ideas behind the DAG construction can be extended to the entire length of the transient simulation. In general, if the transient simulation has N points with indices $0 \leq i \leq N-1$, 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, which will originate at the $\vec{p}^\star$ node. 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 the $\vec{p}^\star$ node, 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 $1 \leq j \leq m_i$.

Also, each edge of the sensitivity DAG has a *weight*, as shown in 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) [18], while all other weights are obtained by doing a perturbation analysis of Eq. (2):

$$w(\vec{p}^\star, \vec{x}^\star(0)) = \frac{\partial\vec{x}^\star(0)}{\partial\vec{p}^\star} = -J^\star_{fx}(0)^{-1}J^\star_{fp}(0), \tag{7}$$

$$w(\vec{p}^\star, \vec{x}^\star(i)) = \frac{\partial\vec{x}^\star(i)}{\partial\vec{p}^\star} = -\left[\alpha_i(0)J^\star_{qx}(i) + \beta_i(0)J^\star_{fx}(i)\right]^{-1} \left[\sum_{j=0}^{m_i}\left(\alpha_i(-j)J^\star_{qp}(i-j) + \beta_i(-j)J^\star_{fp}(i-j)\right)\right], \quad \forall\; 1 \leq i \leq N-1, \text{ and} \tag{8}$$

$$w(\vec{x}^\star(i-j), \vec{x}^\star(i)) = \frac{\partial\vec{x}^\star(i)}{\partial\vec{x}^\star(i-j)} = -\left[\alpha_i(0)J^\star_{qx}(i) + \beta_i(0)J^\star_{fx}(i)\right]^{-1} \left[\alpha_i(-j)J^\star_{qx}(i-j) + \beta_i(-j)J^\star_{fx}(i-j)\right], \quad \forall\; 1 \leq i \leq N-1,\; 1 \leq j \leq m_i. \tag{9}$$

E. Objective functions and the sensitivity DAG

As mentioned in §I, in many applications, we are *not* directly interested in the sensitivities of the solution $\vec{x}^\star(t)$, but would like to compute the sensitivities of important transient performance metrics (*i.e.*, "objective functions") *derived* from $\vec{x}^\star(t)$ (and denoted $\vec{\phi}^\star$ in this paper). Below, we discuss two kinds of objective functions commonly found in the sensitivity analysis literature ("final point" and "integral" objective functions), and show how to incorporate these into the sensitivity DAG. Later (§II-H, §II-I, §II-J), we will do the same for "event-driven" objective functions, a powerful new class of objective functions *not* found in the existing literature.

Final point objective functions. These take the form:

$$\vec{\phi}^\star = \vec{h}(\vec{x}^\star(N-1),\, \vec{p}^\star), \tag{10}$$

where $\vec{x}^\star(N-1)$ is the final point in the transient simulation.

Therefore, the *sensitivity* S_ϕ of such an objective function, evaluated at $\vec{p}^\star$, is given by:

$$S_\phi = J^\star_{hx}(N-1)S_x(N-1) + J^\star_{hp}(N-1), \tag{11}$$

where the Jacobian symbols have their usual meanings.

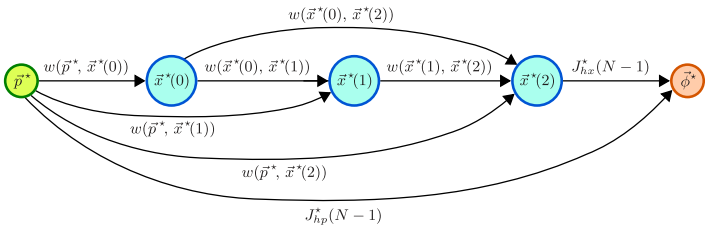


Fig. 2. Adding a final point objective function to the sensitivity DAG of Fig. 1.

To include such an objective function in the sensitivity DAG, we add a new node $\vec{\phi}^\star$ to the DAG, 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 $J^\star_{hx}(N-1)$. This is illustrated in Fig. 2 for the simple case $N = 3$.

Integral objective functions. These take the form:

$$\vec{\phi}^\star = \int_{t=t_0}^{t_f} \vec{h}(t,\, \vec{x}^\star(t),\, \vec{p}^\star)\; dt. \tag{12}$$

Therefore, we have:

$$S_\phi = \int_{t=t_0}^{t_f} \left(J^\star_{hx}(t)S_x(t) + J^\star_{hp}(t)\right)\; dt. \tag{13}$$

In practice, the integral in Eq. (13) is approximated by a summation:

$$S_\phi \approx \sum_{i=0}^{N-2} \left[\left(J^\star_{hx}(i)S_x(i) + J^\star_{hp}(i)\right)(t_{i+1} - t_i)\right]. \tag{14}$$

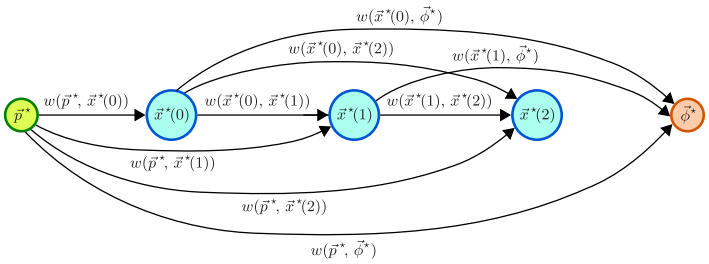


Fig. 3. Adding an integral objective function to the sensitivity DAG of Fig. 1.

To include such an objective function in the sensitivity DAG, we add a new node $\vec{\phi}^\star$ to the DAG, 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). The weights of these edges are:

$$w(\vec{p}^\star,\, \vec{\phi}^\star) = \sum_{i=0}^{N-2} \left(J^\star_{hp}(i)(t_{i+1} - t_i)\right), \text{ and} \tag{15}$$

$$w(\vec{x}^\star(i),\, \vec{\phi}^\star) = J^\star_{hx}(i)(t_{i+1} - t_i),\; \forall\; 0 \le i \le N-2. \tag{16}$$

This is illustrated in Fig. 3 for the simple case $N = 3$.

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

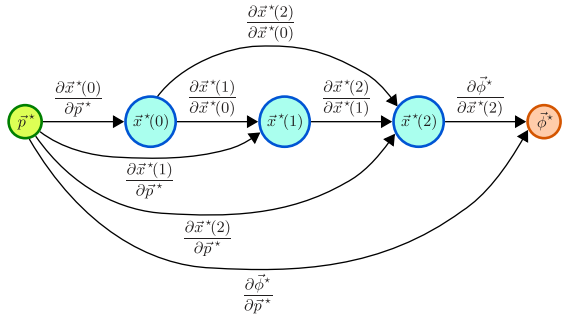


Fig. 4. The DAG of Fig. 2, with edge-weights denoted by partial derivatives.

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. Indeed, this is the same DAG derived in Fig. 2, except that the notation for the edge-weights has been changed to make it clear that the edge-weights are, in fact, partial derivatives. Now, let us repeatedly apply the chain rule to find the sensitivity of the objective function:

$$\begin{aligned}
\text{Sensitivity we need} &= \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}$$

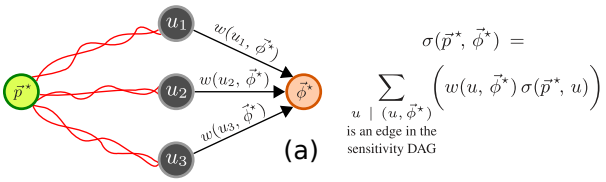
The derivation above shows that the sensitivity we want to compute is a sum of terms, where each term corresponds to a unique path from $\vec{p}^\star$ to $\vec{\phi}^\star$ in the sensitivity DAG; more precisely, each term is a "product of edge-weights in reverse" of some path from $\vec{p}^\star$ to $\vec{\phi}^\star$. Thus, we have a key insight: *solving the sensitivity analysis problem is the same as enumerating paths in the sensitivity DAG*. This holds true for all sensitivity analysis problems, but we omit a rigorous proof due to space constraints.

Taking a cue from this, let us 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

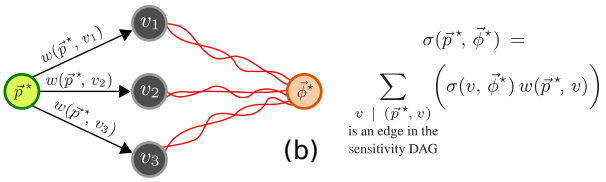
sum of the weights of all the paths in the DAG that start at u and end at v. Thus, solving the sensitivity analysis problem is the same as computing $\sigma(\vec{p}^\star, \vec{\phi}^\star)$ in the sensitivity DAG.

G. Direct and Adjoint approaches to DAG path enumeration

We just reduced sensitivity analysis to the problem of enumerating all paths in the sensitivity DAG between $\vec{p}^\star$ and $\vec{\phi}^\star$, and adding up their weights. The brute-force approach to this, however, is computationally infeasible because the number of such paths grows exponentially as the size of the DAG [25], [26]. Therefore, we leverage dynamic programming techniques to efficiently enumerate DAG paths, and thus solve the sensitivity analysis problem in linear time in the size of the sensitivity DAG [25], [26].



Direct sensitivity analysis: Optimal sub-structure for dynamic programming



Adjoint sensitivity analysis: Optimal sub-structure for dynamic programming

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

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.

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 $\sigma(\vec{p}^\star, \vec{p}^\star) = \mathbf{I}_{|\vec{p}^\star| \times |\vec{p}^\star|}$
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 $\sigma(\vec{p}^\star, u) \mathrel{+}= w(v, u)\ \sigma(\vec{p}^\star, v)$
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 [25], [26]. 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 [3], [9]–[13]. Indeed, Algorithm 2's reverse order traversal is equivalent to the idea of "integrating the sensitivity DAE backwards in time" – the interpretation provided for adjoints in many sensitivity analysis papers [9], [10], [13]. Also, there are strong parallels between direct/adjoint DAG path enumeration in DAGSENS and forward/reverse mode automatic differentiation [17].

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 $\sigma(\vec{\phi}^\star, \vec{\phi}^\star) = \mathbf{I}_{|\vec{\phi}^\star| \times |\vec{\phi}^\star|}$
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 $\sigma(v, \vec{\phi}^\star) \mathrel{+}= \sigma(u, \vec{\phi}^\star)\ w(v, u)$
7 **return** $\sigma(\vec{p}^\star, \vec{\phi}^\star)$

Thus, just by changing the order in which the DAG is traversed, DAGSENS allows one to switch between direct and adjoint sensitivity analysis. We believe that this is a level of simplicity and elegance that is lacking in existing sensitivity analysis approaches. Moreover, DAGSENS works seamlessly for all LMS methods and easily allows the use of different LMS methods for different transient time-steps – all without the need for re-deriving the adjoint equations, specialized matrix structures, and/or complicated backwards-in-time integration formulas for each different LMS method. Also, we believe that due to its ease of understandability, DAGSENS can make the benefits of sensitivity analysis accessible to a a much wider audience, including device engineers, circuit designers, and students.

Finally, we would like to point out that 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. 7, 8, and 9), this is a computation of the form $AB^{-1}C$, which can be done either as $A(B^{-1}C)$, or as $(C^T((B^T)^{-1}A^T))^T$ (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

We now introduce a powerful new class of objective functions that have not been addressed in the existing sensitivity analysis literature. The key idea is that 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, a species in a chemical reaction network reaching its maximum/minimum concentration), and then compute the sensitivities of objective functions that depend on these events.

For our purposes, an "event" *ev* is specified by the condition:

$$g_{ev}(\tau_{ev}^\star, \vec{x}^\star(\tau_{ev}^\star), \vec{p}^\star) = 0, \tag{17}$$

where $\tau_{ev}^\star$ 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 $\tau_{ev}^\star$, or a specification such as "the third time Eq. (17) is satisfied", *etc.*). But for sensitivity analysis, we can ignore these additional specifications because we only do perturbation analysis of Eq. (17) in a small neighbourbood around $\tau_{ev}^\star$.

Note that events corresponding to a signal reaching its maximum/minimum value are specified by adding a new DAE state variable that represents the derivative of the signal in question, and by equating this variable to zero via Eq. (17) (see §III for examples).

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

$$\vec{\phi}^\star = \vec{h}(\tau_1^\star,\, \tau_2^\star,\, \ldots,\, \tau_M^\star,\, \vec{x}^\star(\tau_1^\star),\, \vec{x}^\star(\tau_2^\star),\, \ldots,\, \vec{x}^\star(\tau_M^\star),\, \vec{p}^\star). \tag{18}$$

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. We would like to compute the sensitivity of event-driven objective functions by taking into account both these sets of changes.

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 $S_{x_{ev}} \neq S_x(\tau_{ev}^\star)$; while $S_x(\tau_{ev}^\star)$ 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. (17) yields:

$$S_{\tau_{ev}} = -\frac{J^\star_{g_{ev}x} S_x(\tau_{ev}^\star) + J^\star_{g_{ev}p}}{J^\star_{g_{ev}x}\dot{\vec{x}}^\star(\tau_{ev}^\star) + J^\star_{g_{ev}\tau}},\ \text{and} \tag{19}$$

$$S_{x_{ev}} = S_x(\tau_{ev}^\star) + \dot{\vec{x}}^\star(\tau_{ev}^\star) S_{\tau_{ev}},\ \forall\ 1 \leq ev \leq M,\ \text{where} \tag{20}$$

$$\dot{\vec{x}}^\star(\tau_{ev}^\star) = \left.\frac{d}{dt}\vec{x}^\star(t)\right|_{\tau_{ev}^\star}, \tag{21}$$

and where Jacobian terms have their usual meanings and are all evaluated at $(\tau_{ev}^\star,\, \vec{x}^\star(\tau_{ev}^\star),\, \vec{p}^\star)$. Therefore, we first need to *solve for the event*, *i.e.*, find $\tau_{ev}^\star$ and $\vec{x}^\star(\tau_{ev}^\star)$, 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 use a modified LMS strategy to solve for the event, by solving:

$$\begin{aligned}
&\Big(\alpha_{ev}(0, \tau_{ev}^\star)\ \vec{q}(\vec{x}^\star(\tau_{ev}^\star),\, \vec{p}^\star) + \beta_{ev}(0, \tau_{ev}^\star)\ \vec{f}(\vec{x}^\star(\tau_{ev}^\star),\, \vec{u}(\tau_{ev}^\star),\, \vec{p}^\star)\Big) \\
&+ \sum_{j=1}^{m_{ev}} \Big(\alpha_{ev}(-j, \tau_{ev}^\star)\ \vec{q}(\vec{x}^\star(i-j+1),\, \vec{p}^\star) \\
&\quad + \beta_{ev}(-j, \tau_{ev}^\star)\ \vec{f}(\vec{x}^\star(i-j+1),\, \vec{u}(t_{i-j+1}),\, \vec{p}^\star)\Big) = \vec{0},\ \text{and}
\end{aligned} \tag{22}$$

$$g_{ev}(\tau_{ev}^\star,\, \vec{x}^\star(\tau_{ev}^\star),\, \vec{p}^\star) = 0. \tag{23}$$

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. (19) and (20).

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

$$S_\phi = J^\star_{hp} + \sum_{ev=1}^{M} \left(J^\star_{h\tau_{ev}} S_{\tau_{ev}} + J^\star_{hx(\tau_{ev})} S_{x_{ev}}\right). \tag{24}$$

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

Fig. 6 shows the steps involved in adding an event-driven objective function to the sensitivity DAG. For each event $1 \leq ev \leq M$, we add three new nodes to the DAG (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. (19) and (20) respectively. The edges associated with these nodes, and their weights, are shown in Fig. 6a.

Finally, a single new node $\vec{\phi}^\star$ is added to the DAG 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 added above. The weights of these edges, as shown in the figure, follow Eq. (24).

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.

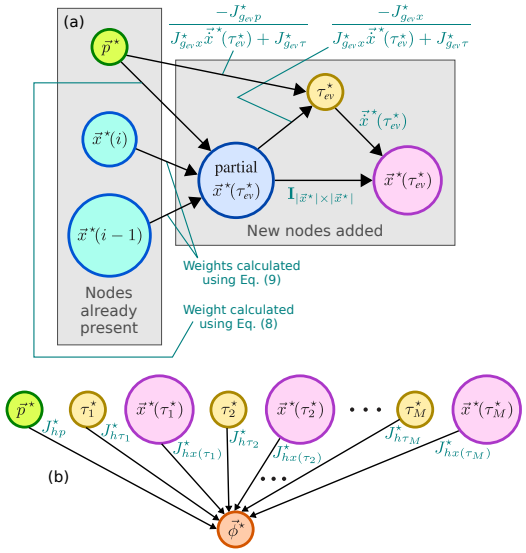


Fig. 6. (a) Adding events, and (b) adding an event-driven objective function, to the sensitivity DAG.

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 $1 \leq ev \leq M$ in the form of Eq. (17), and an event-driven objective function ϕ in the form of Eq. (18)

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 $J^\star_{qx}(t)$, $J^\star_{qp}(t)$, $J^\star_{fx}(t)$, and $J^\star_{fp}(t)$ from the transient simulation.

3 Build a sensitivity DAG G, using information from the transient run and the Jacobians above, via Eqs. (7), (8), and (9).

4 **for** $1 \leq ev \leq M$ **do**

5 Solve for event *ev*, *i.e.*, find $\tau_{ev}^\star$ and $\vec{x}^\star(\tau_{ev}^\star)$, by constructing and solving Eqs. (22) and (23).

6 Augment the sensitivity DAG with nodes corresponding to *ev*, as outlined in §II-J.

7 Augment the sensitivity DAG with a ϕ 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 apply 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. High-speed communications sub-systems

1) A "maximum crosstalk" example: In modern high-speed I/O links, "crosstalk" between parallel channels (*e.g.*, those found in a CPU/DRAM interface) often adversely impacts bandwidth [27]–[29]. When two signal-carrying lines are physically close to one another 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 [27]–[29].

Fig. 7 (a) 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 consists of two sub-circuits: the one on the right where crosstalk is modelled via cross-coupled capacitances, and

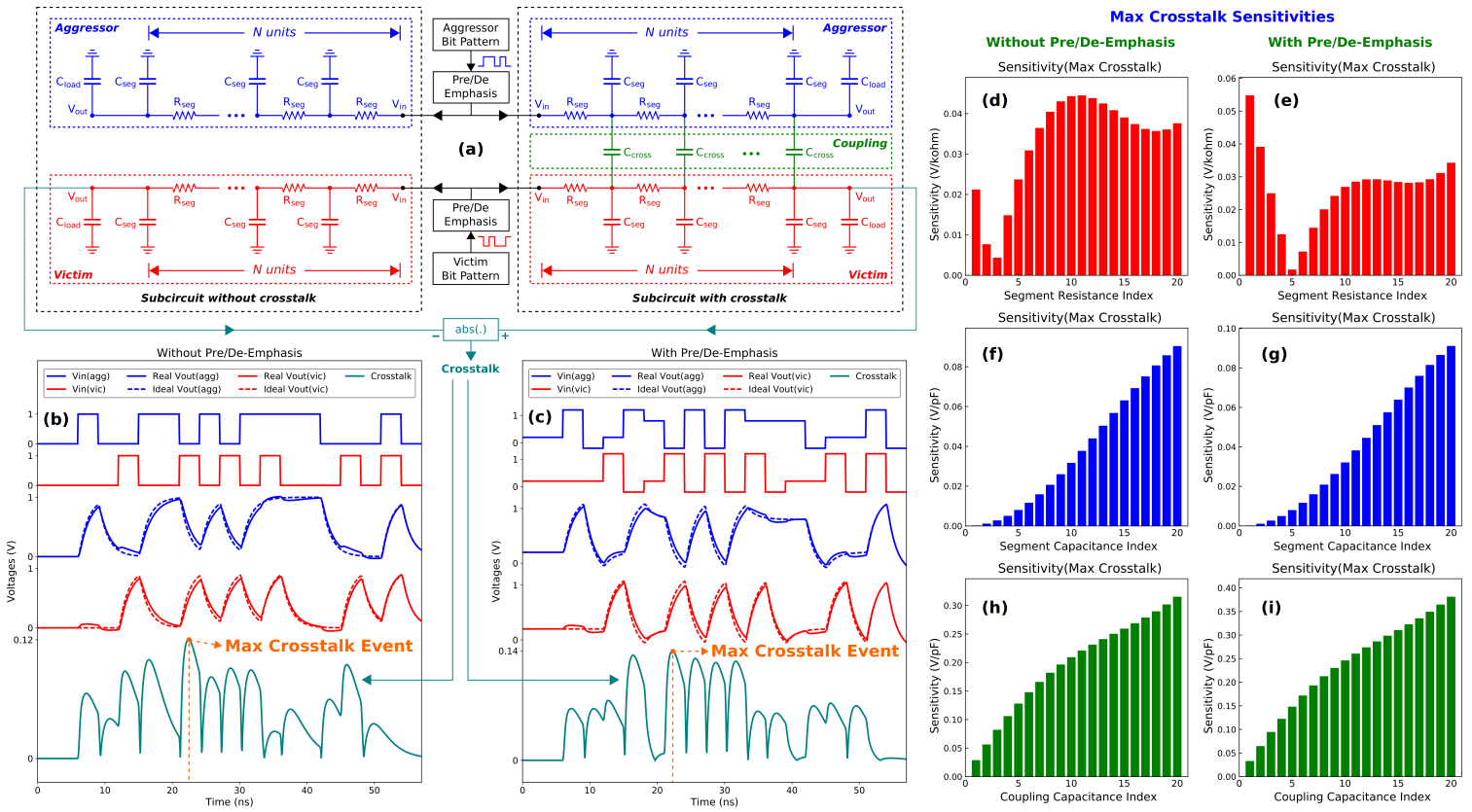


Fig. 7. (a) The circuit used to determine the magnitude of crosstalk induced by the aggressor line on the victim line in our I/O link. (b, c) Transient simulation of the circuit in (a) without and with pre/de-emphasis respectively, with the event corresponding to maximum crosstalk in each case. (d through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

	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 2. The impact of using pre/de-emphasis on the sensitivities of maximum crosstalk (ϕ), with respect to total segment resistance, total segment capacitance, as well as load parameters.

N	t_{dir}	t_{adj}	Adj. speedup
1	2.50 s	2.09 s	1.19
5	5.39 s	4.21 s	1.28
10	9.03 s	6.83 s	1.32
20	16.47 s	12.13 s	1.36
50	38.98 s	27.74 s	1.41
100	1.32 mins	53.92 s	1.47
200	2.92 mins	1.77 mins	1.66
500	10.37 mins	4.41 mins	2.35
1000	1.33 hours	9.03 mins	8.81
2000	6.06 hours	18.27 mins	19.90
5000	Out of memory after $>$ 27 hours	46.11 mins	$>$ 35
10000	Did not try	1.55 hours	N/A

Table 3. Adjoint sensitivity analysis carries powerful advantages over direct sensitivity analysis when the dimension of the objective function is much smaller than that of the DAE parameter space.

the one on the left 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 the course of the transient simulation, where the aggressor and victim lines 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 end, it can have the drawback of increasing crosstalk [27]–[29].

Parts (d) to (i) of Fig. 7 show the results of applying DAGSENS to this system above, where the sensitivities of the maximum crosstalk with respect to each segment resistance, segment capacitance, and segment coupling capacitance are plotted as bar charts. In particular, it is interesting to see (parts d, e) that the maximum crosstalk is much more sensitive to the first few segment resistances when pre/de-emphasis is employed. Also, it is interesting to see that the sensitivities with respect to the 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 2 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 various insights that would not be possible with existing sensitivity analysis tools.

Since our objective function has dimension 1, as opposed to the DAE parameter space that has a dimension $O(3N)$, where N is the number of RC segments, this is also a good test case to illustrate the benefits of adjoint over direct sensitivity analysis. Table 3 illustrates this by showing the speedups achieved by adjoint DAGSENS over direct DAGSENS for various N: as N increases, these speedups become more impressive. We note that, at present, DAGSENS is a proof-of-concept code written in Python rather than production code written in a language like C or C++. In particular, efficient garbage collection and memory management techniques have not been implemented in DAGSENS yet, which is why the program can run out of memory relatively easily. We plan to address these issues in the future (§IV), but we believe that the benefits of adjoint analysis over direct analysis are still clear from Table 3.

2) A PLL example: PLLs are widely used in high-speed communication sub-systems for frequency synthesis, clock and data recovery (CDR), *etc.* [27], [30], [31]. The lock time of a PLL, *i.e.*, how quickly the PLL can lock to a new input frequency, is of critical importance in these applications. Since a PLL achieving lock is a transient event, we can use DAGSENS to calculate the sensivities of a PLL's lock time with respect to its parameters.

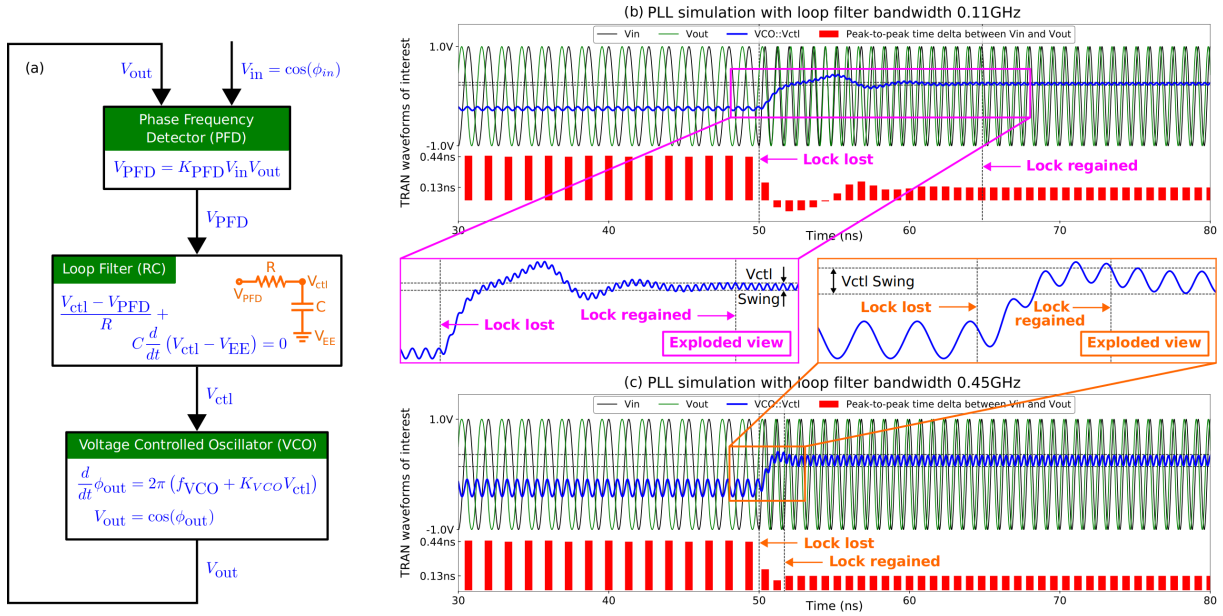


Fig. 8. (a) Block diagram of a PLL, with the underlying equations, (b, c) Transient simulation of low-bandwidth (b) and high-bandwidth (c) PLLs on an input waveform that abruptly changes frequency at $t = 50$ns. The high-bandwidth PLL regains lock more quickly, but features a larger peak-to-peak swing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

		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})	−1.59	45.76	−0.07	188.67
	R ($k\Omega$)	1.21	−32.61	0.13	−259.66
	C (pF)	1.69	−45.65	0.18	−363.53
	K_{VCO} (V^{-1}GHz)	−5.29	0.35	−0.39	5.72
	f_{VCO} (GHz)	−18.48	−0.07	−1.09	0.93

Table 4. Sensitivities of PLL lock times and peak-to-peak V_{ctl} swings when locked, with respect to various macromodel parameters, for both low and high bandwidth loop filters.

Fig. 8 (a) shows a high-level block-diagram for a PLL, and also the equations and parameters associated with each PLL component [30], [31]. Parts (b) and (c) of Fig. 8 show transient simulations of two different PLLs, one (b) with a low-bandwidth loop filter and the other (c) with a high-bandwidth loop filter. In each case, the input waveform abruptly switches its frequency at $t = 50$ns, throwing the PLLs off-lock. The PLLs then 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. The peak-to-peak swing in V_{ctl} is also an objective function of interest; if one used an ideal loop filter, V_{ctl} would settle to a DC value, so this 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 metrics: 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 4 shows the sensitivities of both the event-driven objective functions above (PLL lock times as well as V_{ctl} swings at lock), with respect to the PLL macromodel parameters shown in Fig. 8 (a). 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 the age of 7 nm CMOS and near/sub-threshold computing, statistical characterization of cell libraries for digital design, taking into account the sensitivities of important performance metrics like timing and power consumption, with respect to parameter variability, is crucial [12], [32], [33].

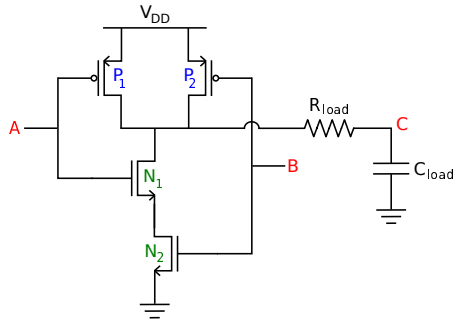


Fig. 9. A CMOS NAND gate driving an RC load.

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), to various NMOS, PMOS, and load parameters.

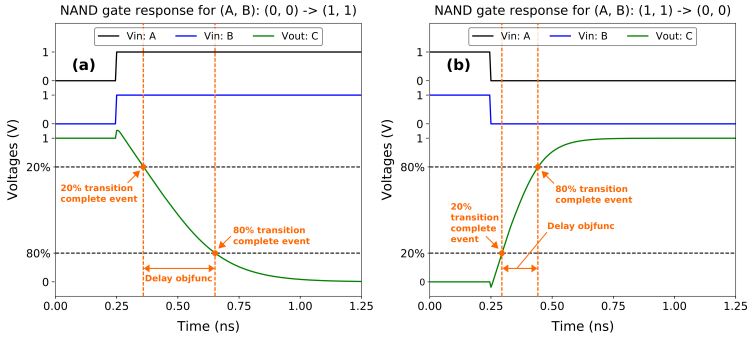


Fig. 10. Transient simulation of the CMOS NAND gate of Fig. 9 for two different input transitions, showing the 20% and 80% "transition complete" events, and the corresponding "delay" objective function in each case.

Fig. 10 shows 2 transitions of the NAND gate above; although there are 6 possible transitions that switch the output, we show only 2 due to space constraints, although we analyze the sensitivities of all 6 in Table 5 below. 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 5 shows the event-driven sensitivities of the NAND gate delay to

	Parameter	Pull down transitions			Pull up transitions		
Input transition $(A,\ B)$		$(0,\ 0) \to (1,\ 1)$	$(0,\ 1) \to (1,\ 1)$	$(1,\ 0) \to (1,\ 1)$	$(1,\ 1) \to (1,\ 0)$	$(1,\ 1) \to (0,\ 1)$	$(1,\ 1) \to (0,\ 0)$
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 5. NAND gate delay sensitivities with respect to various NMOS, PMOS, and load parameters, for all input transitions that switch the output.

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 cases (although not all), the gate delay is more sensitive to PMOS (NMOS) parameters during "pull up" ("pull down") transitions, as one would intuitively expect. Thus, DAGSENS can be useful for finding parametric sensitivities of important timing-related event-driven objective functions for statistical cell library characterization.

C. Biological applications

We now apply DAGSENS to compute event-driven parametric sensitivities in a biological example, *i.e.*, gene expression via transcription, translation, decay, and diffusion in *Drosophila* embryos (Fig. 11, [34], [35]).

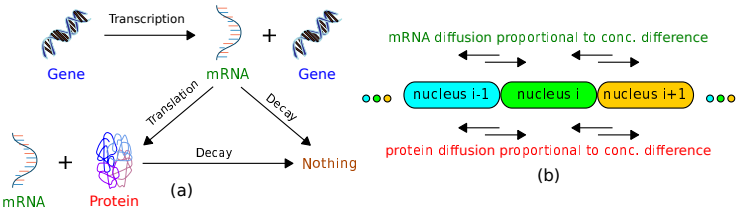


Fig. 11. A model for gene expression in a *Drosophila* embryo, featuring transcription, translation, and decay (part a), as well as diffusion across nuclei (part b).

The system consists of a *Drosophila* gene that generates mRNA molecules via transcription, which in turn generates protein molecules via translation. In parallel, the mRNA and protein molecules also decay. This is all shown in Fig. 11 (a) [34], [35]. Somewhat complicating the process, these reactions take place in a linear chain of sites (called *nuclei*), and whenever there is an mRNA/protein concentration difference between two adjacent nuclei, mRNA/protein molecules flow across nuclei to balance the gap [34], [35].

In this example, we have $N = 52$ nuclei, and each nuclus i (where $1 \le i \le N$) has an mRNA concentration (denoted $[\text{mRNA}]_i$) and a protein concentration (denoted $[\text{protein}]_i$). The system has a single exponentially decaying external input $u(t)$ that governs the rate of transcription. The differential-equation model for the system is:

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

$$\frac{d}{dt}[\text{protein}]_i = \underbrace{\sigma_{\text{protein}}[\text{mRNA}]_i}_{\text{Translation}} + \underbrace{d_{\text{protein}}([\text{protein}]_{i-1} - [\text{protein}]_i)}_{\text{Diffusion from previous nucleus}} + \underbrace{d_{\text{protein}}([\text{protein}]_{i+1} - [\text{protein}]_i)}_{\text{Diffusion from next nucleus}} - \underbrace{\lambda_{\text{protein}}[\text{protein}]_i}_{\text{Decay}}, \quad (26)$$

with the understanding that the "diffusion from previous nucleus" and "diffusion from next nucleus" terms are 0 for the first ($i = 1$) and last ($i = N$) respectively.

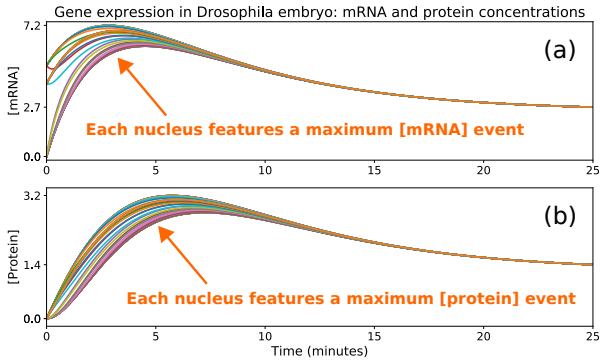


Fig. 12. Transient simulation of gene expression in a *Drosophila* embryo.

Fig. 12 shows a transient run of the system above. As the figure shows, at each nucleus i, there comes a time when $[\text{mRNA}]_i$ reaches its maximum value (before mRNA decay begins to take its toll), and a (slightly later) time when $[\text{protein}]_i$ reaches its maximum value (before protein decay takes its toll). These "maximum concentration" events are of interest in many gene expression systems, and so we set the times of these events, and the corresponding maximum concentration values, to be our event-driven objective functions.

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.

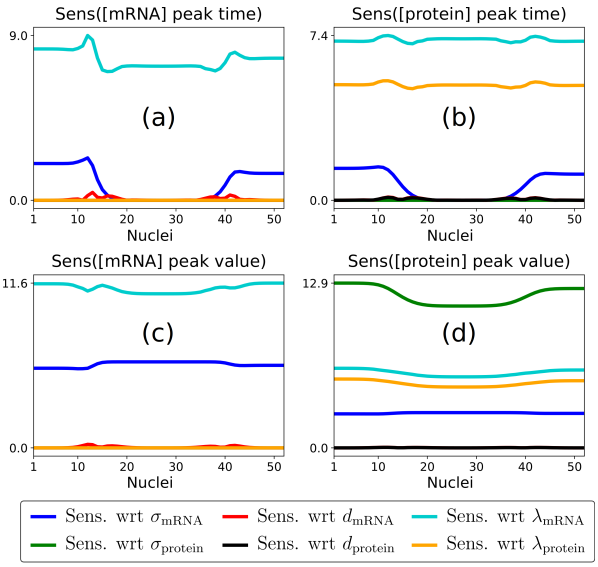


Fig. 13. Sensitivities of peak mRNA and protein concentrations, as well as the times at which these peak concentrations occur, across nuclei, for the *Drosophila* embryo gene expression system.

IV. Summary, Conclusions, and Future Work

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 also shown how DAGSENS can be applied to carry out direct and adjoint transient sensitivity analysis of an entirely new kind of objective function defined based on events that happen during a transient simulation. We have demonstrated this on several real-world applications including high-speed communication, statistical cell library characterization, and gene expression in biological systems.

In future, we would like to significantly improve the DAGSENS codebase, for better CPU and memory performance; in particular, we would like to migrate DAGSENS from a proof-of-concept Python implementation to a production-level C++ implementation in the open-source circuit simulator Xyce [15]. We believe that this would enable us to run DAGSENS on much larger examples than we can at present.

References

[1] J. Nocedal and S. Wright. *Numerical optimization*. Springer-Verlag, New York, 2006.

[2] A. K. Alekseev, I. M. Navon, and M. E. Zelentsov. The estimation of functional uncertainty using polynomial chaos and adjoint equations. *International Journal for Numerical Methods in Fluids*, 67(3):328–341, 2011.

[3] R. M. Errico. What is an adjoint model? *Bulletin of the American Meteorological Society*, 78(11):2577–2591, 1997.

[4] Y. Cao and L. Petzold. A posteriori error estimation and global error control for ordinary differential equations by the adjoint method. *SIAM Journal on Scientific Computing*, 26(2):359–374, 2004.

[5] S. Director and R. Rohrer. The generalized adjoint network and network sensitivities. *IEEE Transactions on Circuit Theory*, 16(3):318–323, 1969.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] R. Bartlett. A derivation of forward and adjoint sensitivities for ODEs and DAEs. Technical report, Sandia National Laboratories, 2008.

[14] Synopsys. HSPICE® user guide: Simulation and analysis, 2010.

[15] 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, Sandia National Laboratories, Albuquerque, NM, USA, 2016.

[16] 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.

[17] A. Griewank and A. Walther. *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. SIAM, 2008.

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

[19] L. W. Nagel. *SPICE2: A computer program to simulate semiconductor circuits*. PhD thesis, UC Berkeley, 1975.

[20] L. Edelstein-Keshet. *Mathematical models in biology*. SIAM, 2005.

[21] A. L. Sangiovanni-Vincentelli. *Computer Design Aids for VLSI Circuits*, chapter Circuit simulation, pages 19–112. Springer, Netherlands, 1984.

[22] L. O. Chua and P. M. Lin. Computer-aided analysis of electronic circuits: Algorithms and computational techniques. 1975.

[23] H. Shichman. Integration system of a non-linear transient network analysis program. *IEEE Transactions on Circuit Theory*, 17(3):378–386, 1970.

[24] C. W. Gear. The numerical integration of ordinary differential equations. *Mathematics of Computation*, 21(98):146–156, 1967.

[25] J. Kleinberg and E. Tardos. *Algorithm design*. Pearson Education, 2006.

[26] C. E. Leiserson, R. L. Rivest, C. Stein, and T. H. Cormen. *Introduction to algorithms*. The MIT Press, 2001.

[27] 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.

[28] 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.

[29] J. A. Davis and J. D. Meindl. *Interconnect technology and design for gigascale integration*. Springer, Netherlands, 2003.

[30] B. Razavi. *Design of analog CMOS integrated circuits*. Tata McGraw-Hill Publishing Company Ltd., New Delhi, India, 2001.

[31] J. L. Stensby. *Phase-locked loops: Theory and applications*. CRC Press, 1997.

[32] 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.

[33] 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.

[34] 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.

[35] 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.