BLAST: Efficient Computation of Nonlinear Delay Sensitivities in Electronic and Biological Networks using Barycentric Lagrange enabled Transient Adjoint Analysis

Arie Meir*‡, and Jaijeet Roychowdhury*
*Department of Electrical Engineering and Computer Science, The University of California, Berkeley, CA, USA
‡Contact author. Email: ariemeir@berkeley.edu

ABSTRACT

Transient waveform sensitivities are useful in optimization and also provide direct insight into system metrics such as delay. We present a novel method for finding parametric waveform sensitivities that improves upon current transient adjoint methods, which suffer from quadratic complexity, by applying barycentric Lagrange interpolation to reduce computation to near linear in the time-interval of interest. We apply our technique to find sensitivities of a "nonlinear" Elmore-delay like metric in digital logic and biochemical pathway examples. Our technique achieves order-of-magnitude speedups over traditional adjoint and direct sensitivity computation.

Categories and Subject Descriptors

B.8.2 [**Integrated Circuits**]: Performance and Reliability—*Performance Analysis and Design Aids*

General Terms

Algorithms, Design, Reliability, Performance

Keywords

Sensitivity Analysis, Circuit Simulation, Computational Modeling

1. INTRODUCTION

Estimating and optimizing gate and interconnect delays have long been central to IC design. With extreme scaling in transistor feature sizes, variability in every step in the manufacturing process – translating to variability in the delay of individual transistors – has become of growing concern [4, 5]. To enable circuit performance metrics (such as critical path delays) to be optimized over the multidimensional parameter space induced by the manufacturing variability, accurate and effective methods for evaluating delay sensitivities are especially important today. In addition to their use in optimization, sensitivities have an immediate merit of its own, allowing designers to obtain insight about the impact that various system parameters have on delay.

Over more than two decades, many models and algorithms have been devised for accurate prediction of delay and its use for optimal design of ICs (*e.g.*, [6, 10, 13, 19–21]). Most of the work on delay modeling and sensitivity calculation, has focused on estimating delays through the use of linear time invariant (LTI) approximations, such as RC or RLC networks [18, 24]. While LTI techniques are appropriate for estimating delays of individual segments of interconnect [19, 20, 23, 25], they can only *approximate* delays through nonlinear elements such as logic gates or sequential elements. Since logic elements typically involve large signal swings and operation in strongly nonlinear regimes (*e.g.*, involving saturation and hysteresis), the appropriateness of LTI approximations can be very suspect. The same is true for systems that involve many nonlinear logic and interconnect elements. Alternative approaches such as [8] apply proprietary, nonlinear metrics for delay optimization, but rely on piecewise-linear device model approximations [11].

In this work, we introduce a novel method for efficient computation of transient sensitivity waveforms, termed BLAST (**B**arycentric **L**agrange **A**djoint **S**ensitivity **T**ransient). We apply this method to analyze the sensitivity of delay to variations in system parameters. To demonstrate the utility of our method, we propose a nonlinear generalization of the classic Elmore delay metric for LTI systems, *i.e.*, a delay metric defined using waveforms in strongly nonlinear systems, that reduces to Elmore delay for LTI systems. This delay metric is directly motivated by the time-domain definition of Elmore delay (using LTI impulse/step responses); it can be computed as a simple post-processing operation after regular (fully nonlinear) transient simulation. A key feature of this delay metric is that it uses information from waveform values over an entire time interval (*i.e.*, not just a single timepoint, or a few discrete timepoints)[1], thereby taking full account of detailed shapes of waveforms. This makes the metric more representative and more broadly applicable than simple alternatives (such as the 50% rise-time point of a step response).

To find the sensitivities of the proposed delay metric to system parameters, we apply transient sensitivity analysis followed by simple post-processing of the sensitivity waveforms. Existing techniques for "efficient" (*i.e.*, adjoint based) transient sensitivity analysis [9, 14, 17] suffer from *quadratic computational complexity* with respect to the length of the time interval of interest, making their application for computing entire waveforms of transient sensitivity (as opposed to the sensitivity at a single time-point) inefficient.

BLAST solves the problem of quadratic time complexity by applying a quadrature technique known as *barycentric Lagrange interpolation* (BLI) [3] to the transient adjoint computation problem. In its essence, BLI is able to approximate a waveform over an interval, to extremely high accuracy, using only a few carefully chosen samples (see section §2.3). As a result, the quadratic time complexity of existing transient adjoint sensitivity methods is reduced to approximately linear. As noted earlier, the waveform sensitivities generated using BLAST are post-processed to obtain network delay sensitivities. The use of BLI also makes this post-processing step highly accurate and computationally inexpensive.

Being able to compute sensitivities of "nonlinear delays" efficiently makes it possible to obtain design insights regarding, *e.g.*, which parameters have the most impact on delay, thus enabling the designer to focus on the most relevant parameters for delay stability — not only for individual logic or cell library components, but for complex combinational or sequential circuits involving many gates and interconnect segments. The near-linear time complexity of BLAST also facilitates use in delay optimization flows, which make repeated calls to sensitivity routines.

Furthermore, BLAST is equally applicable to the domain of quantitative biology, where recent research has focused on novel synthetic genetic and biochemical pathways such as genetic inverters, toggle switches, oscillators, *etc.* [2]. Since such elements are inherently strongly nonlinear, BLAST is particularly well suited for finding their delay sensitivities.

We demonstrate BLAST on examples from electronics and biology, obtaining average speedups of 17× over adjoint sensitivities without BLI, and 30× over direct transient sensitivity computation. We also demonstrate how non-intuitive design insights into the relative importance of parameters in a delay chain can be observed immediately, from the delay

DAC 2012, June 3-7, 2012, San Francisco, California, USA.

[1]Details are provided in the supplementary material §S5.

sensitivities obtained by BLAST. We emphasize that BLAST can be readily applied to any delay metric that uses multiple time points to estimate delay, *i.e.*, it is not limited to our simple Elmore-like metric.
The remainder of the paper is organized as follows. §2 provides brief background on direct and adjoint sensitivity analysis as well a basic overview of the BLI method. We describe the application of BLI to adjoint sensitivities in §3. §4 presents our "nonlinear" Elmore-like delay metric. Results on examples are presented in §5 .

2. PRELIMINARIES

2.1 Direct sensitivity computation for Differential-Algebraic Equations (DAE) systems

We assume that the system of interest is represented as a set of differential-algebraic nonlinear equations [27]:

$$\frac{d}{dt}\left[\vec{q}\left(\vec{x}(t)\right)\right] + \vec{f}\left(\vec{x}(t)\right) + \vec{b}(t) = 0, \tag{1}$$

where $\vec{x}(t)$ represents the internal state vector of dimension n, $\vec{f}(\cdot)$, $\vec{q}(\cdot)$ capture static and dynamic terms, respectively. $\vec{b}(t)$ represents the time-varying input to the system. As we are interested in the sensitivities of the system to external parameters, we further assume that the internal state, and both static and dynamic terms of the system, depend on a vector of parameters $\vec{p}$ of dimension n_p, allowing us to rewrite the DAE as

$$\frac{d}{dt}\left[\vec{q}\left(\vec{x}(t,\vec{p}),\vec{p}\right)\right] + \vec{f}\left(\vec{x}(t,\vec{p}),\vec{p}\right) + \vec{b}(t) = 0. \tag{2}$$

We assume the initial condition is given at time $t = 0$, and it is $\vec{x}(0) = \vec{x_0}$. Moreover, we assume for simplicity that $\vec{x_0}$ and $\vec{b}(t)$ do not depend on $\vec{p}$; it is simple to extend our analysis if this is not the case.
In transient sensitivity analysis, we are interested in finding the time-varying sensitivities of the solution of (2) with respect to $\vec{p}$. To achieve this goal, we first solve the DAE system for some nominal parameter set $\vec{p}_{\text{nom}}$ by running a full transient analysis. Denoting the solution of (2) as $\vec{x_{nom}}(t)$, we let $\Delta\vec{p}$ be a small perturbation to the parameters vector $\vec{p_{nom}}$ and $\Delta\vec{x}$ be a small perturbation to the solution $\vec{x}_{nom}$. We then start the process of linearizing the system (2) around its nominal solution:

$$\begin{aligned}\frac{d}{dt}\left[\vec{q}\left(\vec{x}_{\text{nom}}(t) + \Delta\vec{x}(t), \vec{p}_{\text{nom}} + \Delta\vec{p}\right)\right] \\ + \vec{f}\left(\vec{x}_{\text{nom}}(t) + \Delta\vec{x}(t), \vec{p}_{\text{nom}} + \Delta\vec{p}\right) + \vec{b}(t) = \vec{0}.\end{aligned} \tag{3}$$

Denoting the Jacobian matrices *w.r.t* state variables by

$$\boldsymbol{C}(t) = \left.\frac{\partial\vec{q}(\vec{x},\vec{p})}{\partial\vec{x}}\right|_{\vec{x}_{\text{nom}}(t),\vec{p}_{\text{nom}}},\ \boldsymbol{G}(t) = \left.\frac{\partial\vec{f}(\vec{x},\vec{p})}{\partial\vec{x}}\right|_{\vec{x}_{\text{nom}}(t),\vec{p}_{\text{nom}}}, \tag{4}$$

and the Jacobian matrices *w.r.t* parameters by

$$\boldsymbol{S}_q(t) = \left.\frac{\partial\vec{q}(\vec{x},\vec{p})}{\partial\vec{p}}\right|_{\vec{x}_{\text{nom}}(t),\vec{p}_{\text{nom}}},\ \boldsymbol{S}_f(t) = \left.\frac{\partial\vec{q}(\vec{x},\vec{p})}{\partial\vec{p}}\right|_{\vec{x}_{\text{nom}}(t),\vec{p}_{\text{nom}}}, \tag{5}$$

we can now expand $\vec{q}(\cdot)$ and $\vec{f}(\cdot)$ in first-order Taylor series and simplify (3) to

$$\frac{d}{dt}\left[\boldsymbol{C}(t)\,\Delta\vec{x}(t) + \boldsymbol{S}_q(t)\,\Delta\vec{p}\right] + \boldsymbol{G}(t)\,\Delta\vec{x}(t) + \boldsymbol{S}_f(t)\,\Delta\vec{p} \simeq \vec{0}. \tag{6}$$

Denoting the sensitivity matrix $\boldsymbol{M}(t) = \left.\frac{\partial\vec{x}}{\partial\vec{p}}\right|_{\vec{x}_{\text{nom}}(t),\vec{p}_{\text{nom}}}$ and "dividing" by $\Delta\vec{p}$, it can be shown that (6) is equivalent to

$$\frac{d}{dt}\left[\boldsymbol{C}(t)\,\boldsymbol{M}(t) + \boldsymbol{S}_q(t)\right] + \boldsymbol{G}(t)\,\boldsymbol{M}(t) + \boldsymbol{S}_f(t) = \vec{0}. \tag{7}$$

For clarity, we note that $\boldsymbol{C}(t), \boldsymbol{G}(t) \in \mathbb{R}^{n\times n}$ and $\boldsymbol{S}_q(t), \boldsymbol{S}_f(t), \boldsymbol{M}(t) \in \mathbb{R}^{n\times n_p}$. $\boldsymbol{M}(t)$ is the matrix of transient sensitivities we are interested in. Rearranging (7) into the form of (2) yields:

$$\frac{d}{dt}\left[\boldsymbol{C}(t)\,\boldsymbol{M}(t)\right] + \boldsymbol{G}(t)\,\boldsymbol{M}(t) + \left\{\underbrace{\frac{d}{dt}\left[\boldsymbol{S}_q(t)\right] + \boldsymbol{S}_f(t)}_{\boldsymbol{S}(t)}\right\} = \vec{0}. \tag{8}$$

If (8) is solved as a matrix initial value problem with initial condition $\boldsymbol{M}(0) = \boldsymbol{0}$, we obtain transient sensitivities in "direct" fashion. Note that (8) can be solved as n_p separate vector DAE systems, using the columns of $\boldsymbol{S}$ as inputs and solving to obtain the columns of $\boldsymbol{M}$. Once $\boldsymbol{M}(t)$ is available, it is easy to find $\Delta\vec{x}$ in (6) via

$$\Delta\vec{x}(t) = \boldsymbol{M}(t)\,\Delta\vec{p}. \tag{9}$$

For a small number of parameters n_p, direct sensitivity computation as above involves the same order of computation as the transient solution itself. However, if the system size n and the number of parameters n_p are both large (furthermore, $n_p \gg n$ for typical real-life systems), one has to keep track of the $n_p \times n$ entries of the sensitivity matrix at every time point of the simulation, increasing computation and memory requirements to the point of infeasibility. Getting around this computational bottleneck is the primary motivation for *adjoint* transient sensitivity, where the sensitivities of a *few* selected "outputs", with respect to *all* parameters, can be obtained more efficiently than by the above direct route.
A practical note concerning the computation of $S(t)$ is appropriate at this point. manual, analytic differentiation of $f(\cdot)$ and $q(\cdot)$ to obtain entries of $S(t)$ tends to be laborious and error prone. Run-time automatic differentiation [12, 22] is an attractive solution for this issue, though it is typically more compute-intensive than the use of hard-coded derivatives. In this work, we have extended [22] and applied the extended automatic differentiation method to compute $S(t)$.

2.2 Adjoint Operator for Linear Differential Equations

A linear differential equation can be written in the form of (2) as:

$$\frac{d}{dt}\boldsymbol{C}(t)\,\vec{x}(t) + \boldsymbol{G}(t)\,\vec{x}(t) = \vec{u}(t). \tag{10}$$

(10) can be viewed as a linear operator $\mathcal{L}$: $\vec{u}(t) \mapsto \vec{x}(t)$ *i.e.*, a linear mapping between inputs $\vec{u}(t)$ in a domain $\mathcal{D}$ and solutions $\vec{x}(t)$ in a range $\mathcal{R}$. It is a well known fact that any linear mapping has an adjoint operator [15]. The adjoint operator, usually denoted by $\mathcal{L}^\dagger$: $\vec{y}(t) \mapsto \vec{z}(t)$, takes its input in the range space $\mathcal{R}$ and produces its output in the domain space $\mathcal{D}$. Rewriting equation (6) in the same form as (10), we get

$$\frac{d}{dt}\left[\boldsymbol{C}(t)\,\Delta\vec{x}(t)\right] + \boldsymbol{G}(t)\,\Delta\vec{x}(t) = \vec{u}(t) = -\boldsymbol{S}(t)\Delta\vec{p}. \tag{11}$$

Solving (11) defines the mapping $\mathcal{L}$: $\vec{u}(t) \mapsto \Delta\vec{x}(t)$. It can be shown[2] that the adjoint operator $\mathcal{L}^\dagger$: $\vec{y}(t) \mapsto z(t)$ of $\mathcal{L}$: $\vec{u}(t) \mapsto \Delta\vec{x}(t)$ is given by the differential equation

$$-C^*(t)\frac{d}{dt}\vec{z}(t) + G^*(t)\vec{z}(t) = \vec{y}(t). \tag{12}$$

Assume we are interested in the sensitivities of some scalar output defined as $d(t) = \vec{c}^*\vec{x}(t)$, where $\vec{c}^*$ is a row vector; assume further that we are specifically interested only in the sensitivity $d(T_0)$, *i.e.*, limiting our attention to a specific time point $t = T_0$ on the integration interval $[0, T]$. The reason for this will become clearer in the following section, as we demonstrate how finding the sensitivity at a small number of carefully chosen points allows us to approximate the sensitivity for any time point $t \in [0, T]$.
It can be shown[3] that the sensitivity vector of the output $d(t)$ at a point $t = T_0$, denoted $\vec{m}_d(T_0)$, is given by

$$\vec{m}_d(T_0) = \frac{\vec{c}^*\Delta x(T_0)}{\Delta\vec{p}} = -\int_0^T \vec{z}^*_{\vec{c},T_0}(\tau)\,\boldsymbol{S}(\tau)d\tau, \tag{13}$$

where $\vec{z}_{\vec{c},T_0}(t)$ is obtained by solving the adjoint system

$$-\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{c}\,\delta(t - T_0), \tag{14}$$

backwards from $t = T$ to $t = 0$, with initial condition $\vec{z}(T) = \vec{0}$. The reader is invited to follow our analysis in §S3 of the supplementary material; equations (13) and (14) are obtained simply by replacing $\vec{e}_1$ by $\vec{c}$.
Note that the sub-indices of $\vec{z}_{\vec{c},T_0}(t)$, namely $\vec{c}$ and T_0, indicate that the solution of the adjoint system (12) depends on the input $\vec{y}(t) = \vec{c}\delta(t - T_0)$.

[2]Details are provided in the supplementary material §S2.
[3]See the supplementary material §S3.

Observing equation (14) reveals that it cannot immediately be solved using standard numerical integration methods, because the input on the RHS is a δ-function. Numerical methods are not well suited for δ-function inputs, which involve an infinite value at a single time-point. Therefore, further analytical machinery is needed to re-phrase (14) in a form suitable for numerical integration. To simplify this analysis, we now restrict ourselves to ODEs: *i.e.*, $\vec{q}(\vec{x}) \equiv \vec{x}$ in (2), leading (*w.l.o.g.* for invertible $\boldsymbol{C}$) to $\boldsymbol{C}(t) \equiv I_{n\times n}$. For the ODE case, (14) thus becomes

$$-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{c}\;\delta(t - T_0). \tag{15}$$

It can be shown[4] that finding the solution of (15) over the interval $[0, T]$ is equivalent to solving the homogeneous part of (15) *i.e.*,

$$-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{0}, \tag{16}$$

with initial condition $\vec{z}(T_0) = \vec{z}_{\vec{c},T_0}(T_0) = \vec{c}$, backwards from $t = T_0$ to $t = 0$. Applying this result, we can compute the sensitivity $\vec{m}_d(T_0)$ using standard numerical methods: we first obtain the solution of the homogeneous system (16) using $\vec{z}(T_0) = \vec{c}$ as the initial condition, then compute the integral (13). In section §3, we will define an algorithm for obtaining $\vec{m}_d(T_0)$ based on this analysis.

2.3 Lagrange interpolation and Chebyshev Nodes

Details about Lagrangian interpolation can be found in most textbooks on numerical analysis such as [1], but in the interest of self-sufficiency, we briefly survey the basic concepts of the method here. Given a function $f(x)$ and p sample points $\{c_1, \cdots, c_p\}$, the unique polynomial of degree $p-1$ that agrees with $f(\cdot)$ at each of these points is

$$L(x) = \sum_{m=1}^{p} u_m(x) f(c_m), \tag{17}$$

where $u_m(x)$ is the m^{th} Lagrange polynomial (of degree $p-1$)

$$u_m(x) = \frac{\prod_{\substack{k=1\\k\neq m}}^{p}(x - c_k)}{\prod_{\substack{k=1\\k\neq m}}^{p}(c_m - c_k)}. \tag{18}$$

p is referred to as the interpolation order.

In this work, we are interested in approximating the true sensitivity waveform $\vec{m}_d(t)$ by an Lagrangian approximation $L(t)$ which agrees with $\vec{m}_d(t)$ exactly at a small number p of sample points $\{c_1, \cdots, c_p\}$. To estimate the quality of such an approximation, a bound on the error at any point in the interval $t \in [0, T]$ is useful. By choosing the sample points c_i to be Chebyshev nodes over the interval $[0, T]$, it can be shown that the error goes down faster than exponentially with respect to the interpolation order p. Chebyshev nodes on $[a, b]$, are defined as

$$c_m = \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{(2m-1)\pi}{2p}\right), \quad m = 1, \cdots, p. \tag{19}$$

With Chebyshev interpolation nodes, the approximation error over $[a, b]$ is bounded as

$$|f(x) - L(x)| \leq \frac{2(\frac{b-a}{4})^p}{p!} \max_{\xi\in[a,b]} |f^{(p)}(\xi)|. \tag{20}$$

Therefore, if the interval $[a, b]$ is fixed, and the derivatives of $f(\cdot)$ over this interval are bounded, the approximation error falls faster-than-exponentially (due to the factorial term) with p [7]. As a rule of thumb, choosing p in the range 15-20 typically leads to double-precision accuracy, *i.e.*, the "approximation" is as good as the original function for all computational purposes. In our case, $[a, b]$ is the integration interval $[0, T]$, which is fixed, and the sensitivity function $\vec{m}_d(t)$ is assumed to be smooth, *i.e.*, to have bounded derivatives, the error bound (20) holds. This allows us to choose a relatively small interpolation order p and evaluate $\vec{m}_d(t)$ using the approximation polynomial $L(t)$, which for practical purposes is equivalent to evaluating the original function $\vec{m}_d(t)$. We demonstrate the dependence of the approximation error on p when we discuss the performance of BLAST[5].

[4]See the supplementary material §S4.

[5]See the supplementary material §S1.

Using the approximation $L(x)$ instead of the original function $f(x)$ becomes computationally advantageous if a) computing $L(x)$ at each x is expensive and b) values of $f(x)$ are needed at many (*i.e.*, $\gg p$) points x. The key to this efficiency is that only p evaluations of $f(x)$ are needed to set up the approximation $L(x)$ in (17). Once this is done, $L(x)$ can be evaluated very cheaply (O(p) using Barycentric techniques, see §2.4 below) for any value of x.

2.4 Barycentric Lagrange Interpolation

In the form based on (18), each evaluation of $L(x)$ requires $O(p^2)$ additions and multiplications. Another potential impediment is having to recompute all evaluations from scratch once a new sampling point c_{p+1} is been added. An improvement over the classic Lagrange form, described in [3], implements the computation of $u_m(x)$ as

$$u_m(x) = u(x)\frac{w_m}{x - c_m}, \tag{21}$$

where

$$u(x) = \prod_{k=1}^{p}(x - c_k) \tag{22}$$

and w_m are the *barycentric weights*, defined as

$$w_m = \frac{1}{\prod_{\substack{k=1\\k\neq m}}^{p}(c_m - c_k)}. \tag{23}$$

Using (22) and (23), the interpolation polynomial can be written as

$$L(x) = u(x)\sum_{m=1}^{p}\frac{w_m}{x - c_m} f(c_m), \tag{24}$$

which is referred to as the "first form of the barycentric interpolation formula" by Rutishauser in [28]. This formula requires $O(p^2)$ flops for the initial computation of the barycentric weights. The barycentric weights are independent of x, therefore the cost of their computation is amortized over a large number of function evaluations. In addition, each evaluation requires $O(p)$ flops for evaluating $L(x)$, once the weights are known. As shown in [3], incorporating a new node c_{p+1}, also requires $O(p)$ additional operations, an improvement over the direct Lagrange form approach.

To summarize, we have shown that the transient sensitivity waveform $\vec{m}_d(t)$ can be approximated well by the interpolation polynomial $L(t)$. Moreover, using the barycentric form of $L(t)$ outlined in (24) we can now evaluate $L(t)$ at any point, $t \in [0, T]$ at the cost of $O(p)$ flops.

3. BLI BASED TRANSIENT ADJOINT SENSITIVITY COMPUTATION

We now describe how to apply Barycentric Lagrange Interpolation to speed up adjoint-based sensitivity computation when sensitivities are needed not just at one timepoint, but over an entire interval. We first apply the results of §2.2 to present an algorithm for evaluating the sensitivity vector $\vec{m}_d(T_0)$ at a specific point $T_0 \in [0, T]$. Assuming that the sensitvity function is smooth, we then apply the results of section §2.3 and §2.4 to arrive at an algorithm which computes the sensitivity waveform $\vec{m}_d(t)$, over the entire interval $[0, T]$. We compute $\vec{m}_d(t)$ efficiently by evaluating the sensitivity function at a small number of interpolation nodes p, and then apply the BLI method to evaluate the approximation polynomial $L(t)$ at any other point in the interval for a small computational price of $O(p)$ operations.

3.1 Local sensitivity vector using adjoint equation

Assume, as before, that our scalar output of interest is $d(t) = \vec{c}^*\vec{x}(t)$. The sensitivity vector of this output with respect to all parameters was given by equation (13), rewritten here for convenience:

$$\vec{m}_d(T_0) = -\int_0^{T_0} \vec{z}^*_{\vec{c},T_0}(\tau)\,\boldsymbol{S}(\tau)d\tau. \tag{25}$$

Note that the upper integration limit is effectively T_0, since the adjoint solution $\vec{z}_{\vec{c},T_0}(t)$ is zero for the interval $t \in [T_0, T]$.

Algorithm 1, presented below utilizes (25) to compute the sensitivity vector $\vec{m}_d(T_0)$:

***Algorithm 1**: Local Sensitivity*

Input: Circuit DAE D, timepoint of interest T_0, output vector $\vec{c}$, initial circuit state $\vec{x}_0$, final time T
Output: Sensitivity vector $\vec{m}_d(T_0)$

1 $\vec{p} \longleftarrow \vec{p}_{nom}$;
2 $\vec{x}_{nom}(t) \longleftarrow$ Numeric solution of (2) over $t \in [0, T_0]$ with i.c. $\vec{x}_0$;
/* See equation (8) for details */
3 $\boldsymbol{G}(t) \longleftarrow \left.\frac{d\vec{f}(\vec{x},\vec{p})}{d\vec{x}}\right|_{\vec{x}_{\rm nom}(t),\vec{p}_{\rm nom}}$;
4 $\boldsymbol{S}(t) \longleftarrow \frac{d}{dt}[\boldsymbol{S}_q(t)] + \boldsymbol{S}_f(t) = \frac{d}{dt}\left[\left.\frac{d\vec{q}(\vec{x},\vec{p})}{d\vec{p}}\right|_{\vec{x}_{\rm nom}(t),\vec{p}_{\rm nom}}\right] + \left.\frac{d\vec{f}(\vec{x},\vec{p})}{d\vec{p}}\right|_{\vec{x}_{\rm nom}(t),\vec{p}_{\rm nom}}$
/* See §S4 of supplementary material */
5 $\vec{z}_{\vec{c},T_0}(t) \longleftarrow$ solution of equation (16) *backwards from* $t = T_0$ *to* $t = 0$ *with "initial" condition* $\vec{z}_{\vec{c},T_0}(T_0) = \vec{c}$;
6 $\vec{m}_d(T_0) \longleftarrow -\int_0^{T_0} \vec{z}^*_{\vec{c},T_0}(\tau)\boldsymbol{S}(\tau)\, d\tau$;
7 **return** $\vec{m}_d(T_0)$;

3.2 Sensitivity computation using barycentric Lagrange interpolation (BLI)

In many real-life scenarios, such as for the purpose of computing delay sensitivities, the whole transient sensitivity waveform $\vec{m}_d(t)$ over a period of time $t \in [0, T]$ is of interest.

When using direct sensitivity computation, computing $\vec{m}(t)\ \forall t \in [0, T]$, is essentially the same effort as computing $\vec{m}(t_0)$: a single, albeit expensive, solution of (8) over $[0, T]$ provides $\boldsymbol{M}(t)$, from which $\vec{m}_d(t)$ can be obtained via $\vec{m}_d(t) = \vec{c}^*\,\boldsymbol{M}(t)$. Requiring no additional computation for finding all of $\vec{m}_d(t)$ is an attractive feature of direct sensitivity computation.

However, finding all of $\vec{m}_d(t)$ using adjoint sensitivities to compute quantities in in (25) requires the adjoint algorithm to be re-run for each sample point $t \in [0, T]$ - which requires $O((n+n_p)t)$ operations. Hence the total computation needed for finding $\vec{m}_d(t)\ \forall t \in [0, T]$ is $O((n+n_p)T^2)$. This quantity grows quadratically with T, an undesirable expense that is due to the non-incremental nature of the adjoint transient sensitivity computation algorithm.

In this section, we apply the BLI method outlined in §2.4 to speed up adjoint computation of $\vec{m}_d(t),\ \forall t \in [0, T]$. The algorithm we present computes $\vec{m}_d(t),\ \forall t \in [0, T]$ in $O((n + n_p)T)$ floating point operations. Consider (25), which we rewrite as

$$\vec{m}_d(T_0) = -\int_0^{T_0} \boldsymbol{S}^*(\tau)\ \vec{z}_{\vec{c},T_0}(\tau)\ d\tau. \tag{26}$$

We have explicitly used the notation $\vec{z}_{\vec{c},T_0}(t)$, as a reminder that the solution $\vec{z}(t)$ of the adjoint system, depends on the output vector $\vec{c}$ and the time point T_0. In particular, note that the adjoint solution $\vec{z}_{\vec{c},T_0}$ satisfies (see section §S4 of the supplementary material for details):

$$\vec{z}_{\vec{c},T_0}(T_0) = \vec{c}, \text{ and} \tag{27}$$

$$\vec{z}_{\vec{c},T_0}(\tau) \equiv \vec{0} \quad \forall \tau > T_0. \tag{28}$$

If we sample the interval $[0, T_0]$ using N_{T_0} samples $\{\tau_1, \cdots, \tau_{N_{T_0}}\}$, and approximate (26) by a simple discrete summation (for illustration), we obtain

$$\vec{m}_d(T_0) \simeq \vec{m}_{d_{approx}}(T_0) \triangleq -\sum_{i=1}^{N_{T_0}-1} \boldsymbol{S}^*(\tau_i)\ \vec{z}_{\vec{c},T_0}(\tau_i)\ \Delta_i, \tag{29}$$

where $\Delta_i \triangleq \tau_{i+1} - \tau_i$ and $\vec{m}_{d_{approx}}(t)$ denotes a discrete numerical approximation to $\vec{m}_d(t)$.

Suppose we fix a set of time-points

$$\mathcal{T}_T \triangleq \{\tau_0, \tau_1, \cdots, \tau_{N_T}\} \tag{30}$$

for discretizing the interval $[0, T]$, and want to re-use $\mathcal{T}_T$ for calculating (29) for all $t \in [0, T]$. Then it is convenient to re-write (29) as a summation over $\mathcal{T}_T$, using (28) to define $\vec{z}_{\vec{c},T_0}(\tau)$ when $\tau > T_0$, as

$$\vec{m}_{d_{approx}}(t) = -\sum_{i=1}^{N_T} \boldsymbol{S}^*(\tau_i)\ \vec{z}_{\vec{c},T_0}(\tau_i)\ \Delta_i. \tag{31}$$

We would like to find $\vec{m}_{d_{approx}}(t)$ for each $t \in \mathcal{T}_T$. Computing (31) directly for each $t \in \mathcal{T}_T$, would require N_T separate summations, each of which has N_T terms – requiring a total computation of $O(N_T^2 \times (n+n_p))$. To compute (26) efficiently for many values of $t \in [0, T]$, we have applied the BLI method outlined in §2.3 and §2.4. Interpolating $\vec{m}_{d_{approx}}(t)$ we get:

$$\vec{m}_{d_{approx}}(t) \simeq \sum_{j=1}^{p} u_j(t)\vec{m}_{d_{approx}}(c_j), \tag{32}$$

where c_j are the Chebyshev nodes on $[0, T]$, and $u_j(t)$ are the Barycentric Lagrange polynomials (see (21)). Assuming $\vec{m}_d(t)$ is smooth, *i.e.*, its derivatives are bounded over $[0, T]$, we have the approximation error falling faster-than-exponentially with p, as presented in section §2.3.

Algorithm 2 presented below, is used to compute the sensitivity waveform $\vec{m}_{d_{approx}}(t)$ in the interval $[0, T]$:

Algorithm 2: Sensitivity Waveform Computation

Input: Circuit DAE D, output vector $\vec{c}$, interpolation order p, time discretization $\mathcal{T}_T$, initial circuit state at time 0 x_0, final time T
Output: Sensitivity vector waveform $\vec{m}_{d_{approx}}(t) \quad \forall t \in [0, T]$

1 $a \longleftarrow 0$;
2 $b \longleftarrow T$;
/* Evaluate sensitivity at Chebyshev nodes */
3 **for** *m=1* **to** *p* **do**
  /* See equation (19) */
4 $c_m \longleftarrow \frac{a+b}{2} + \frac{b-a}{2}\cos\left(\frac{(2m-1)\pi}{2p}\right)$;
5 $\vec{m}_{basis}(m) \leftarrow \boldsymbol{LocalSensitivity}(D, c_m, \vec{c}, x_0, T)$;
6 **end**
/* Interpolate sensitivity $\forall t \in [0, T]$ */
7 **for** *i=1* **to** N_T **do**
8 $\vec{m}_{d_{approx}}(i) = \sum_{j=1}^{p} u_j(t)\vec{m}_{basis}(j)$
9 **end**
10 **return** $\vec{m}_{d_{approx}}(t)$;

We assume that the Barycentric Lagrange polynomials, $u_j(t)$ (21), are precomputed in advance and available at step 7.

3.3 Spatial and temporal complexity considerations

Consider the computation and memory requirements of Algorithm 1:

1. Step 2 requires $O(nT)$ in both computation and memory.
2. Step 5 also requires $O(nT)$ in both computation and memory.
3. Computing the $\vec{m}_d(T_0)$ integral numerically in step 6 requires $O((n+n_p)T)$ flops, since $\boldsymbol{S}(t)$ is *sparse* rectangular of size $n \times n_p$.

Hence, the overall time required to compute adjoint sensitivity at a single time point is $O((n + n_p)T)$. For large n_p, this is far superior to the direct sensitivity procedure, where finding $\boldsymbol{M}(t)$, a dense matrix of size $n \times n_p$, requires $O(nn_pT)$ storage and $O(nn_pT)$ computation.

For Algorithm 2, the computational cost breakdown is:

1. p iterations of Algorithm 1 in steps 3−6, which cost $O(pT(n+n_p))$.
2. Evaluating $\vec{m}_{d_{approx}}(t)$ at each point using the BLI polynomials, which requires $O(n_pp)$ flops. Assuming a discretized $[0, T]$ interval with $O(T)$ time points, leads to total computational effort of $O(Tn_pp)$ flops for steps $7 - 9$.

Overall, the total computational effort required for evaluating the entire waveform $\vec{m}_d(t)$ over $[0, T]$ via (32) is $O(pT(n + n_p) + Tn_pp)$. If the interval $[0, T]$ is fixed, p is typically a small constant, hence this computation is much faster that the quadratic complexity of brute-force evaluation of, *e.g.*, (31).

4. A "NONLINEAR" DELAY METRIC

Substantial efforts have been invested by the CAD research community to predict and optimize circuit delay. As mentioned in the introduction, previous works typically assume small-signal conditions when analyzing delay [19, 20, 23, 25], and rely heavily on those assumptions. However, several delay optimization works use proprietary, nonlinear delay metrics, *e.g.*, in the context of underlying piecewise-linear device model functions [8, 11]. Here, we propose an Elmore-delay [10] like metric for nonlinear

systems. We then apply the transient sensitivities we have computed using BLAST in order to find delay sensitivities to various system parameters. Given a Linear Time Invariant (LTI) system with impulse response $h_A(t)$ (at some output/node A), the classic Elmore-delay metric is defined as:

$$T_d(A) = \int_0^\infty h_A(t) \cdot t \; dt. \tag{33}$$

Motivated by this definition, we define a delay metric $T_{d_{NL}}(A)$ that uses the response at A of any system, linear or nonlinear, to a step input. Denoting this response by $g_A(t)$, we define:

$$T_{d_{NL}}(A) = \frac{\int_0^\infty g'_A(t) \cdot t \; dt}{\int_0^\infty g'_A(t) \; dt}. \tag{34}$$

For LTI systems, this definition reduces to the classic Elmore delay metric, since the derivative of a step response is the impulse response which Elmore's original metric is based upon. However, (34) also applies to nonlinear systems, for which the impulse-response based Elmore delay formula (33) cannot be used.

Computing the sensitivities of this delay using standard adjoint methods runs is quadratic in the time interval of the simulation, as described in §3.2. By using our BLI-based adjoint sensitivity procedure BLAST, however, the complexity is reduced to linear. Note that BLAST can be readily applied to find sensitivities of any other delay metric which uses information from multiple time-points.

5. RESULTS

5.1 Delay sensitivity in a digital inverter chain

We apply BLI-based transient adjoint sensitivity computation to the delay metric in section §4 to analyze delay sensitivities in a chain of three CMOS inverters, shown in Fig. 1. To compute delay sensitivities (in other words, to find the vector $\frac{dT_{d_{NL}}}{d\vec{p}}$), the entire sensitivity waveform is required; we obtain this waveform from the algorithm in section §3.2.

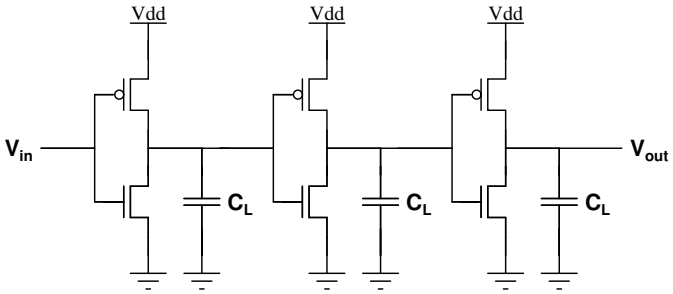

Figure 1: Three stage CMOS 3 inverter chain

An example of the sensitivity waveform for the inverter chain can be seen in Fig. 2, where the direct sensitivity and the adjoint sensitivity are presented side by side.

The sensitivities of the output delay with respect to all model parameters are presented in Table 1, where column I_j lists the sensitivities to parameters in the j^{th} stage of the 3-stage chain.

Parameter	I_1	I_2	I_3
V_{DD}	+10.67E-02	-84.31E-04	-12.09E-04
β_N	+31.19E-07	+12.48E-08	-11.02E-09
β_P	+50.03E-04	-40.31E-06	-12.33E-06
V_{T_N}	-61.90E-03	-15.91E-03	+47.31E-04
V_{T_P}	-53.42E+00	+23.20E-03	+16.02E-04
R_{DS_N}	+34.56E+01	-21.69E+01	+81.04E-01
R_{DS_P}	-99.42E+01	+21.61E+01	-45.86E+00
C_L	-14.56E-09	+12.38E-10	+10.02E-11

Table 1: Digital inverter chain output delay sensitivities to parameters of the different stages

As might be expected given the fact that each nonlinear inverter has an "amplifying" effect on parts of the input waveform, the output delay is much more sensitive to parameters in earlier stages of the chain than to those in later stages, *i.e.*, the impact of the first stage inverter is much larger on the delay, than the impact of the last stage.

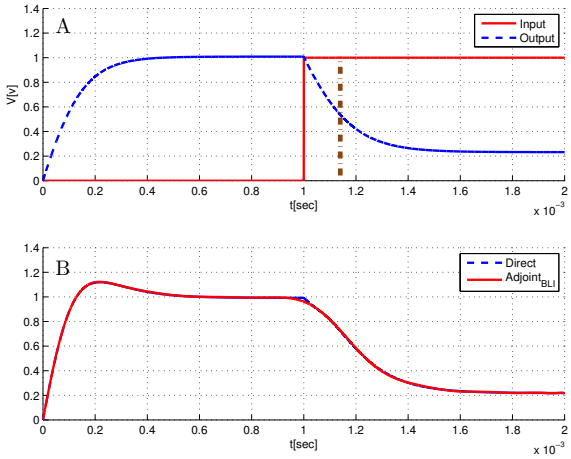

Figure 2: A. Inverter response to a step input, and the delay (vertical line) found by the delay metric. B. Sensitivity of an inverter output to V_{DD} under a step input

5.2 Genetic "inverter-chain" sensitivity analysis

Over more than a decade, research in synthetic biology has focussed on designing and implementing simple, artificial, biological networks. The intention of this research direction is to design standard, modular building blocks in order to build larger artificial biological systems. Another important motivation is to study the behaviour of simple, canonical biocircuits in order to gain insight into the dynamics of real metabolic pathways [2, 26]. In [16], a synthetic transcriptional cascade is described, and its outputs manually analysed with respect to their sensitivity to design parameters. We demonstrate the applicability of BLAST by computing the delay sensitivities of this transcriptional cascade.

Although functionally similar, the underlying implementation mechanism and the mathematical model of the biological inverter is rather different from its electronic counterpart: a schematic description of a single inverter is presented in Fig. 3. In a very crude approximation, the presence of a transcription factor **tf1** induces the promoter **ip1** and results in the production of a protein **dr1**, which in turn represses the production of a fluorescent report protein **fp1** downstream by blocking its promoter **rp1**. To chain the standalone inverters into a cascade, the second output is engineered to produce a repressor protein for the next stage in all but the last stage, which produces a fluorescent protein for output readout. The input to the system is encoded in the amount of the transcription factor **tf1**.

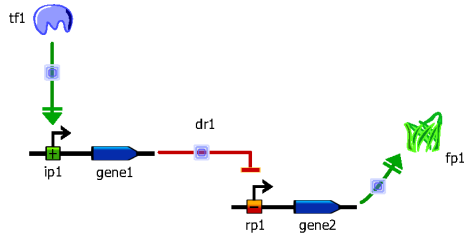

Figure 3: Schematic diagram of a biochemical inverter

Our model for the single biological inverter, based on a simplified version of [16], consists of the following bio-chemical reactions:

$$R + pR_a \rightarrow pR_i, \tag{35}$$
$$pR_i \rightarrow R + pR_a, \tag{36}$$
$$pR_a \rightarrow pR_a + mRNA, \tag{37}$$
$$mRNA \rightarrow mRNA + GFP, \tag{38}$$
$$mRNA \rightarrow \phi, \tag{39}$$
$$GFP \rightarrow \phi. \tag{40}$$

In the above, *R* stands for repressor, prR_a for active promoter, pR_i for inactivated promoter, *mRNA* stands for the messenger RNA synthesized

during transcription and *GFP* represents the **G**reen **F**lourescent **P**rotein, which acts as the output of the inverter. In this reaction system, promoter inactivation is modeled by (35) and (36), transcription is modeled by (37), translation by (38) and the degradation of mRNA and GFP by equations (39) and (40) respectively.

The sensitivities of the output delay to the parameters of the different stages are shown in Table 2; sensitivity waveforms are shown in Fig. 4. Remaining consistent with the digital electronics example, column I_j lists the sensitivities to the parameters of the j^{th} stage.

Parameter	I_1	I_2	I_3
k_{on}	-52.81E-03	+38.21E-04	-41.01E-04
k_{off}	+26.44E-03	-19.33E-04	+20.21E-04
$k_{transcription}$	+41.23E-03	-43.16E-03	-31.29E-04
$k_{translation}$	+38.55E-03	-42.31E-03	-27.91E-04
k_{deg_p}	-18.43E-03	+19.97E-03	-30.96E-05
k_{deg_m}	-13.10E-04	+19.11E-04	+12.90E-06
p_{tot}	+45.04E-01	+45.95E-02	-55.11E-03

Table 2: Biological inverter chain output delay sensitivities to parameters of the different stages

It can be observed that in the biological case too, the sensitivity of the delay at the output is affected more by parameters in the first stage of the cascade. In fact, due to the large variability in the distinct biological implementations of the same basic component, and the inherent, often undesired, inter-component coupling through the reaction medium, sensitivity analysis can assist in estimating compatibility between various components of the same bio-circuit.

This example demonstrates how BLAST is as relevant to the biological domain as it is to electronics. Indeed, we believe that with the rising prominence of synthetic biological design techniques, the rôle of delays in the design of complex biochemical pathways will achieve an importance equalling or surpassing that in electronic/digital design.

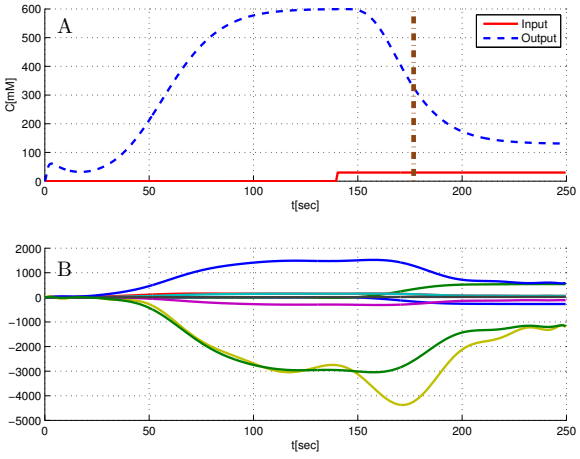

Figure 4: A. Genetic inverter response to a step input and the delay (vertical line) found by the delay metric. B. Sensitivity of genetic inverter output to parameters under a step input

5.3 Performance evaluation

To evaluate computational performance, we measured the run-time of BLAST, and compared it with the direct sensitivity computation as well as the full adjoint computation(without BLI). Sensitivities via BLAST feature average speedups factors of **29.81** and **17.24** when compared to direct and adjoint sensitivities, respectively. Charts and additional performance results are provided in §S1 in the supplementary material.

6. SUMMARY AND CONCLUSIONS

In this paper, we have presented BLAST: an efficient method for computing entire waveforms of transient sensitivity with respect to parameters. We have also proposed an Elmore-delay like metric to estimate delay in nonlinear systems, and as an example, have successfully applied BLAST to calculate the delay sensitivities of an inverter chain to its system parameters. We have shown that BLAST achieves an average speedup of ~ 30 compared to direct transient sensitivities, and an average speedup of ~ 17 relative to traditional adjoint. We have also shown how BLAST can be readily applied as a design tool in a synthetic biology application. Our hope for the future is that by applying design tools such as BLAST from the CAD community to biological applications, we can contribute to a transformation in the field of quantitative biology, similar to the one that reshaped the field of digital electronics in the 1970s.

7. REFERENCES

[1] F. Acton. *Numerical methods that work*. Spectrum Series. Mathematical Association of America, 1990.

[2] E. Andrianantoandro, S. Basu, D. Karig, and R. Weiss. Synthetic biology: new engineering rules for an emerging discipline. *Mol Syst Biol*, 2:2006, 2006.

[3] J. Berrut and L. Trefethen. Barycentric lagrange interpolation. *SIAM Review*, 46(3):501, 2002.

[4] D. Boning and S. Nassif. Models of process variations in device and interconnect. In *Design of High Performance Microprocessor Circuits, chapter 6*. IEEE Press, 1999.

[5] K. Bowman, S. Duvall, and J. Meindl. Impact of die-to-die and within-die parameter fluctuations on the maximum clock frequency distribution for gigascale integration. *Solid-State Circuits, IEEE Journal of*, 37(2):183–190, Feb 2002.

[6] P. Chan and K. Karplus. Computing signal delay in general rc networks by tree/link partitioning. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 9(8):898–902, Aug 1990.

[7] W. Cheney and D. Kincaid. *Numerical Mathematics and Computing*. Brooks/Cole, Pacific Grove, CA, 3d edition, 1994.

[8] A. Conn, I. Elfadel, J. Molzen, W.W., P. O'Brien, P. Strenski, C. Visweswariah, and C. Whan. Gradient-based optimization of custom circuits using a static-timing formulation. In *Design Automation Conference, 1999. Proceedings. 36th*, pages 452 –459, 1999.

[9] S. Director and R. Rohrer. The Generalized Adjoint Network and Network Sensitivities. *IEEE Trans. Ckt. Theory*, 16:318–323, August 1969.

[10] W. C. Elmore. The transient response of damped linear networks with particular regard to wideband amplifiers. *Journal of Applied Physics*, 19(1):55–63, Jan 1948.

[11] P. Feldmann, T. Nguyen, S. Director, and R. Rohrer. Sensitivity computation in piecewise approximate circuit simulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 10(2):171 –183, feb 1991.

[12] A. Griewank. On automatic differentiation. *Mathematical Programming: recent developments and applications*, pages 83–108, 1989.

[13] R. Hitchcock, G. Smith, D. Cheng, et al. Timing analysis of computer hardware. *IBM Journal of Research and Development*, 26(1):100–105, Jan 1982.

[14] D. Hocevar, P. Yang, T. Trick, and B. Epler. Transient sensitivity computation for mosfet circuits. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 4(4):609, 1985.

[15] K. Hoffman and R. Kunze. *Linear algebra*. Prentice-Hall mathematics series. Prentice-Hall, 1971.

[16] S. Hooshangi, S. Thiberge, and R. Weiss. Ultrasensitivity and noise propagation in a synthetic transcriptional cascade. *Proc Natl Acad Sci U S A*, 102(10):3581–3586, Mar. 2005.

[17] Z. Ilievski, H. Xu, A. Verhoeven, E. Maten, W. Schilders, and R. Mattheij. Adjoint transient sensitivity analysis in circuit simulation. *Scientific Computing in Electrical Engineering*, 2006.

[18] T. Lin and C. Mead. Signal delay in general rc networks. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 3(4):331–349, October 1984.

[19] N. Menezes, R. Baldick, and L. Pileggi. A sequential quadratic programming approach to concurrent gate and wire sizing. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 16(8):867–881, Aug 1997.

[20] N. Menezes, S. Pullela, F. Dartu, and L. Pillage. Rc interconnect synthesis-a moment fitting approach. In *Computer-Aided Design, 1994., IEEE/ACM International Conference on*, pages 418–425, Nov 1994.

[21] S. Nazarian, M. Pedram, E. Tuncer, and T. Lin. Sensitivity-based gate delay propagation in static timing analysis. In *Quality of Electronic Design, 2005. ISQED 2005. Sixth International Symposium on*, pages 536–541, March 2005.

[22] R. Neidinger. Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming. *SIAM Review*, 52(3):545–563, 2010.

[23] A. Odabasioglu, M. Celik, and L. Pileggi. Prima: passive reduced-order interconnect macromodeling algorithm. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 17(8):645–654, Aug 1998.

[24] J. Ousterhout. Crystal: A timing analyzer for nmos vlsi circuits. Technical Report UCB/CSD-83-119, EECS Department, University of California, Berkeley, Jan 1983.

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

[26] P. Purnick and R. Weiss. The second wave of synthetic biology: from modules to systems. *Nature Reviews Molecular Cell Biology*, 10(6):410–422, 2009.

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

[28] H. Rutishauser. *Vorlesungen ber numerische Mathematik*. Birkhuser-Verlag, 1976.

Supplementary Material

S1. PERFORMANCE RESULTS

BLAST performance data is presented in Fig. S1.1 where the sensitivity computation runtime is plotted vs. the number of integration points T. For comparison, we present the runtimes of the direct sensitivity and the adjoint sensitivity methods without applying the BLI technique (marked as $Adjoint_{Full}$ in the plots). The speedup factor depends on the choice of the parameters n, n_p and T, and for coherence of presentation, we report performance measurements in Table (S1.1) for a fixed integration interval and varying n and n_p. From the performed experiments, BLAST obtains an average speedup factor of **29.81** when compared to direct sensitivity computation under the same configuration, and **17.24** speedup factor when compared to the adjoint sensitivity method without BLI. A certain pattern can be observed from these measurements, and was confirmed by additional experiments: as we fix the integration interval and make our system larger (by increasing n and n_p), the speedup factor achieved over the direct sensitivity method grows. At the same time, if we compare the speedup factor to the adjoint method that uses no BLI, we see that as the size of the system dominates the length of the integration interval T, the speedup becomes smaller.

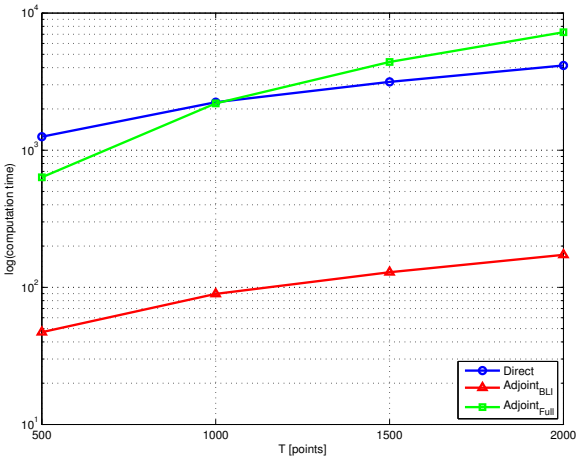

Figure S1.1: Sensitivity map computation time: Direct,Adjoint, Adjoint with BLI

n	n_p	T	$Direct$	$Adjoint_{BLI}$	$Adjoint_{Full}$
1	8	1000	2237.76	89.57	2195.07
2	16	1000	4127.92	145.82	2739.87
4	32	1000	8456.16	268.83	4107.86
8	64	1000	17865.36	517.92	6882.84

Table S1.1: Performance measurements (runtime in seconds)

To use BLAST, one has to choose a value of p, the interpolation order parameter. As usual, this choice represents a trade-off between accuracy and computation time. To test whether the choice of p can become the accuracy bottleneck of the system, we have measured the relative error as a function of the interpolation order p. We present our accuracy measurements for both equispaced and Chebyshev nodes in Fig. S1.2. It can be seen, that for $p = 20$ with Chebyshev nodes, we effectively reach double precision, which means that for a large system (large n and n_p) the factor introduced by p in the time complexity, becomes insignificant, and BLAST effectively performs at linear time. This means that the optimal choice of p will have no effect on the accuracy of the system, as the accuracy bottleneck will be somewhere else.

For completeness, we present the error in the sensitivity computed by BLAST in Fig. S1.3 and compare it to the accuracy of the sensitivity map produced by the direct method.

S2. ADJOINT OPERATOR FOR LINEAR DIFFERENTIAL EQUATIONS

Given an operator $\mathcal{L}$ taking its input from some domain $\mathcal{D}$ into the output range $\mathcal{R}$, the adjoint $\mathcal{L}^\dagger$ of the operator $\mathcal{L}$ can be intuited as an inner-

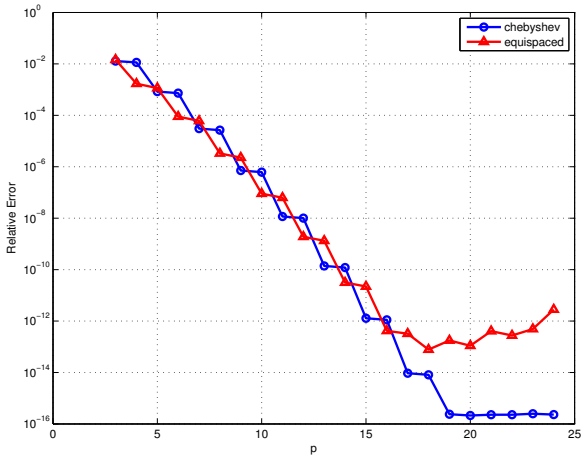

Figure S1.2: Approximation error vs. interpolation order tested on $f(t) = e^{-t}sin(2\pi ft)$

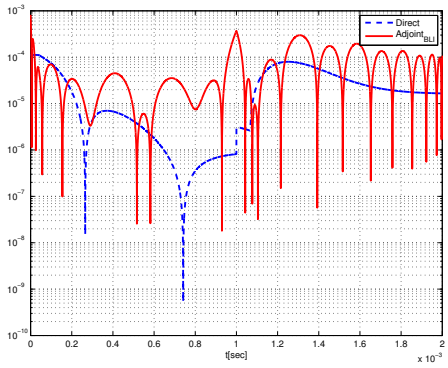

Figure S1.3: Error vs time for direct and adjoint sensitivities

product preserving operator. For a fixed point y in the range, we operate $\mathcal{L}$ on any u in the domain $\mathcal{D}$, and take an inner product between the image of u, i.e. $\mathcal{L}(u)$, and the fixed point y :see Fig. S2.1 for illustration. The adjoint operator, denoted $\mathcal{L}^\dagger$, operating on y in the range, will return z in the domain $\mathcal{D}$, whose inner product with the original u is the same, as the inner product of $\mathcal{L}(u)$ and the fixed point y. If for any choice of the fixed point y, we can find such a z, we say that the mapping $y \mapsto z$ defines the adjoint operator $\mathcal{L}^\dagger$.

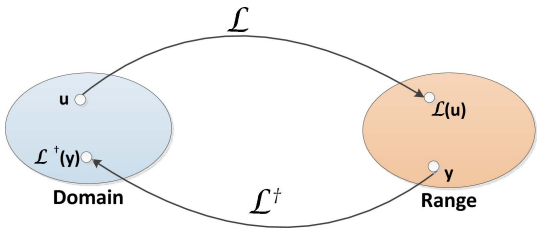

Figure S2.1: Schematic representation of the adjoint operator.

More formally, $\mathcal{L}^\dagger$ would be defined as the adjoint operator of $\mathcal{L}$, if $\forall u \in \mathcal{D}, \forall y \in \mathcal{R}$, we have:

$$\langle \mathcal{L}\{u\}, y\rangle_{\mathcal{R}} = \langle u, \mathcal{L}^\dagger\{y\}\rangle_{\mathcal{D}}. \tag{41}$$

Equation (12) which is rewritten here for convenience as

$$-C^*(t)\frac{d}{dt}\vec{z}(t) + G^*(t)\vec{z}(t) = \vec{y}(t), \tag{42}$$

defines a mapping $\vec{y} \mapsto \vec{z}$, *i.e.* between $\vec{y}(t) \in \mathcal{R}$ and $\vec{z}(t) \in \mathcal{D}$. We will show that this mapping is the adjoint of the mapping $\mathcal{L}$, defined by equation (10), rewritten here for convenience as

$$\frac{d}{dt}\boldsymbol{C}(t)\,\vec{x}(t) + \boldsymbol{G}(t)\,\vec{x}(t) = \vec{u}(t). \tag{43}$$

To show that $\vec{z} = \mathcal{L}^\dagger\{\vec{y}\}$, we need to show, that following the definition in

(41), the inner product is preserved, *i.e.*,

$$\begin{aligned}\forall \vec{u} \in \mathcal{D}, \vec{y} \in \mathcal{R} \text{ s.t. } \mathcal{L}\{\vec{u}\} = \vec{x} \text{ and } \mathcal{L}^\dagger\{\vec{y}\} = \vec{z}, \\ \langle \vec{x}(t), \vec{y}(t) \rangle_{\mathcal{R}} = \langle \vec{u}(t), \vec{z}(t) \rangle_{\mathcal{D}}\,.\end{aligned} \tag{44}$$

We use the standard definition of inner product for vector functions:

$$\langle \vec{x}(t), \vec{y}(t) \rangle = \int_A^B w(\tau)\, [\vec{y}^*(\tau)\, \vec{x}(\tau)]\, d\tau, \tag{45}$$

where $w(t)$ is some appropriate scalar weighting function. We assume $w(t) \equiv 1$ in this work.
We note that

$$\begin{aligned}\langle \vec{x}(t), \vec{y}(t) \rangle_{\mathcal{R}} &= \left\langle \vec{x}(t), -\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\, \vec{z}(t) \right\rangle_{\mathcal{R}} \\ &= \left\langle \vec{x}(t), -\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) \right\rangle_{\mathcal{R}} + \langle \vec{x}(t), \boldsymbol{G}^*(t)\, \vec{z}(t) \rangle_{\mathcal{R}}\,,\end{aligned} \tag{46}$$

and also that

$$\begin{aligned}\langle \vec{u}(t), \vec{z}(t) \rangle_{\mathcal{D}} &= \left\langle \frac{d}{dt}\left[\boldsymbol{C}(t)\, \vec{x}(t)\right] + \boldsymbol{G}(t)\, \vec{x}(t), \vec{z}(t) \right\rangle_{\mathcal{D}} \\ &= \left\langle \frac{d}{dt}\left[\boldsymbol{C}(t)\, \vec{x}(t)\right], \vec{z}(t) \right\rangle_{\mathcal{D}} + \langle \boldsymbol{G}(t)\, \vec{x}(t), \vec{z}(t) \rangle_{\mathcal{D}}\,.\end{aligned} \tag{47}$$

Our purpose is to show that the quantities (46) and (47) are equal. From inner product properties, it is obvious that the second terms are equal. We will show that the first terms are also equal, *i.e.*, that

$$\left\langle \vec{x}(t), -\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) \right\rangle_{\mathcal{R}} = \left\langle \frac{d}{dt}\left[\boldsymbol{C}(t)\, \vec{x}(t)\right], \vec{z}(t) \right\rangle_{\mathcal{D}}\,.$$

To do this, we use (45) to expand out the inner products and get

$$\left\langle \vec{x}(t), -\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) \right\rangle_{\mathcal{R}} = \int_A^B -\left[\frac{d}{dt}\vec{z}^*(t)\right]\, \boldsymbol{C}(t)\, \vec{x}(t)\, dt.$$

Since integration by parts can be stated as

$$\int_A^B \frac{dr}{dt} s(t)\, dt = [r(t)\, s(t)]_A^B - \int_A^B r(t)\frac{ds}{dt}\, dt. \tag{48}$$

we set $s(t) \equiv -\boldsymbol{C}(t)\vec{x}(t)$ and $\vec{r}(t) \equiv \vec{z}^*(t)$, to arrive at:

$$\begin{aligned}\left\langle \vec{x}(t), -\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) \right\rangle_{\mathcal{R}} &= \int_A^B -\left[\frac{d}{dt}\vec{z}^*(t)\right]\, \boldsymbol{C}(t)\, \vec{x}(t)\, dt \\ &= \int_A^B \frac{dr(t)}{dt} s(t)\, dt \\ &= [r(t)\, s(t)]_A^B - \int_A^B r(t)\, \frac{ds}{dt}\, dt \\ &= [-\vec{z}^*(t)\boldsymbol{C}(t)\vec{x}(t)]_A^B + \int_A^B \vec{z}^*(t)\frac{d}{dt}\left[\boldsymbol{C}(t)\vec{x}(t)\right]\, dt \\ &= [-\vec{z}^*(t)\boldsymbol{C}(t)\vec{x}(t)]_A^B + \left\langle \frac{d}{dt}\left[C(t)\vec{x}(t)\right], \vec{z}(t) \right\rangle_{\mathcal{D}}.\end{aligned}$$

To complete our proof, we are left with the task of making the term $[-\vec{z}^*(t)\boldsymbol{C}(t)\vec{x}(t)]_A^B$, zero. This term, which we will call the "boundary effect" term, leads to challenges in many situations having to do with adjoints. One way to deal with it, is to make $A \to -\infty$, $B \to \infty$, and restrict the functions $\vec{x}(t)$, $\vec{z}(t)$, to be zero at $\pm\infty$. This is a viable practical solution to this problem for stable systems of differential equations, since we can usually choose inputs that go to zero far away from the regions of time we are interested in. If we cannot (*i.e.*, the system exhibits non-zero asymptotic behaviour), then other technical tricks – like adding weight functions to the inner-product definition, and restricting the domain of definition of the adjoint operator – can be tried to zero out boundary effect terms. In applications, one should be careful to check that such boundary terms are indeed zero, or if non-zero, that they are properly accounted for in the overall analysis.
In our situation, however, getting around this is particularly simple. Since both $\vec{x}(t)$ and $\vec{z}(t)$ are solutions of differential equations, we do need to specify initial conditions. We choose the initial conditions to be $\vec{x}(A) = 0$ and $\vec{z}(B) = 0$. This choice ensures that $[-\vec{z}^*(t)\boldsymbol{C}(t)\vec{x}(t)]_A^B = 0$. As we will see later, the adjoint differential equation (42) will need to be solved *backwards*, from B to A, so an "initial" condition on $\vec{z}(B)$ is a natural one for the adjoint.

S3. LOCAL SENSITIVITY USING THE ADJOINT OPERATOR

In section §2.2 we have used the linearized system (11), rewritten here for convenience as

$$\frac{d}{dt}\boldsymbol{C}(t)\, \Delta\vec{x}(t) + \boldsymbol{G}(t)\, \Delta\vec{x}(t) = \vec{u}(t) = -\boldsymbol{S}(t)\Delta\vec{p}, \tag{49}$$

and its adjoint system, rewritten here for convenience as

$$-C^*(t)\frac{d}{dt}\vec{z}(t) + G^*(t)\vec{z}(t) = \vec{y}(t), \tag{50}$$

to arrive at the sensitivity vector of a specific output $d(t)$ at a given time point $t = T_0$.
While most technicalities were omitted from the main article, we now provide the detailed derivation, arriving at equation (57), which lays the foundation for Algorithm 1 (*Local Sensitivity Algorithm*).
The main feature of the adjoint operator relevant for our purpose, is the preservation of inner product. We will utilize this property to reduce the computational burden of sensitivity analysis. In this work, we use the standard definition of inner product for vector functions (45) rewritten here for convenience:

$$\langle \vec{x}(t), \vec{y}(t) \rangle = \int_A^B w(\tau)\, [\vec{y}^*(\tau)\, \vec{x}(\tau)]\, d\tau. \tag{51}$$

In (51), $w(t)$ is some scalar weighting function, which we define to be $w(t) \equiv 1$.
As mentioned in section §S2 of supplementary material, inner product preservation simply means, that for a linear operator $\mathcal{L} : \vec{u}(t) \mapsto \Delta\vec{x}(t)$, and its adjoint operator $\mathcal{L}^\dagger : \vec{y}(t) \mapsto \vec{z}(t)$ the following holds:

$$\begin{aligned}\forall \vec{u} \in \mathcal{D}, \vec{y} \in \mathcal{R} \text{ s.t. } \mathcal{L}\{\vec{u}\} = \Delta\vec{x} \text{ and } \mathcal{L}^\dagger\{\vec{y}\} = \vec{z}, \\ \langle \Delta\vec{x}, \vec{y} \rangle_{\mathcal{R}} = \langle \mathcal{L}\{\vec{u}\}, \vec{y} \rangle_{\mathcal{R}} = \langle \vec{u}, \mathcal{L}^\dagger\{\vec{y}\} \rangle_{\mathcal{D}} = \langle \vec{u}, \vec{z} \rangle_{\mathcal{D}}.\end{aligned} \tag{52}$$

To demonstrate the utility of (52) let us assume that we are interested in a single output $x_1(t)$. Let us further assume that we are interested in sensitivity of $x_1(t)$ at a single time point T_0. Denoting an elementary vector as $\vec{e}_1$, and applying the basic property of the inner product we get:

$$\Delta x_1(T_0) = \langle \Delta\vec{x}(t), \vec{e}_1\delta(t - T_0) \rangle\,, \tag{53}$$

which now allows us to apply (52) and arrive at:

$$\begin{aligned}&\langle \Delta\vec{x}(t), \vec{e}_1\delta(t - T_0) \rangle = \\ &\langle \mathcal{L}\{\vec{u}(t)\}, \vec{e}_1\delta(t - T_0) \rangle_{\mathcal{R}} = \left\langle \vec{u}(t), \underbrace{\mathcal{L}^\dagger\{\vec{e}_1\delta(t - T_0)\}}_{\vec{z}_{T_0}(t)} \right\rangle_{\mathcal{D}}.\end{aligned} \tag{54}$$

Note that the $\mathcal{L}$ represents the linear operator defined by (49), and $\vec{z}_{T_0}(t)$ is the solution of the differential equation defined by the adjoint operator $\mathcal{L}^\dagger : \vec{y}(t) \mapsto \vec{z}(t)$ for an input $\vec{y}(t) = \vec{e}_1\delta(t - T_0)$. From (49) we observe that the input $\vec{u}(t)$ to our forward equation is $\vec{u}(t) = -\boldsymbol{S}(t)\Delta\vec{p}$, allowing us to rewrite (54), and using (53) arrive at:

$$\Delta x_1(T_0) = \left\langle \vec{u}(t), \underbrace{\mathcal{L}^\dagger\{\vec{e}_1\delta(t - T_0)\}}_{\vec{z}_{T_0}(t)} \right\rangle_{\mathcal{D}} = \langle -\boldsymbol{S}(t)\Delta\vec{p}, \vec{z}_{T_0}(t) \rangle_{\mathcal{D}}\,. \tag{55}$$

Finally we can apply the inner product definition from (51) and get:

$$\Delta x_1(T_0) = \langle -\boldsymbol{S}(t)\Delta\vec{p}, \vec{z}_{T_0}(t) \rangle_{\mathcal{D}} = -(\int_A^B \vec{z}^*_{T_0}(\tau)\, \boldsymbol{S}(\tau)d\tau)\Delta\vec{p}. \tag{56}$$

Performing a "division" operation similar to the one in (7), we arrive at $\vec{m}_1(T_0)$, defined as the sensitivity vector of $x_1(t)$ with respect to parameters $\vec{p}$, evaluated at time $t = T_0$:

$$\vec{m}_1(T_0) = \frac{\Delta x_1(T_0)}{\Delta\vec{p}} = -\int_A^B \vec{z}^*_{T_0}(\tau)\, \boldsymbol{S}(\tau)d\tau. \tag{57}$$

To compute $\vec{m}_1(T_0)$ we first need to obtain $\vec{z}_{T_0}(t)$ by solving the adjoint system for an input $\vec{u}(t) = \vec{e}_1\delta(t - T_0)$, and then compute the integral in (57). Note that the matrices $\boldsymbol{S}(t)$ can be readily computed from (8).

A subtle, yet important point concerning the integration limits in the definition of inner product(A and B), is in order. As outlined in section §S2 of the supplementary material, we need to zero out the boundary effect term $[-\vec{z}^*(t)\boldsymbol{C}(t)\Delta\vec{x}(t)]_A^B$. Choosing $A = 0$, will set the term $[-\vec{z}^*(t)\boldsymbol{C}(t)\Delta\vec{x}(t)]_A$ to zero, since the initial condition of (49), given at $t = 0$, is $\vec{0}$.

As for B, we know from the definition of the inner product it follows that we must have $B \geq T_0$. Moreover, for reasons elucidated in section §S4 of the supplementary material, we choose $B > T_0$. To zero out the term $[-\vec{z}^*(t)\boldsymbol{C}(t)\Delta\vec{x}(t)]^B$, we must impose the the "initial" condition $\vec{z}(B) = \vec{0}$ when solving the adjoint DAE (50); to obtain $\vec{z}(t)$ over the interval $[A, B]$ (needed for computing (57)), the adjoint DAE must be solved backwards from B to A. In this context, note that if (49) is a stable system (as will typically be the case in applications), then (50) is an unstable system — in other words, its solution will blow up exponentially as t increases. Hence, solving it backwards is *necessary*, from a numerical standpoint, to avoid uncontrolled error blowup; since (12) is unstable going forward in time, it is stable going backward in time, hence standard numerical integration methods can be applied directly.

S4. ANALYSIS OF ADJOINT ODE WITH δ-FUNCTION INPUT

In section §2.2 of our work, we show that a key step in the computation of transient adjoint sensitivities, is the solution of the adjoint equation (14), rewritten here for convenience:

$$-\boldsymbol{C}^*(t)\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{c}\;\delta(t - T_0). \tag{58}$$

As mentioned before, we need to solve (58) backwards from $t = B$ to $t = 0$, with "initial" condition $\vec{z}(B) = \vec{0}$, and $B > T_0$.

However, this solution cannot immediately be obtained using standard numerical integration methods, because the input is a δ-function. Numerical methods are not well suited for δ-function inputs, which involve an infinite value at a single time-point, while being identically zero elsewhere. Therefore, further analysis is needed to re-phrase (58) in a form suitable for numerical integration.

To simplify this analysis, we now restrict ourselves to ODEs: *i.e.*, $\vec{q}(\vec{x}) \equiv \vec{x}$ in (2), leading (*w.l.o.g.* for invertible $\boldsymbol{C}$) to $\boldsymbol{C}(t) \equiv I_{n\times n}$. For the ODE case, (58) becomes

$$-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{c}\;\delta(t - T_0), \tag{59}$$

to be solved backwards from $t = B$ to $t = 0$, with "initial" condition given as $\vec{z}(B) = \vec{0}$.

The δ-function form of the input (59), together with its zero initial condition at $t = B$, has the following implications as we integrate backwards from $t = B$ to $t = 0$ (note that $B > T_0$, by assumption above):

1. Over the interval $[T_0^+, B]$, the solution of (59), $\vec{z}(t)$, is identically the zero vector. This is because (59), a linear ODE, has an identically zero input over this interval, as well as an initial condition of zero, at $t = B$.

2. Over the interval $[T_0^-, T_0^+]$, the δ-function in the input is "active". Integrating (59) over this interval, we obtain

$$\begin{aligned}
&\int_{T_0^-}^{T_0^+}\left[-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t)\right]dt = \int_{T_0^-}^{T_0^+}\vec{c}\;\delta(t - T_0)\;dt\\
\Rightarrow &\int_{T_0^-}^{T_0^+}\left[-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t)\right]dt = \vec{c}\\
\Rightarrow &-[\vec{z}(t)]_{T_0^-}^{T_0^+} + \int_{T_0^-}^{T_0^+}\boldsymbol{G}^*(t)\,\vec{z}(t)\;dt = \vec{c}\\
\Rightarrow &\vec{z}(T_0^-) - \vec{z}(T_0^+) + \int_{T_0^-}^{T_0^+}\boldsymbol{G}^*(t)\,\vec{z}(t)\;dt = \vec{c}\\
\Rightarrow &\vec{z}(T_0^-) = \vec{z}(T_0^+) - \int_{T_0^-}^{T_0^+}\boldsymbol{G}^*(t)\,\vec{z}(t)\;dt + \vec{c}.
\end{aligned} \tag{60}$$

Note that

- $\vec{z}(T_0^+) \equiv \vec{0}$ (we just established this in 1, above), and
- $\int_{T_0^-}^{T_0^+}\boldsymbol{G}^*(t)\,\vec{z}(t)\;dt \equiv \vec{0}$ (we are integrating finite quantities over an infinitesimally small interval).

Hence (60) becomes

$$\vec{z}(T_0^-) = \vec{c}. \tag{61}$$

3. Over the interval $[0, T_0^-]$, the input to (59) is again identically zero since the δ-function is not active. However, the "initial" condition over this interval is given by (61), *i.e.*, $\vec{z}(T_0^-) = \vec{c}$.

From these observations, we see that finding the solution of (59) over the interval $[0, T_0^-]$, is equivalent to solving the homogeneous part of (59) (*i.e.*, with zero input), *i.e.*,

$$-\frac{d}{dt}\vec{z}(t) + \boldsymbol{G}^*(t)\,\vec{z}(t) = \vec{0}, \tag{62}$$

with initial condition $\vec{z}(T_0^-) = \vec{0}$, backwards from $t = T_0^-$ to $t = 0$. Since T_0^- is only infinitesimally separated from T, we can replace T_0^- with T, since no infinite values or δ-functions arise in (62).

S5. ELMORE DELAY SENSITIVITY

For a system with step response $g_A(t)$ at a node A, we have proposed an Elmore-delay like metric in section §4 of our work. The metric $T_{d_{NL}}(A)$ was defined as

$$T_{d_{NL}}(A) = \frac{\int_0^\infty g'_A(t)\cdot t\;dt}{\int_0^\infty g'_A(t)\;dt}. \tag{63}$$

Assuming that we operate in a finite integration interval $[0, T]$, we can rewrite and simplify (63) as

$$T_{d_{NL}}(A) = \frac{\int_0^T g'_A(t)\cdot t\;dt}{\int_0^T g'_A(t)\;dt} = \frac{\int_0^T g'_A(t)\cdot t\;dt}{g_A(T) - g_A(0)}. \tag{64}$$

We can now write the expression for the delay sensitivity vector:

$$\begin{aligned}
\frac{d}{d\vec{p}}T_{d_{NL}}(A) =& \frac{\frac{d}{d\vec{p}}\big(\int_0^T g'_A(t)\cdot t\;dt\big)\cdot\big[g_A(T) - g_A(0)\big]}{\left(\frac{d}{d\vec{p}}\big[g_A(T) - g_A(0)\big]\right)^2}\\
&- \frac{\big(\int_0^T g'_A(t)\cdot t\;dt\big)\cdot\frac{d}{d\vec{p}}\big[g_A(T) - g_A(0)\big]}{\left(\frac{d}{d\vec{p}}\big[g_A(T) - g_A(0)\big]\right)^2}\\
=& \frac{\big(\int_0^T \frac{dg'_A}{d\vec{p}}(t)\cdot t\;dt\big)\cdot\big[g_A(T) - g_A(0)\big]}{D^2}\\
&- \frac{\big(\int_0^T g'_A(t)\cdot t\;dt\big)\cdot\frac{d}{d\vec{p}}\big[g_A(T) - g_A(0)\big]}{D^2},
\end{aligned} \tag{65}$$

where D^2 denotes the denominator for notational simplicity.

For the purpose of computing (65) numerically, assume we are using the set of time samples

$$\mathcal{T}_T \triangleq \{\tau_0, \tau_1, \cdots, \tau_{N_T}\}, \tag{66}$$

spanning the interval $[0, T]$. Approximating the integral by finite summation, and the temportal derivatives by finite backward differences, we are ready to write the discrete approximation expression for the delay sensitivity vector, but first another notational simplification. Let us assign the aliases α and β, to the first and second terms, in (65), respectively, and rewrite (65) as:

$$\frac{d}{d\vec{p}} T_{d_{NL}}(A) = \frac{\alpha - \beta}{D^2}. \tag{67}$$

Expanding the different terms we arrive at:

$$\begin{aligned}
\alpha &= \left[\sum_{i=1}^{N_T} \left(\frac{dg_A}{d\vec{p}}(\tau_i) - \frac{dg_A}{d\vec{p}}(\tau_{i-1})\right) \cdot \bar{\tau}_i \cdot \Delta_i\right] \cdot \left(g_A(\tau_{N_T}) - g_A(\tau_0)\right), \\
\beta &= \left(\frac{dg_A}{d\vec{p}}(\tau_{N_T}) - \frac{dg_A}{d\vec{p}}(\tau_0)\right) \cdot \left[\sum_{i=1}^{N_T} \left(g_A(\tau_i) - g_A(\tau_{i-1})\right) \cdot \bar{\tau}_i \cdot \Delta_i\right], \\
D^2 &= \left(\frac{dg_A}{d\vec{p}}(\tau_{N_T}) - \frac{dg_A}{d\vec{p}}(\tau_0)\right)^2,
\end{aligned} \tag{68}$$

where $\Delta_i \triangleq \tau_{i+1} - \tau_i$, and $\bar{\tau}_i \triangleq \frac{(\tau_i + \tau_{i-1})}{2}$. Note that $\frac{dg_A}{d\vec{p}}(\tau_i)$ is exactly the transient sensitivity vector, we have computed using BLAST.

To summarize, in this section we have shown a possible application of BLAST, in order to compute the delay sensitivity vector $\frac{dT_{d_{NL}}}{d\vec{p}}(A)$. While we have utilized a particular, Elmore-delay like metric, a similar technique can be used to leverage BLAST in any delay metric, which uses multiple timepoints on the waveform to compute the delay.