Time-Domain Analysis of Large-Scale Circuits by Matrix Exponential Method With Adaptive Control

Shih-Hung Weng, *Student Member, IEEE,* Quan Chen, *Member, IEEE,* and Chung-Kuan Cheng, *Fellow, IEEE*

***Abstract*—We propose an explicit numerical integration method based on matrix exponential operator for transient analysis of large-scale circuits. Solving the differential equation analytically, the limiting factor of maximum time step changes largely from the stability and Taylor truncation error to the error in computing the matrix exponential operator. We utilize Krylov subspace projection to reduce the computation complexity of matrix exponential operator. We also devise a prediction-correction scheme tailored for the matrix exponential approach to dynamically adjust the step size and the order of Krylov subspace approximation. Numerical experiments show the advantages of the proposed method compared with the implicit trapezoidal method.**

***Index Terms*—Adaptive time step, matrix exponential, transient simulation.**

I. Introduction

THE TREND toward ever higher integration density of very large-scale integration has made the time-domain circuit simulation [1] a bottleneck in today's integrated circuit design flows. For designs with millions of elements, SPICE-like simulation [1] can easily take days, even weeks, to complete the task. Therefore, accurate yet fast circuit simulation methods have always been one of the major demands in industry. Many efforts have been made to speed up the transient circuit simulation by improved integration algorithms [2]–[11].

Time-domain circuit simulation involves solving a system of ordinary differential equations (ODEs), which describes the behavior of circuit and can be solved numerically in either implicit or explicit way. The ODE system of circuit is usually stiff since the magnitude of elements in a circuit varies in a wide range. For example, capacitance usually ranges from 10^{-16} to 10^{-12}. Most of SPICE-like simulators adopt implicit methods, e.g., backward Euler and trapezoidal methods (TRAP), to overcome the stability problem of stiff ODE system. However, implicit methods are required to solve a linear system at each time step and hence increase computation time of circuit simulation. Despite the impressive achievements in iterative matrix solving methods [12], the ill-conditioning matrix resulting from implicit methods significantly degrades the performance of the iterative techniques. On the other hand, although direct technique [13] can be applied on any matrix, empirically, both computation and memory complexity are about $O(n^{1.5})$, where n is the number of unknowns. The limitation of performance and memory are the major problems of implicit methods for circuit simulation.

In contrast, for each time step, explicit methods, in general, need only sparse matrix-vector multiplication whose computation and memory complexity are $O(n)$. Nevertheless, the stability issue of explicit methods for stiff ODE enforces the use of smaller time steps when simulating a circuit, and the benefits of sparse matrix-vector product are damaged.

Apart from the above numerical methods, we can solve an ODE system in a semianalytical way, where the time span is still discretized but within each time step the equation is solved analytically by the matrix exponential operator $e^{\mathbf{A}}, \mathbf{A} \in \mathbb{C}^{N\times N}$. This leads to a distinct class of numerical approach for differential equations called exponential time differencing (ETD), which is dated back to 1960s [14] and has been "reinvented" over the years in some areas, such as computational physics [15], [16] and chemistry [17]. Solving differential equations analytically removes the local truncation error (LTE) of polynomial expansion approximation in most numerical methods, and the stability of ETD is as the same as TRAP, which is *A-stable* for passive circuits. Therefore, the step size of ETD is free from restrictions of the stability and the LTE of polynomial expansion.

The core computation of ETD lies in calculating the matrix exponential. Moler and Van Loan summarized 19 ways of computing a matrix exponential in their classic work [18], all of which are considered costly and thus limit the usage of ETD in time-domain simulation for circuits with huge size. In recent years, the Krylov subspace method has been introduced as the 20th way and enables an efficient evaluation of the product of $e^{\mathbf{A}}\mathbf{v}$ for very large-scale matrix $\mathbf{A}$ [18].

In this paper, we adapt the idea of ETD into the context of time-domain circuit simulation and develop an explicit time-marching scheme called matrix exponential method (MEXP) [19], [20]. Our method directly computes the analytical solution of the differential equations resulting from modified nodal analysis (MNA) [21]. We utilize the Krylov subspace method [22], [23] to approximate the matrix exponential operator, which significantly reduces the complexity of matrix

Manuscript received August 22, 2011; revised November 25, 2011 and February 8, 2012; accepted February 22, 2012. Date of current version July 18, 2012. This work was supported by the National Science Foundation, under Project CCF-1017864. This paper was recommended by Associate Editor P. Li.

S.-H. Weng and C.-K. Cheng are with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla, CA 92093 USA (e-mail: s2weng@ucsd.edu; ckcheng@ucsd.edu).

Q. Cheng is with the Department of Electrical and Electronic Engineering, University of Hong Kong, Pokfulam, Hong Kong (e-mail: q1chen@ucsd.edu).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2012.2189396

exponential computation. The largest step size of our method is generally limited by the nonlinearity of nonlinear devices and also approximation error of the matrix exponential operator.

Furthermore, the matrix exponential formulation together with the Krylov subspace method has some special properties that can be exploited to develop more efficient dynamic time step control than that currently used in SPICE-like simulators based on implicit methods. The properties include the scaling invariance of Krylov subspace projection and a convenient error estimate in computing the matrix exponential. Utilizing these properties, a prediction–correction scheme is designed in this paper for adaptive control of the step size and the order of Krylov subspace approximation during the numerical integration.

The remainder of this paper is organized as follows. Section II introduces the general matrix exponential formulation for linear and nonlinear circuits. Section III presents the technique to regularize the singular matrix in the matrix exponential formulation. Section IV presents the efficient computation of matrix exponential operator by Krylov subspace approximation. Section V discusses the adaptive control for MEXP. Section VI shows the experimental results and Section VII concludes this paper.

II. Matrix Exponential Formulation

In this section, we present the primary formulation of MEXP in linear and nonlinear circuit simulation. In MNA, a linear circuit is represented by a system of linear differential algebraic equations (DAEs) as follows:

$$\mathbf{C}\dot{\mathbf{x}}(t) = -\mathbf{G}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t) \tag{1}$$

where $\mathbf{C}$ is the capacitance and inductance matrix, $\mathbf{G}$ describes the conductance and the incidence between voltages and currents, and $\mathbf{B}$ is the input matrix of independent sources. Vector $\mathbf{x}(t)$ contains nodal voltages and certain branch currents at time t, and $\mathbf{u}(t)$ denotes the input voltage and current sources. Provided that $\mathbf{C}$ is invertible, (1) is reducible to a system of ODEs. Given an initial condition $\mathbf{x}(0)$ of the circuit (e.g., from dc analysis), one can obtain the analytical solution [24] of (1) as

$$\mathbf{x}(t) = e^{\mathbf{A}(t)}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(\mathbf{t}-\tau)}\mathbf{b}(\tau)d\tau$$

where $\mathbf{A} = -\mathbf{C}^{-1}\mathbf{G}$ (we do not need to explicitly compute $\mathbf{C}^{-1}\mathbf{G}$ as will be shown in Section IV) and $\mathbf{b}(t) = \mathbf{C}^{-1}\mathbf{B}\mathbf{u}(t)$. Given the solution at time t and a time step h, the solution at $t+h$ is

$$\mathbf{x}(t+h) = e^{\mathbf{A}h}\mathbf{x}(t) + \int_0^h e^{\mathbf{A}(h-\tau)}\mathbf{b}(t+\tau)d\tau. \tag{2}$$

Following the convention of SPICE-like simulators, we assume that the given input $\mathbf{u}(t)$ is piecewise linear (PWL), i.e., $\mathbf{u}(t)$ is linear within every time step. The integral term in (2) can be computed analytically, turning (2) to (3) involving three functions with matrix exponential operators as follows:

$$\begin{aligned}\mathbf{x}(t+h) &= e^{\mathbf{A}h}\mathbf{x}(t) \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{b}(t) \\ &+ (e^{\mathbf{A}h} - (\mathbf{A}h + \mathbf{I}))\mathbf{A}^{-2}\frac{\mathbf{b}(t+h) - \mathbf{b}(t)}{h}.\end{aligned} \tag{3}$$

We call the solution scheme based on this formulation as MEXP. MEXP is an A-stable explicit method because $\mathbf{x}$ approaches zero as h tends to infinity when eigenvalues of $\mathbf{A}$ are negative.

Note that MEXP is an exact method in the sense that one can solve (1) analytically, provided that the matrix exponential is computed exactly and the PWL assumption of input is satisfied. This is of theoretical difference from linear multistep methods, such as the forward Euler method and the trapezoidal methods. These methods approximate (1) by polynomial expansion and drop high-order terms, which is the source of LTE. Therefore, for linear circuits, the step size of MEXP is not restricted by LTE or stability, but instead solely by the computation error of matrix exponential, which will be detailed in Section IV.

For nonlinear circuits, the differential equation is given by

$$\dot{\mathbf{q}}(\mathbf{x}(t)) + \mathbf{C}^l\dot{\mathbf{x}}(t) = -\left(\mathbf{G}^l\mathbf{x}(t) + \mathbf{i}(\mathbf{x}(t))\right) + \mathbf{B}\mathbf{u}(t) \tag{4}$$

where $\mathbf{C}^l$ and $\mathbf{G}^l$ describe the linear capacitances, inductances and conductances, and $\mathbf{q}(\mathbf{x}(\mathbf{t}))$ and $\mathbf{i}(\mathbf{x}(t))$ denote the nonlinear charges and currents from the nonlinear elements. With a mild approximation, we assume the nonlinear charge varies linearly within the time interval (t_n, t_n+h), which leads to a differential equation similar to (1) as follows:

$$\mathbf{C}_n\dot{\mathbf{x}}(t) = -\left(\mathbf{G}^l\mathbf{x}(t) + \mathbf{i}(\mathbf{x}(t))\right) + \mathbf{B}\mathbf{u}(t) \tag{5}$$

with $\mathbf{C}_n = \mathbf{C}^l + \mathbf{C}_n^{nl}$, where $\mathbf{C}_n^{nl}$ is the companion capacitance matrix for nonlinear devices evaluated at t_n. The exact solution of (5) is therefore in an analogous form with (2) as follows:

$$\begin{aligned}\mathbf{x}(t_n+h) &= e^{\mathbf{A}_n h}\mathbf{x}(t_n) \\ &+ \int_0^h e^{\mathbf{A}_n(h-\tau)}\left[\mathbf{F}(\mathbf{x}(t_n+\tau)) + \mathbf{b}(t_n+\tau)\right]d\tau\end{aligned} \tag{6}$$

where $\mathbf{A} = -\mathbf{C}_n^{-1}\mathbf{G}^l$ and $\mathbf{F}(\mathbf{x}(\tau)) = -\mathbf{C}_n^{-1}\mathbf{i}(\mathbf{x}(\tau))$.

The second-order implicit approximation of $\mathbf{F}(\tau) = (\mathbf{F}(\mathbf{x}_n) + \mathbf{F}(\mathbf{x}_{n+1}))/2$, which is A-stable [15], leads to an algebraic nonlinear system as follows:

$$\begin{aligned}\mathbf{x}_{n+1} &= \frac{(e^{\mathbf{A}h} - \mathbf{I})}{2}\mathbf{A}^{-1}\mathbf{F}(\mathbf{x}_{n+1}) + e^{\mathbf{A}h}\mathbf{x}_n \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\left(\frac{\mathbf{F}_n}{2} + \mathbf{b}_n\right) \\ &+ (e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I})\mathbf{A}^{-2}\Delta\mathbf{b}_n\end{aligned} \tag{7}$$

where $\Delta\mathbf{b}_n = \frac{\mathbf{b}_{n+1} - \mathbf{b}_n}{h}$. The nonlinear equation arising from the (implicit) approximation of $\mathbf{F}$ involves the product of a function of matrix exponential and the nonlinear function, which couples the responses from the linear elements and nonlinear elements in the circuit. Such coupled system is generally expensive to solve by standard iterative solvers, e.g., Newton's method or fixed point method, since the functions of matrix exponential need to be reevaluated by the Krylov subspace approximation in every iteration, as will be more obvious in Section IV.

To decouple the linear and nonlinear terms in the above equation, we adapt the scheme developed in [25], which approximates $e^{-\mathbf{A}_n(\tau)}\mathbf{F}(\mathbf{x}(t_n+\tau))$ instead of $\mathbf{F}(\mathbf{x}(t_n+\tau))$ by

a Lagrange polynomial in the temporal integral of (6). The second-order implicit approximation is then of the form

$$\begin{aligned}\mathbf{x}_{n+1} = &\frac{h}{2}\mathbf{F}\left(\mathbf{x}_{n+1}\right) + e^{\mathbf{A}h}\left(\mathbf{x}_n + \frac{h}{2}\mathbf{F}_n\right)\\ &+ \left(e^{\mathbf{A}h} - \mathbf{I}\right)\mathbf{A}^{-1}\mathbf{b}_n\\ &+ \left(e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I}\right)\mathbf{A}^{-2}\Delta\mathbf{b}_n\end{aligned} \tag{8}$$

with a LTE of

$$-\frac{h^3}{12}\left((\mathbf{A}h)^2\mathbf{F}_n + (\mathbf{A}h)\dot{\mathbf{F}}_n + \ddot{\mathbf{F}}_n\right). \tag{9}$$

In (8), the nonlinear function of $\mathbf{x}_{n+1}$ is only multiplied by a scalar coefficient, other than the matrix exponential function in (7). The remaining three matrix exponential functions involve only known quantities from previous time steps. This decoupling of nonlinearity and matrix exponential (essentially linearity) facilitates the numerical solution greatly, in that the time-consuming evaluation of matrix exponential is needed only once in each time step, while the nonlinear solution can iterate multiple times until convergence.

The nonlinear equation of (8) can be readily solved by Newton's method. Nevertheless, it is more convenient and consistent to solve it by standard fixed point solver as shown in [25], since the starting point of the matrix exponential framework is to replace solution of linear system with (efficient) computation of matrix exponential as follows:

$$\begin{aligned}\mathbf{x}_{n+1}^{k+1} = &\frac{h}{2}\mathbf{F}\left(\mathbf{x}_{n+1}^{k}\right) + e^{\mathbf{A}h}\left(\mathbf{x}_n + \frac{h}{2}\mathbf{F}_n\right)\\ &+ \left(e^{\mathbf{A}h} - \mathbf{I}\right)\mathbf{A}^{-1}\mathbf{b}_n\\ &+ \left(e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I}\right)\mathbf{A}^{-2}\Delta\mathbf{b}_n.\end{aligned} \tag{10}$$

To ensure convergence, one needs to maintain the following condition throughout the fixed point iterations:

$$\left\|\frac{h}{2}\frac{d\mathbf{F}\left(\mathbf{x}\right)}{d\mathbf{x}}\bigg|_{\mathbf{x}_{n+1}}\right\| < 1 \;\Rightarrow\; \left\|\frac{h}{2}\mathbf{C}_n^{-1}\mathbf{G}^{nl}\big|_{\mathbf{x}_{n+1}}\right\| < 1 \tag{11}$$

which can be roughly examined by the norms of $\mathbf{C}_n$ and $\mathbf{G}^{nl}$ at each step. The convergence condition imposes a restriction on the step size depending on the nonlinearity of the circuit for simulation. This restriction is generally mild compared with the restriction from accuracy requirement of numerical solution, e.g., (9). Newton's method can be resorted to for the cases with very strong nonlinearity. The error incurred by the constant capacitance approximation can be measured by the difference between the charge evaluated with the converged $\mathbf{x}_{n+1}$ and the charge from the presuming linear variation from $\mathbf{q}_n$, that is

$$\left\|\mathbf{C}_{n+1}^{nl}\mathbf{x}_{n+1} - \mathbf{C}_n^{nl}\mathbf{x}_{n+1}\right\| \tag{12}$$

where $\mathbf{C}^{nl}$ here refers to the capacitance part only.

The primary version of MEXP (3) has two limitations when applied in large-scale circuit simulation. First, the system of (1) has to be convertible to an ODE for which an analytical solution is available, i.e., $\mathbf{C}$ must be nonsingular. This is usually not the case with generic MNA formulation. The matrix $\mathbf{A}$, however, needs not to be nonsingular, since the three matrix exponential functions in their Taylor expansion form are just power series of $\mathbf{A}$. Second, direct computation of matrix exponential is prohibitive [18] as the dimension of matrices in modern circuit simulation easily exceeds one million. The two problems will be addressed in the following two sections.

III. Regularization

The most common cause of singular $\mathbf{C}$ matrix in (1) is the empty rows in $\mathbf{C}$ corresponding to the nodes without capacitance and the currents of independent and controlled sources, which have no time-differential terms appear in the equation. On top of this "explicit" singularity, it is often the case that some "hidden" dependency among variables would make $\mathbf{C}$ noninvertible or ill-conditioned even though it has no zero rows [24].

We have reported in a separate work [26] a two-phase regularization technique to construct from the original MNA system with singular $\mathbf{C}$ matrix an equivalent system with invertible $\mathbf{C}$, i.e., converting a descriptor system (DAE) to an explicit state-space system (ODE). A succinct review is given here. The first phase of regularization utilizes graph theory to analyze the network topology and reduce the DAE index of the MNA equation by eliminating certain elements via Gaussian elimination (GE). In the second phase, a systematic elimination process is applied to remove implicit dependency among variables, resulting to a nonsingular system. For clarity, the systematic elimination is presented first followed by the topological index reduction.

A. Systematic Elimination

We utilize the regularization flow developed by Natarajan [27], which reduces $\mathbf{C}$ to its row-echelon form as follows:

$$\begin{bmatrix}\mathbf{C}_{11} & \mathbf{C}_{12}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\dot{\mathbf{x}}_1\\ \dot{\mathbf{x}}_2\end{bmatrix} = -\begin{bmatrix}\mathbf{G}_{11} & \mathbf{G}_{12}\\ \mathbf{G}_{21} & \mathbf{G}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix} + \begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix}\mathbf{u}. \tag{13}$$

Then another row-echelon transform is applied to the submatrix of $[\mathbf{G}_{21}\ \mathbf{G}_{22}]$ from the bottom row. The columns of $\mathbf{G}$ (and $\mathbf{C}$) are rearranged to ensure $\mathbf{G}_{22}$ is lower triangular. Finally, block GE is applied to obtain a reduced system of equations as follows:

$$\mathbf{C}_r\dot{\mathbf{x}}_1 = -\mathbf{G}_r\mathbf{x}_1 + \mathbf{B}_{0r}\mathbf{u} + \mathbf{B}_{1r}\dot{\mathbf{u}} \tag{14}$$

where

$$\mathbf{C}_r = \mathbf{C}_{11} - \mathbf{C}_{12}\mathbf{G}_{22}^{-1}\mathbf{G}_{21}\quad \mathbf{G}_r = \mathbf{G}_{11} - \mathbf{G}_{12}\mathbf{G}_{22}^{-1}\mathbf{G}_{21} \tag{15a}$$

$$\mathbf{B}_{0r} = \mathbf{B}_1 - \mathbf{G}_{12}\mathbf{G}_{22}^{-1}\mathbf{B}_2\quad \mathbf{B}_{1r} = -\mathbf{C}_{12}\mathbf{G}_{22}^{-1}\mathbf{B}_2 \tag{15b}$$

$$\mathbf{x}_2 = -\mathbf{G}_{22}^{-1}\left(\mathbf{G}_{21}\mathbf{x}_1 - \mathbf{B}_2\mathbf{u}\right). \tag{15c}$$

Provided C_r is invertible, a variable transform of $\mathbf{x}_r = \mathbf{x}_1 - \mathbf{C}_r^{-1}\mathbf{B}_{1r}\mathbf{u}$ is applied to absorb the derivative of $\mathbf{u}$, rendering a regular ODE as in (1) as follows:

$$\mathbf{C}_r\dot{\mathbf{x}}_r = -\mathbf{G}_r\mathbf{x}_r + \mathbf{B}_r\mathbf{u} \tag{16}$$

with $\mathbf{B}_r = \mathbf{B}_{0r} - \mathbf{G}_r\mathbf{C}_r^{-1}\mathbf{B}_{1r}$.

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} G_{ii} & G_{ij} & 1 \\ G_{ji} & G_{jj} & -1 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i} \\ I_{s_j} \\ \hline V_s \end{array}\right] \tag{17a}$$

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline V_s \end{array}\right] \tag{17b}$$

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} 0 \\ 0 \\ \hline -\dot{V}_s \end{array}\right] \tag{17c}$$

$$\left[\begin{array}{cc|c} 0 & C_{ij}+C_{ii} & 0 \\ 0 & C_{jj}+C_{ji} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} -C_{ii}\dot{V}_s \\ -C_{ji}\dot{V}_s \\ \hline -\dot{V}_s \end{array}\right] \tag{17d}$$

$$\left[\begin{array}{cc|c} 0 & C_{ij}+C_{ii} & 0 \\ 0 & C_{jj}+C_{ji}+C_{ij}+C_{ii} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji}+G_{ij}+G_{ii} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s+I_{s_i}+G_{ii}V_s \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} -C_{ii}\dot{V}_s \\ -C_{ji}\dot{V}_s-C_{ii}\dot{V}_s \\ \hline -\dot{V}_s \end{array}\right] \tag{17e}$$

$$\left[\,C_{jj}+C_{ji}+C_{ij}+C_{ii}\,\right]\left[\,\dot{v}_j\,\right] = -\left[\,G_{jj}+G_{ji}+G_{ij}+G_{ii}\,\right]\left[\,v_j\,\right] + \left[\,I_{s_j}+G_{ji}V_s+I_{s_i}+G_{ii}V_s\,\right] + \left[\,-C_{ji}\dot{V}_s-C_{ii}\dot{V}_s.\,\right] \tag{17f}$$

The regularization of (13) has two bottlenecks.

1) Reducing $\mathbf{C}$ to the row-echelon form is costly (LU decomposition with row pivoting), and will introduce extra fill-ins into $\mathbf{G}$ during simultaneous operations (multiplications with the inverse of L, U factors).
2) It cannot guarantee that $\mathbf{C}_r$ is nonsingular after the first round of regularization, and if so, the process has to be repeated to eliminate more variables from $\mathbf{x}_1$ until a nonsingular $\mathbf{C}_r$ is achieved. This problem arises when the system of DAEs have an index higher than one, i.e., the output equation contains derivatives of the source terms, which would present in the above procedure only after the second cycle [27]. Such iterative check and elimination of singularity is unfavorable to computation efficiency and sparsity preservation.

The second-order index of a circuit is mostly due to the presence of CV-loop and LI-cutset in the circuit topology [28]. Hence, we propose to reduce the index-2 circuit to its index-1 equivalent *prior to* the elimination process of (13) by detecting and breaking all CV-loops and LI-cutsets in the topology. The index-1 system are then fed into (13) that is guaranteed to stop after *one* iteration.

B. Topological Index Reduction

Our index reduction method combines topology analysis and algebraic transformation in such a way that the latter (essentially GE) is only applied on a small portion of the original system selected by the former. Modifications are made on the matrix equation level instead of the netlist level for better adaptability. One key observation is that a loop with capacitors only does not lead to index-2 circuit in MNA; only when voltage source(s) come into the loop will the second order of index present [29]. This is different from LI-cutset in which inductors alone can form a cutset leading to index-2 system (because inductor currents are state variables in MNA).

Our method hence starts with eliminating one (nondatum) node voltage and the branch current for each (dependent and independent) voltage source regardless whether it is part of a CV-loop. This intends to break all (potential) CV-loops *in one shot* taking advantage of the (usually) small number of voltage sources in a circuit. The MNA stamp of independent voltage source is given in (17a) and the corresponding elimination flow follows from (17b) to (17f) (assume v_i and i_v are eliminated). Dependent voltage sources are eliminated analogously.

Treatment to LI-cutsets is similar, except now one inductor (not current source) per LI-cutset must be selected for elimination. The inductors to be eliminated are selected from a derived graph with only inductive branches and current sources [30]. Once the inductor is chosen for a given LI-cutset (denoted as L_1), a similar process to (17b) is applied to eliminate one node voltage (the node without capacitance). If the two end nodes of L_1 both have capacitance, only the inductor current needs to be eliminated. For the current variable, the algebraic constraint from Kirchhoff's current law (KCL) of the LI-cutset (18) and its differential version are applied to eliminate i_{L_1} and $\dot{i}_{L_1}$ in $\mathbf{G}$ and $\mathbf{C}$, respectively, as follows:

$$i_{L_1} + \sum_{j=2}^{N_L} i_{L_j} + \sum_{j=1}^{N_{I_s}} I_{s_j} = 0. \tag{18}$$

C. Treatment to Floating Capacitance

After eliminating all voltage sources and the selected inductors from the topology, the matrix $\mathbf{C}$ could still be singular, due to hidden (algebraic) singularity in $\mathbf{C}$ caused by the "floating

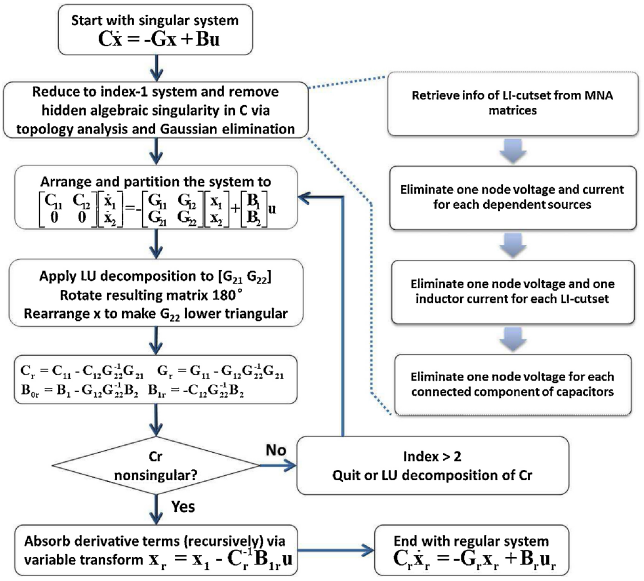

Fig. 1. Flow of regularization process.

capacitors," a group of connected capacitors isolated by non-capacitive elements, leaving one node voltage in the group algebraically dependent on the others. To avoid using LU decomposition to reveal this hidden singularity, we represent each group of floating capacitors by a connected component and locate them in a derived graph comprising only capacitive nodes and branches. We can then eliminate one node voltage using the algebraic and differential KCL of that node [26]. This way, the LU decomposition can be replaced by simply permuting all nonzero rows to the upper part of $\mathbf{C}$, which is much more desirable for speed and sparsity.

The complete flow of the regularization method is illustrated in Fig. 1. The underlying rationale of our method is to avoid (to most extent) certain matrix operations, such as (iterative) LU decomposition and matrix–matrix multiplications, that are of high complexity and tend to damage the sparsity of MNA matrices. This is crucial for any practical techniques in large-scale circuit simulation considering the giant size of the problems. We achieve this goal by preprocessing DAE index and floating capacitors before system elimination, and using topological analysis to guide such preprocessing that affects only a small portion of the entire matrices.

D. Complexity Analysis

The computational cost of regularization consists of the costs from topology analysis and algebraic transformation. The graph algorithms for topology analysis, such as finding minimal spanning tree and connected components, are in complexity of $O(N_g)$ or $O(N_g \log N_g)$ [31], where N_g is the size of a reduced graph [30]. Since the size of the graph is much smaller than the number of MNA variables, the cost of topology analysis is insignificant.

The algebraic transformation mainly includes row-wise elimination and LU decomposition of $[\mathbf{G}_{21}\mathbf{G}_{22}]$ in the systematic elimination stage, whose cost is generally topology dependent. The cost of row-wise elimination is determined by the number of voltage sources, LI-cutsets, and floating capacitors in a circuit. Based on our experience, the number of these "trouble maker" is usually less than 0.1% for million-scale designs. For the LU decomposition, the cost depends on the number of nodes without capacitances in a circuit, and such a circuit is uncommon.

IV. Computation of Matrix Exponential

A. Merge of Three Functions into One Matrix Exponential

The analytical solution (3) has three matrix exponential functions, which are generally referred as φ-functions of the zero, first, and second order [32]. Al-Mohy and Higham [33, Th. 2.1] has shown that instead of explicitly calculating φ-functions, a series of φ-functions can be calculated by computing the exponential of an $(n + p) \times (n + p)$ matrix, where n is the dimension of $\mathbf{A}$ and p is the order of the φ-functions, which is second order in (3). Therefore, we only need to calculate the exponential of a slightly larger matrix to obtain the analytical solution (3), which can be rewritten as

$$\mathbf{x}(t+h) = \begin{bmatrix} \mathbf{I}_n & 0 \end{bmatrix} e^{\mathbf{A}'h} \begin{bmatrix} \mathbf{x}(t) \\ e_2 \end{bmatrix} \tag{19}$$

with

$$\mathbf{A}' = \begin{bmatrix} \mathbf{A} & \mathbf{W} \\ \mathbf{0} & \mathbf{J} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h} & \mathbf{b}(t) \end{bmatrix}$$
$$\mathbf{J} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}. \tag{20}$$

Using (19), only one matrix exponential evaluation is needed in each time step, and the problem boils down to how to compute (19) efficiently.

B. Krylov Subspace Method

Intuitively, one could compute the matrix exponential $e^{\mathbf{A}}$ first and then multiply it by a vector. However, direct computation of the matrix exponential is expensive ($\sim O(n^3)$) and usually results in a full matrix that degrades the performance of subsequent matrix–vector multiplications. Fortunately, MEXP only needs the product of $e^{\mathbf{A}}\mathbf{v}$, which could be approximated efficiently using Krylov subspace projection [22], [23].

Krylov subspace approximation reduces the problem to the evaluation of the exponential of a much smaller matrix. According to the definition of exponent of matrix, we can write $e^{\mathbf{A}}\mathbf{v}$ as follows:

$$e^{\mathbf{A}}\mathbf{v} \equiv (\mathbf{I} + \mathbf{A} + \frac{\mathbf{A}^2}{2!} + \cdots + \frac{\mathbf{A}^k}{k!} + \ldots)\mathbf{v}. \tag{21}$$

The approximation of the above equation can be readily obtained from a Krylov subspace spanned by the basis of m vectors as follows:

$$\mathrm{K}_m(\mathbf{A}, \mathbf{v}) = \mathrm{span}\{\mathbf{v}, \mathbf{A}\mathbf{v}, \ldots, \mathbf{A}^{m-1}\mathbf{v}\}.$$

The Arnoldi process in Algorithm 1 can be used to construct an orthonormal basis $\mathbf{V_m}$ and a $m \times m$ upper Hessenberg matrix $\mathbf{H}(1:m, 1:m)$ denoted as $\mathbf{H_m}$ for the Krylov subspace

$$
\begin{aligned}
e^{\mathbf{A}'h}\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix} &= \exp\left(\begin{bmatrix}-\mathbf{C}^{-1}\mathbf{G} & \mathbf{C}^{-1}\mathbf{W}_u\\ 0 & \mathbf{J}\end{bmatrix}h\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix} = \exp\left(\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}\begin{bmatrix}-\mathbf{G} & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}\end{bmatrix}\frac{h}{\alpha}\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}\\
&= \exp\left(\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}\begin{bmatrix}-\mathbf{G}-(\mathbf{C}/\alpha) & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}-\mathbf{I}_2\end{bmatrix}\frac{h}{\alpha}+\begin{bmatrix}\mathbf{I}_n & 0\\ 0 & \mathbf{I}_2\end{bmatrix}\frac{h}{\alpha}\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}\\
&= \exp\left(\underbrace{\frac{1}{\alpha}\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}}_{\tilde{\mathbf{C}}}\underbrace{\begin{bmatrix}-\mathbf{G}-(\mathbf{C}/\alpha) & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}-\mathbf{I}_2\end{bmatrix}}_{\tilde{\mathbf{G}}}h\right)\underbrace{\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}}_{\mathbf{v}}e^{\frac{h}{\alpha}} = e^{\tilde{\mathbf{A}}h}\mathbf{v}.
\end{aligned}
\tag{22}
$$

Algorithm 1 Arnoldi process

Input: vector $\mathbf{v}$, $n \times n$ matrix $\mathbf{A}$ and m
Output: $(m+1) \times m$ matrix $\mathbf{H}$ and $\mathbf{V_m} = [\mathbf{v}_1, \ldots, \mathbf{v}_m]$
$\mathbf{v_1} = \mathbf{v}/\|\mathbf{v}\|_2$;
for $j = 1, 2, \ldots, m$ **do**
  $\mathbf{w} = \mathbf{A}\mathbf{v}_j$;
  for $i = 1, 2, \ldots, j$ **do**
    $\mathbf{H}(i, j) = \mathbf{w}^{\mathsf{T}}\mathbf{v}_i$;
    $\mathbf{w} = \mathbf{w} - \mathbf{H}(i, j)\mathbf{v}_i$;
  end
  $\mathbf{H}(j+1, j) = \|\mathbf{w}\|_2$;
  $\mathbf{v}_{j+1} = \mathbf{w}/\mathbf{H}(j+1, j)$;
end

K_m. Note that in each Arnoldi iteration, we compute $\mathbf{A}\mathbf{v}$ as $-\mathbf{C}^{-1}(\mathbf{G}\mathbf{v})$. Thus, the major cost requires one sparse matrix–vector multiplication and one sparse linear solve involving $\mathbf{C}$ only. The relation between $\mathbf{V_m}$ and $\mathbf{H_m}$ is given by

$$\mathbf{A}\mathbf{V_m} = \mathbf{V_m}\mathbf{H_m} + \mathbf{H}(m+1, m)\mathbf{v}_{m+1}e_m^{\mathsf{T}}$$

where e_m is the mth unit vector with dimension $m \times 1$. Because of the orthogonality of columns in $\mathbf{V_m}$, $\mathbf{H_m}$ can be expressed as

$$\mathbf{H_m} = \mathbf{V_m^{\mathsf{T}}}\mathbf{A}\mathbf{V_m}. \tag{23}$$

Then we project $\mathbf{A}$ onto the Krylov subspace and with (23) derive the approximation of $e^{\mathbf{A}}\mathbf{v}$ as [23]

$$
\begin{aligned}
e^{\mathbf{A}}\mathbf{v} &\approx \mathbf{V_m}\mathbf{V_m^{\mathsf{T}}}e^{\mathbf{A}}\mathbf{v} = \|\mathbf{v}\|_2\mathbf{V_m}\mathbf{V_m^{\mathsf{T}}}e^{\mathbf{A}}\mathbf{V_m}e_1\\
&= \|\mathbf{v}\|_2\mathbf{V_m}e^{\mathbf{H_m}}e_1.
\end{aligned}
\tag{24}
$$

To use the Krylov subspace method to compute (19), we first rewrite it into (22) where

$$\tilde{\mathbf{A}} = \frac{1}{\alpha}\tilde{\mathbf{C}}^{-1}\tilde{\mathbf{G}}, \qquad \mathbf{W}_u = \mathbf{B}\begin{bmatrix}\frac{\mathbf{u}(t+h)-\mathbf{u}(t)}{h} & \mathbf{u}(t)\end{bmatrix}. \tag{25}$$

The scaling factor α is introduced to balance the quantities in $\tilde{\mathbf{C}}$ and $\tilde{\mathbf{G}}$. The original exponent $\mathbf{A}'$ is left shifted by (multiple of) an identity matrix to make it nonsingular and as well turn the real parts of some small eigenvalues of $\tilde{\mathbf{A}}$ that may be positive in nonlinear simulation into negative. We generate $\mathbf{V_m}$ and $\mathbf{H_m}$ by Algorithm 1 with $\tilde{\mathbf{A}}h$ and $\mathbf{v}$ as inputs. Using (24), the overall solution of a new time step is

$$\mathbf{x}(t+h) = \begin{bmatrix}\mathbf{I}_n & 0\end{bmatrix}\|\mathbf{v}\|_2\mathbf{V_m}e^{\mathbf{H_m}h}e_1. \tag{26}$$

The value of m in the Krylov subspace approximation depends mostly on the spectrum of $\tilde{\mathbf{A}}$. The large (magnitude) eigenvalues of $\tilde{\mathbf{A}}$ correspond to the small eigenvalues of $\tilde{\mathbf{C}}$, i.e., the fast mode of the circuit, and the small eigenvalues relate to the slow mode of the circuit. It is commonly known that the Krylov subspace method approximates large eigenvalues better than small eigenvalues. A more precise statement is that the eigenvalues of $\mathbf{H_m}$ or the Ritz values tend to match the well-separated (extreme) eigenvalues $\tilde{\mathbf{A}}$ with priority to minimize the characteristic polynomial of $\mathbf{H_m}$ over the entire spectrum of $\tilde{\mathbf{A}}$. Therefore, a larger m is required only when $\tilde{\mathbf{A}}$ has many large and well-separated eigenvalues, meaning that the circuit contains many distinct fast modes. In our experiments, m often ranges from 10 to 100 while the actual dimension of $\tilde{\mathbf{A}}$ could be millions. With this small size, the $e^{\mathbf{H_m}h}$ can be computed efficiently by many existing techniques [34], [18], and the overall complexity of MEXP is greatly reduced.

We would like to mention that the Krylov subspace method has also been applied in some iterative methods [12], e.g., conjugate gradient or generalized minimal residual method to speed up the solution of linear system arising from, e.g., implicit numerical integration methods. It has been proved in [22] that convergence of Krylov subspace method for matrix exponential operator is faster than that for the iterative solution of linear systems.

C. Stability

The stability region of matrix exponential formulation (22) is the same as TRAP. Both methods are A-stable for passive circuits whose eigenvalues of $\tilde{\mathbf{A}}$ are negative. The approximative computation by the Krylov subspace method in (26) is also A-stable when $\tilde{\mathbf{A}}$ is normal by the following theorem.

Theorem 1: For passive circuits, MEXP computed by the Krylov subspace approximation is A-stable when $\tilde{\mathbf{A}}$ is normal.

Proof: We can express $\tilde{\mathbf{A}}$ in Jordan normal form as

$$\tilde{\mathbf{A}} = \mathbf{P}\mathbf{J}\mathbf{P}^{-1}$$

where $\mathbf{J}$ is upper triangular matrix and its diagonal terms are eigenvalues. It is trivial to represent the matrix exponential of $\tilde{\mathbf{A}}h$ as follows:

$$e^{\tilde{\mathbf{A}}h} = \mathbf{P}e^{\mathbf{J}h}\mathbf{P}^{-1}.$$

Since the eigenvalues of $\mathbf{J}$ are negative in passive circuits, the norm of $e^{\mathbf{J}h}$ tends to 0 as time step h increases to infinity. Therefore, (19) is A-stable for passive circuits.

To ensure stability of MEXP after performing Krylov subspace approximation, we need to guarantee that the norm of $\mathbf{V_m}e^{\mathbf{H_m}}$ also shrinks as h increases. Since $\mathbf{V_m}$ is orthonormal basis, we have

$$\|\mathbf{V_m}e^{\mathbf{H_m}h}\|_2 \leq \|e^{\mathbf{H_m}h}\|_2.$$

It is proved in [23] that the logarithmic norm of $\mathbf{H_m}h$ is no larger than that of $\tilde{\mathbf{A}}h$. We then have

$$\|\mathbf{V_m}e^{\mathbf{H_m}h}\|_2 \leq \|e^{\mathbf{H_m}h}\|_2 \leq e^{\mu(\mathbf{H_m}h)} \leq e^{\mu(\tilde{\mathbf{A}}h)}$$

where $\mu(\cdot)$ is the logarithmic norm. Since $\mu(\tilde{\mathbf{A}}h)$ of normal matrix is the largest eigenvalue of $\tilde{A}$, which is negative, the norm of $\mathbf{V_m}e^{\mathbf{H_m}}$ also tends to 0 as h increases to infinity. Hence, MEXP computed by the Krylov subspace approximation is A-stable for normal matrix. ■

Note that $\tilde{A}$ might not be normal for the applications to circuit analysis. It is reported in [16] that larger m could avoid instability from the Krylov subspace approximation because larger m approximates $e^{\tilde{\mathbf{A}}h}$ with less error (shown in Section IV-D). In our experiment, MEXP by Krylov subspace approximation with m ranged from 10 to 100 is stable for all test cases.

D. Error Analysis

A priori error bound of computing the matrix exponential (24) by the Krylov subspace projection is given by

$$err \leq 2\|\mathbf{v}\|_2 \frac{\rho^{m+1}e^{\rho}}{(m+1)!} \tag{27}$$

where $\rho = \|\tilde{\mathbf{A}}h\|_2$ [22], [23]. The equation indicates the approximation error depends on m and the $2-$norm of $\tilde{\mathbf{A}}h$. For stiff problems where $\mathbf{C}$ contains capacitance of very small values, the matrix $\tilde{\mathbf{A}}$ will have a large norm and therefore a small h is required to reduce the error in Krylov subspace computation of matrix exponential. This suggests that the proposed MEXP is more suitable for moderately stiff problems. One can also increase m to allow the usage of larger step size while maintaining accuracy, but at the cost of an increasing computation. This calls for a careful selection of h and m, which will be discussed in the next section.

In practice, the prior bound may not be sharp and is costly to evaluate. *A posteriori* error estimation proposed by Saad [23] is commonly adopted to determine the error of (26), which reads

$$err = \|\mathbf{v}\|_2 \mathbf{H}(m+1,m)|e_m^{\mathsf{T}}\varphi_1(\mathbf{H_m})e_1| \tag{28}$$

where $\varphi_1(x) = \frac{e^x-1}{x}$.

V. Adaptivity

One pleasing feature of MEXP lies in the ease of adaptively adjusting h during the numerical integration. According to (23), the Krylov subspace projection is scaling invariant, i.e., $\alpha\mathbf{A} \rightarrow \alpha\mathbf{H}_m$. Once we have to shrink/enlarge time step h in order to satisfy the error bound, it is convenient to reevaluate (26) with a new h by simply scaling the matrix $\mathbf{H}_m$ provided the PWL assumption of input waveforms ($\tilde{\mathbf{A}}$ remains constant). Thus, the reevaluation process of adjusting time step involves scaling of $\mathbf{H}_m$ and recomputing of the matrix exponential of $\mathbf{H}_m$. The time complexities for scaling and dense matrix exponential are $O(n^2)$ and $O(n^3)$, respectively. Since the size of $\mathbf{H}_m$ is small, the computation cost of whole reevaluation process is insignificant. In contrast, the implicit methods have to resolve the whole linear system whenever h is changed.

Taking advantage of this ease, we devise a prediction-correction scheme to dynamically adjust the step size h and the dimension of Krylov subspace approximation m during time stepping. At each step, a new pair of h and m are first predicted based on the knowledge of current step, with attempt to minimize the computation needed to complete the remaining time integration under given error constraint. When the *a posteriori* error resulted from the predicted h and m does not meet certain criteria, a correction scheme is applied by adjusting h until the error is satisfactory.

Given a predefined global error budget *Tol* and the error at nth step ϵ_n estimated by (28), we require the error at $n+1$th step ϵ_{n+1} to meet the following inequality:

$$\frac{\epsilon_{n+1}}{\epsilon_n} \leqslant \gamma\frac{\epsilon_{n+1}^{\max}}{\epsilon_n} = \gamma\frac{h_{n+1}Tol}{t_f\epsilon_n} = \gamma\frac{h_nTol}{t_f\epsilon_n}\frac{h_{n+1}}{h_n} = \frac{\gamma}{w}\frac{h_{n+1}}{h_n} \tag{29}$$

where t_f is the end time, $\epsilon_{n+1}^{\max}$ is the maximum error allowed at t_{n+1}, and γ is a safety factor commonly taking 0.8. The quality of the solution at t_n is measured by the ratio

$$w = \frac{\epsilon_n}{\epsilon_n^{\max}} = \frac{t_f\epsilon_n}{h_nTol}. \tag{30}$$

A. Prediction of h and m

Unlike the separate changes of h and m in [32], we allow h and m vary at the same time, by a moderate extent, and estimate the error at the next step, according to the prior error bound (27), as

$$\frac{\epsilon_{n+1}}{\epsilon_n} = \left(\frac{h_{n+1}}{h_n}\right)^{\frac{m_{n+1}}{\beta}+1}\kappa^{-(m_{n+1}-m_n)} \tag{31}$$

where β and κ are two parameters to be determined. The optimal combination of h and m is selected to minimize the remaining computation after current time point subjected to the error constraint in (29), that is

$$\min\ \frac{t_f-t_n}{h}Q(m) \tag{32a}$$

$$s.\,t.\ \left(\frac{h}{h_n}\right)^{\frac{m}{\beta}}\kappa^{-(m-m_n)} \leqslant \frac{\gamma}{w}. \tag{32b}$$

Note that, by intuition, we expect the error constraint is a monotone decreasing function of m, i.e., the higher is the dimension of Krylov subspace, the smaller is the error. This imposes a limit on the factor by which a new step size can

grow from requiring the derivative with reference to (w.r.t.) m of (32b) is negative, which gives

$$h/h_n \leqslant \kappa^{\beta}. \tag{33}$$

Function $Q(m)$ is the estimated cost of one stepping in terms of m, in which the most time-consuming part is the Arnoldi process listed in Algorithm 1. We neglect the cost of computing the matrix exponential of the reduced matrix. Each Arnoldi process costs roughly $\frac{3}{2}\left(m^2 - m + 1\right)(N+2)$ flops (for orthogonalization), m sparse matrix–vector multiplications and m sparse linear solves. Computation required in the last two operations can be estimated by $2mN_{\tilde{G}}$ and $2mN_{\tilde{C}}^{p}$, where $N_{\tilde{G}}$ and $N_{\tilde{C}}$ denote the numbers of nonzeros of $\tilde{\mathbf{G}}$ and $\tilde{\mathbf{C}}$. The complexity factor p for sparse linear solve depends on the structure of $\tilde{\mathbf{C}}$ and the solution method, whose value usually ranges from 1 (diagonal matrix) to 1.5. As a result, we formulate $Q(m)$ as

$$Q(m) = c_1 m^2 + c_2 m + c_3 \tag{34}$$

with $c_1 = \frac{3}{2}(N+2)$, $c_2 = 2(N_{\tilde{G}} + N_{\tilde{C}}^{p}) - \frac{3}{2}(N+2)$, and $c_3 = \frac{3}{2}(N+2)$.

We argue that the objective function (32a) achieves minimum when the constraint (32b) takes equality and postpone the proof later in this section. With this assumption, we solve h from (32b) as

$$\begin{aligned}\ln h &= \beta\left(\log\left(\frac{\gamma}{w}\right) - m_n \log\kappa\right)\frac{1}{m} + (\beta\log\kappa + \log h_n) \\ &= c_4 m^{-1} + c_5 = P(m).\end{aligned} \tag{35}$$

Substituting (35) into (32a), the objective function becomes a function of m only, namely, $\left(t_f - t_n\right) Q(m) e^{-P(m)}$, and the extreme value is obtained when the function derivative is zero, yielding

$$2c_1 m^3 + (c_2 + c_1 c_4)\, m^2 + c_2 c_4 m + c_3 c_4 = 0 \tag{36}$$

whose positive roots are the solution of m. The corresponding h is then obtained by (35). The new h is restricted by the negative derivative constraint (33) and the constraint of PWL input, and thus will be overwritten by the maximum value jointly set by the two constraints when any of them is hit. The prediction of m is updated accordingly.

In the following, we prove that the m and h selected by the above process is the optimal solution of the optimization problem (32).

Lemma 1: Provided (33) holds, the polynomial in (36) has one and only one positive root.

Proof: It is trivial to show $c_1 > 0$, $c_2 > 0$, and $c_3 > 0$. If (33) holds, we have $c_4 < 0$ from (35). The number of sign changes between the coefficients of the polynomial in (36) is one regardless of the sign of the second coefficient. Determined by Descartes' rule of signs, the polynomial has exactly one positive root. ■

Theorem 2: Given (33), the m_{opt} and h_{opt} computed by (36) and (35) are the optimal solution of (32).

Proof: We prove it by contradiction. Denote (32a) and (32b) by $F(h,m)$ and $C(h,m) \leqslant \frac{\gamma}{w}$. We assume there exists another pair of (h', m') $(h' \neq h_{\text{opt}}, m' \neq m_{\text{opt}})$ being a solution no worse than $h_{\text{opt}}, m_{\text{opt}}$ for (32), that is

$$F(h', m') \leqslant F(h_{\text{opt}}, m_{\text{opt}}),\ C(h', m') \leqslant C(h_{\text{opt}}, m_{\text{opt}}). \tag{37}$$

If $C(h', m') < C(h_{\text{opt}}, m_{\text{opt}})$, since $C(h,m)$ is an increasing function of h and $F(h,m)$ is a decreasing function of h, one can increase h' to $\tilde{h}'$ to make $C(\tilde{h}', m') = \frac{\gamma}{w}$ and $F(\tilde{h}', m') < F(h', m') \leqslant F(h_{\text{opt}}, m_{\text{opt}})$, which is contradictory to the fact that $F(h_{\text{opt}}, m_{\text{opt}})$ is at its minimum for the equality constraint. If $C(h', m') = C(h_{\text{opt}}, m_{\text{opt}})$ (and $F(h', m') = F(h_{\text{opt}}, m_{\text{opt}})$), it is equivalent that m' is another positive solution of (36), which is in contradiction to Lemma 1. ■

We determine the two parameters β and κ in a heuristic manner taking advantage of the fact that, given a calculated Krylov pair $\mathbf{H}$ and $\mathbf{V}$, the effort required to obtain *a posteriori* error estimate for a new h and m is trivial. Assume h_n, m_n, and ϵ_n are known at current step, for each prediction, we compute the error at five sampling points surrounding (h_n, m_n), namely, (eh_n, m_n), $(\frac{1}{e}h_n, m_n)$, $(h_n, \frac{3}{4}m_n)$, $(eh_n, \frac{3}{4}m_n)$, and $(\frac{1}{e}h_n, \frac{3}{4}m_n)$. Here, we only scale down m_n as the new $\mathbf{H}$ is simply a submatrix of the original one, and upscaling m_n requires extra Arnoldi iterations. Then the two unknown parameters are determined by the least squares (LSs) fitting of (31) with the above five data points.

With only moderate accuracy requirement, we solve the LS fitting problem by taking logarithm on both sides of (31) as follows:

$$\left(\frac{m}{\beta} + 1\right)\log\frac{h}{h_n} - (m - m_m)\log\kappa = \log\frac{\epsilon}{\epsilon_n}. \tag{38}$$

With the notation of $a_1 = 1/\beta, a_2 = \log\kappa, y_1 = \log\frac{h}{h_n}, y_2 = m, z = \log\frac{\epsilon}{\epsilon_n}$, the parameters β and κ are derived from the LS solution of the overdetermined system as

$$\begin{bmatrix} y_1^{(1)} & m_n - y_2^{(1)} \\ y_1^{(2)} & m_n - y_2^{(2)} \\ \vdots & \vdots \\ y_1^{(5)} & m_n - y_2^{(5)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} z^{(1)} - y_1^{(1)} \\ z^{(2)} - y_1^{(2)} \\ \vdots \\ z^{(5)} - y_1^{(5)} \end{bmatrix} \tag{39}$$

where (y_1^i, y_2^i) and z^i are computed from the five sampling points and the corresponding *a posteriori* errors.

B. Correction of h Based on A Posteriori Error

The prediction scheme provides a useful insight for selecting h and m for the next time step. Nevertheless, it may occasionally lead to (h,m) pair that has *a posteriori* error violating the error constraint or too small to fully use the error margin. Therefore, we employ a posterior correction scheme to refine h to ensure the error stay within an appropriate region below the error threshold. Specifically, the scheme will repeatedly enlarge or shrink h by a given ratio and forward the time frame only when the *a posteriori* error falls into an

Algorithm 2 Overall prediction-correction flow

Input: matrices $\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$, input $\mathbf{u}(t)$, initial time step h, initial m, total error budget Tol, and total time t_f
Output: result $\mathbf{x}(t)$
$t = 0$; $\mathbf{x}(0)$ = dc_analysis;
while $t \leq T$ **do**
  $[\mathbf{C}_r, \mathbf{G}_r, \mathbf{B}_r]$ = regularization($\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$);
  evaluate $\mathbf{H}_m$ and $\mathbf{V}_m$ by Krylov subspace method;
  while $w < w_{\min}$ **do**
    scale up h and $\mathbf{H}_m$;
    compute *a posteriori* error ϵ by (28) and new w;
  end
  while $w > w_{\max}$ **do**
    scale down h and $\mathbf{H}_m$;
    compute *a posteriori* error ϵ and new w;
  end
  calculate $\mathbf{x}_{\text{new}}$ by (26);
  $[\beta, \kappa]$ = findParameter(h, m, ϵ, $\mathbf{H}$, $\mathbf{V}$);
  $[h_{\text{new}}, m_{\text{new}}]$ = prediction(t_f, t, h, m, w, β,κ);
  $t = t + h$;
  $\mathbf{x}(t) = \mathbf{x}_{\text{new}}$;
  $h = h_{\text{new}}$;
  $m = m_{\text{new}}$;
end

interval of $[w_{\min}, w_{\max}]$. When enlarging h, the two constraints defined in Section V-A also apply to prevent overshooting.

Note that a similar step-size control have been applied in commercial SPICE-like simulators, such as HSPICE, to constrain the LTE of each time step. Yet the adaptivity of implicit methods and matrix exponential approach is quite different in the following aspects.

1) When a step size is changed, the linear system in the implicit methods will also change, such as $\mathbf{C}/h - \mathbf{G}/2$ in TRAP. Thus, the implicit methods must solve the linear system again for every time-step reversal. In practice, it is often seen that HSPICE takes a long time to perform one-time stepping because the LTE control forces the simulator to solve a linear system many times to find a feasible h. MEXP is free from this overhead when adjusting time step owing to the scaling invariant property of Krylov subspace projection.
2) Since in implicit methods there is no easy update of solution for a different step size, an increase in step size, if possible, can only happen in the next step. In contrast, with simple scaling and reevaluation of a small matrix exponential, one can apply the largest permissible step size right in the same step.
3) Varying order of approximation, e.g., automatic switch between first, second, and higher order of implicit methods is difficult. On the other hand, the matrix exponential approach allows a simultaneous adjustment of h and m within a wide range to optimize the computational efficiency.

The overall flow of prediction-correction scheme is shown in Algorithm 2.

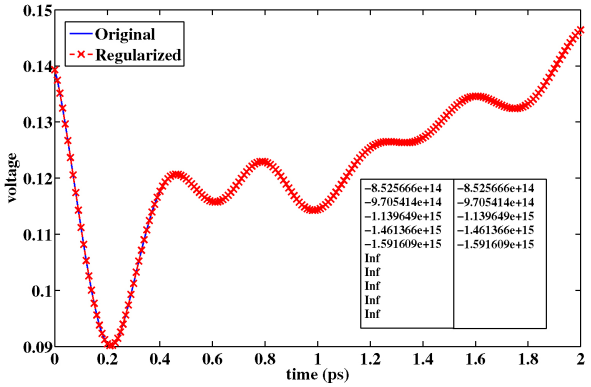

Fig. 2. Accuracy of regularization process.

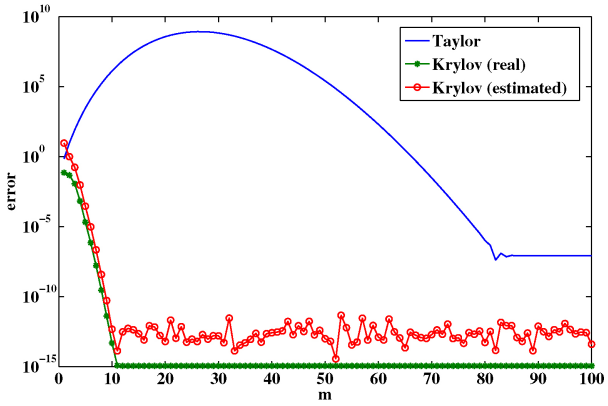

Fig. 3. Errors of computing $e^{\mathbf{A}h}v$ by Taylor's expansion and Krylov subspace method w.r.t. m. Reference solution is obtained by expm($\mathbf{A}h$)v (h = 0.1ps). Both real error and *a posteriori* error estimate of Krylov method are shown.

VI. Experimental Results

We prototype MEXP in MATLAB and integrate with a SPICE-like circuit simulator SMORES developed in MIT [35]. Experiments are performed on a server with Intel Xeon 3.0 GHz CPU and 16 GB memory, with testbench circuits of different sizes and characteristics.

Table I provides detailed specifications. Type indicates linear (L) or nonlinear (NL) circuits. Index gives the DAE index of the MNA systems. The numbers of nonzeros in $\mathbf{C} + \mathbf{G}$ before and after regularization are also shown. For fairness, a MATLAB implementation of TRAP is used to provide benchmarks for accuracy and performance comparisons. All linear systems are solved by the direct solver (backslash) in MATLAB.

A. Results for Regularization

Fig. 2 validates the accuracy of regularization method, in which the transient response of D2 before and after regularization are compared. Since no approximation is introduced, the regularization maintains the accuracy very well (relative mismatch between the two curves is 4.3×10^{-11}). The five largest (in magnitude) generalized eigenvalues of $(\mathbf{C}, -\mathbf{G})$ and $(\mathbf{C}_r, -\mathbf{G}_r)$, shown inset of Fig. 2, also demonstrates an exact equivalence between the original and the regularized systems.

TABLE I
SPECIFICATION OF TEST CASES

Design	Category	Type	Nodes	Nodes w/o Cap	Index	nnz($\mathbf{C}+\mathbf{G}$) Before Reg.	nnz($\mathbf{C}+\mathbf{G}$) After Reg.
D1	Power grid	L	2.5K	0	1	12.3K	12.3K
D2	Trans. line	L	5.6K	431	2	0.9M	0.9M
D3	Power grid	L	160K	0	1	1.8M	1.8M
D4	Power grid	L	1.6M	0.6M	2	5.4M	4.8M
D5	Power grid	L	4M	0	1	44.2M	44.2M
D6	Inv. chain	NL	82	0	1	342	342
D7	Power amp.	NL	342	105	2	2.2K	2.1K
D8	16-bit adder	NL	579	0	1	3.6K	3.6K
D9	ALU	NL	10K	373	1	44.3K	43.4K

B. Performance of Krylov Subspace Method in Computing Matrix Exponential

In this section, we show the numerical advantage of Krylov subspace method over traditional Taylor's expansion (21). While Taylor's expansion can approximate matrix exponential with the order of m, its accuracy is worse than that of Krylov subspace method with m dimensions. To demonstrate the difference, we perform both Taylor's expansion and Krylov subspace method on a small RC circuit with 500 nodes and capacitances whose values vary from 10^{-11} to 10^{-16}. Fig. 3 shows the advantage of using the Krylov subspace method (24) to calculate $e^{\mathbf{A}h}v$, compared with Taylor's expansion (21). The reference result is computed via the MATLAB built-in function expm [34], which is accurate for small-scale matrices.

The convergence rate of Taylor's expansion depends on the norm of the matrix in the series in (21), i.e., how fast the factorial in denominator can dominate the nominator. In Fig. 3, the error increases with m at first due to the faster increase of matrix power than that of factorial, and then drops later when the factorial starts to outweigh the matrix power. The error only saturates after $m = 80$ at about 10^{-8}. The Krylov subspace method approximates the matrix exponential by orthogonal basis of the Krylov space $K_m(\mathbf{A}, \mathbf{v})$. Since the orthogonalization process minimizes norm of $\mathbf{v}_{m+1}$ in (23), which is major source of the approximation error, the Krylov subspace method is more accurate than Taylor's expansion under the same dimensions. Fig. 3 shows that the error of Krylov subspace method saturates to 10^{-15} at $m = 11$. The error estimated by the *a posteriori* formula (28) is also shown, which stays above the real error all the time and is fairly sharp. Also, we would like to mention that the actual approximation error of Krylov subspace method could be smaller than the result shown in Fig. 3, which is limited by the double precision.

C. Performance of Uniform MEXP

Fig. 4 shows the transient response of D3 simulated by MEXP, TRAP (TR), and forward Euler method (FE). We apply fixed step sizes 1 ps and 5 ps for both MEXP and TRAP method, and 1 ps for FE. The m in MEXP is 20.

The figure demonstrates the capability of MEXP to use large step size for numerical integration. With a larger h of 5 ps, MEXP can till have its waveform "jump" to the correct points (the yellow crosses) at the waveform of 1 ps. The pointwise

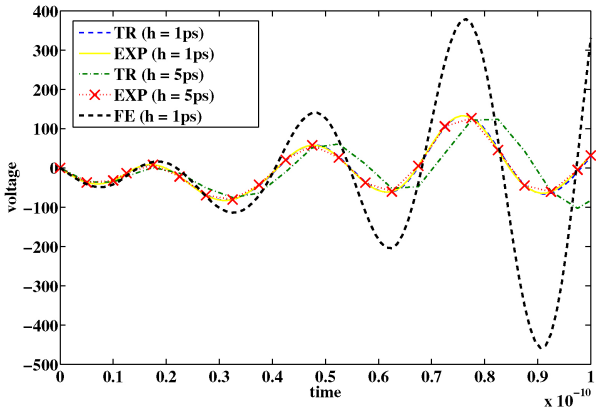

Fig. 4. Accuracy comparison among matrix exponential, trapezoidal, and FEs for different step size (linear cases).

mismatch between the two waveforms is only 9.4×10^{-3}. On the other hand, TRAP cannot capture the high frequency behavior as MEXP when using a large step size of 5 ps. Therefore, MEXP is reliable even when the time steps skip some high frequency details, provided that the matrix exponential is calculated accurately. Such "coarse-grain" accuracy is owing to the analytical nature of MEXP, which allows designers to take a fast yet accurate sweep of the global behavior of a circuit by a very large step size. The explicit forward Euler is unstable even with the time step of 1 ps.

Table II gives a detailed comparisons between MEXP and TRAP using fixed h and m for the five linear cases. A reference solution is first obtained by TRAP using a small time step $\mathrm{h_{ref}}$ for a time span t_f. The runtime and the L2-norm error w.r.t. the reference solution are recorded for the TRAP and MEXP when using $h = 10h_{\mathrm{ref}}$ and $h = 100h_{\mathrm{ref}}$. Among the four examples, D1 is highly stiff with minimum capacitance $\sim 10^{-19}$, D2 and D4 are moderately stiff with minimum capacitance $\sim 10^{-16}$, and D3 and D5 are less stiff with minimum capacitance $\sim 10^{-13}$.

For systems with small to moderate stiffness (D2 and D3), MEXP has a better accuracy than TRAP, owing to the analytical nature of the former's solution. MEXP causes more errors for D1 of large stiffness, due to the large norm of $\tilde{\mathbf{A}}h$. Either a smaller h or a larger m is required for better accuracy, which suggests MEXP is more suitable for slight and moderately stiff systems. Nevertheless, this is solely due to the accuracy consideration (and efficiency tradeoff), instead

TABLE II
PERFORMANCE COMPARISON WITH UNIFORM h AND m FOR LINEAR CASES

Case	TR						EXP			
	h_{ref}	T	t	h	t	Error w.r.t. h_{ref}	h	t	Error w.r.t. h_{ref}	m
D1	0.01 ps	100 ps	1957.3 s	0.1 ps	34.5 s	3.0×10^{-4}	0.1 ps	180.3 s	8.6×10^{-3}	30
				1 ps	1.9 s	4.4×10^{-3}	1 ps	20.6 s	4.0×10^{-1}	
D2	0.01 ps	10 ps	2728.3 s	0.1 ps	282.2 s	7.1×10^{-2}	0.1 ps	589.4 s	3.7×10^{-3}	30
				1 ps	43.5 s	2.1×10^{-1}	1 ps	90.8 s	3.8×10^{-2}	
D3	0.1 ps	100 ps	27064.5 s	1 ps	2907.1 s	2.8×10^{-3}	1 ps	1190.2 s	2.8×10^{-5}	20
				10 ps	426.6 s	2.1×10^{-1}	10 ps	176.8 s	3.2×10^{-5}	
D4	0.01 ps	10 ps	1.8×10^{5} s (est.)	0.1 ps	14760.2 s	N/A	0.1 ps	3796.2 s	N/A	40
				1 ps	2102.5 s	N/A	1 ps	565.8 s	N/A	
D5	0.1 ps	100 ps	N/A	0.1 ps	N/A	N/A	0.1 ps	6491.1 s	N/A	20
				1 ps	N/A	N/A	1 ps	1168.7 s	N/A	

of the stability limitation confronting the traditional explicit methods.

In terms of runtime, MEXP is slower than TRAP for small cases, but provides a noticeable speedup for large cases (∼4X for D4). This is attributed to the fact that in the Arnoldi process we only need to factor the matrix $\tilde{\mathbf{C}}$, which is generally sparser and well structured than $\tilde{\mathbf{C}} + \tilde{\mathbf{G}}$ that needs to be factored in common implicit methods. For the extremely large-example D5 (the matrix dimension exceeds $10M$), TRAP simply breaks down due to the memory limit in matrix factorization, while MEXP remains applicable, suggesting a better scalability in terms of memory usage. Apart from the benefit from improved matrix structure, the orthogonalization process in Arnoldi iteration is naturally parallelizable, which implies more potential computational benefit compared to the direct linear solution.

D. Performance of Adaptive MEXP

This section demonstrates the advantage of using the adaptive scheme in Section V in MEXP. We first verify that the formula (31) provides a usable error prediction for matrix exponential computation. Fig. 5 compares the predicted error and the real *a posteriori* error by (28) at each step of a simulation with D1. A ramp signal is used to avoid PWL input restriction on step size. The total error budget is 10^{-4}. Step sizes are tuned by a factor of 1.25 each time in posterior correction to ensure w to fall into [0.6, 1.2]. It is seen that the prediction generally captures the behavior of the real posteriori error.

To further demonstrate the quality of the predicted (h, m) pair, we vary the h and m over a range at each step and evaluate the corresponding cost function (34). The ranges of h and m variations are $[0.1h, 10h]$ and $[\frac{3}{4}m, \frac{4}{3}m]$, respectively. We choose a new (h, m) corresponding to the minimal $Q(m)$ (and satisfying error constraint), which is regarded the real "optimal" solution, and compare it to the predicted (h, m). Fig. 6 indicates a good match between the predicted and real optimal (h, m) pairs, and thus the effectiveness of our prediction scheme.

Table III compares the performance of TRAP and MEXP with adaptive control for the four linear cases. The error is measured by w in (30) with an overall budget $Tol = 10^{-3}$. In the adaptive TRAP, LTE of each step is measured by $h_n^3 \dddot{\mathbf{x}}/12$, and is used to provide a new h for the reversal of

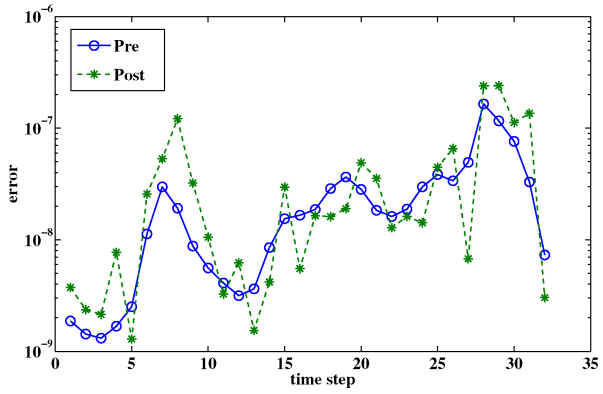

Fig. 5. Predicted and *a posteriori* errors comparison.

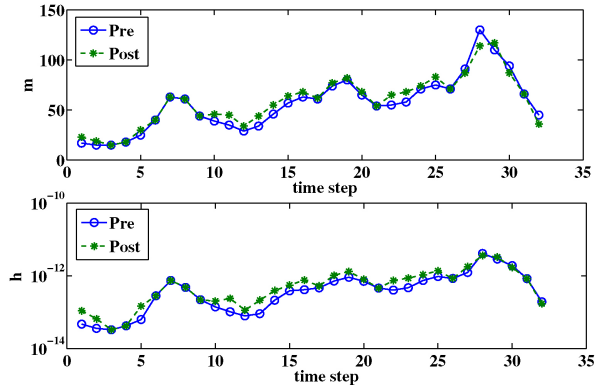

Fig. 6. (a) Predicted and (b) real optimal (h, m) pair.

current step (if $w > 1.2$) or for the next step (if $w < 0.6$), i.e., $h_{\text{new}} = \sqrt{w} h_{\text{old}}$. We also implement two versions of adaptive MEXP: one adjusts h only using the correction scheme, and the other adjusts both h and m by the prediction-correction scheme described in Section V. The correction is again conducted to ensure $0.6 \leq w \leq 1.2$. N_t and N_{ws} denote the number of time steps and the number of linear solves that have been wasted due to the time step adjustment in TRAP.

Albeit indispensable in practice, the adaptive time-step control largely affects the performance of TRAP due to the repeating solution of a large linear system whenever the LTE requirement is not met. In the worst-case D2, nearly one-third of total linear solves are "wasted" in the LTE control.

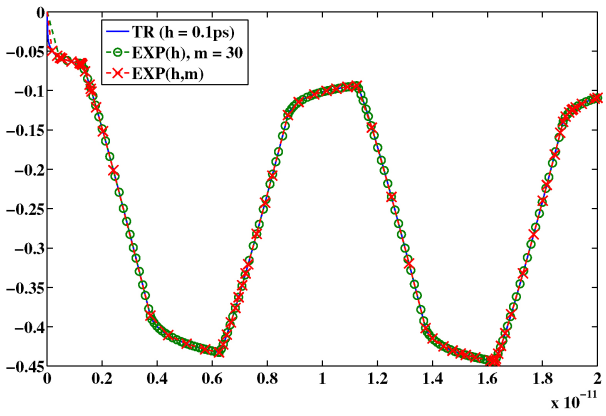

Fig. 7. Selection of time points for the two adaptive MEXPs (D1, $Tol = 10^{-3}$).

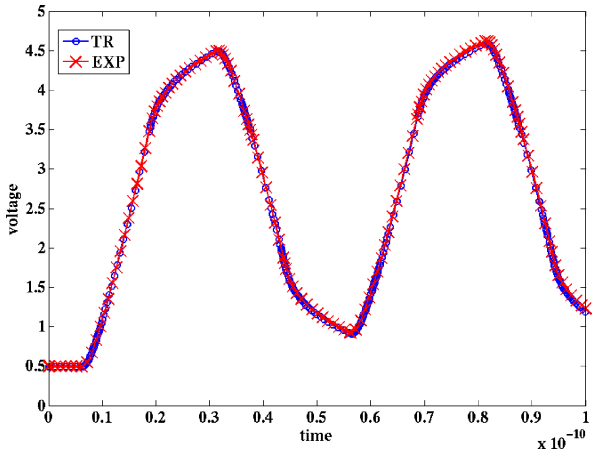

Fig. 8. Accuracy and time-point selection of MEXP for nonlinear circuit (D8).

MEXP, on the other hand, avoids this kind of overhead by projecting the original large-scale matrix onto a much smaller subspace, on which the error estimate and management are highly efficient. With the same error budget, the maximum speedup from the adaptive MEXP is over 15X (D3). The performance of EXP(h, m) is superior over EXP(h), with improvement from 1.3X to 2.8X. This demonstrates the benefit of allowing m to vary over steps at the same time using our prediction-correction scheme. Fig. 7 shows the difference in point selection of EXP(h) and EXP(h, m) for D1 with a two-square wave input.

Situation in nonlinear circuit simulation is more complicated. The time step is not only limited by the error in computing the matrix exponential term, but also by the error and convergence rate in solving the nonlinear system. Hence, the step size of nonlinear circuits may not be as large as that of linear circuits whose the error is only from the Krylov subspace method. In Table IV, we show the performance data of TRAP and MEXP for the four nonlinear examples. Nonlinearity is handled by the Newton's method in TRAP, and the fixed point iteration (10) in MEXP, both with the same convergence criteria. In TRAP, we follow the HSPICE convention [36] to control time step by counting the number of iterations [increases (reduce) h by 1.25X if the number of iterations is less than 3 (larger than 20)], and by LTE as in the linear cases after the Newton's iteration converges. If the LTE does not meet the prescribed accuracy requirement, the time step is reversed and the Newton's iteration is restarted with a smaller h. N_{it} denotes the number of total number of nonlinear iterations and N_{ws} the number of iterations wasted due to time-step reversal. In MEXP, h is controlled by the error of computing matrix exponential, and the error of nonlinear approximation (9) and (12). *A posteriori* correction is used to reduce h when the accuracy of Krylov subspace approximation is not sufficient. The fixed-point solution process is repeated if the nonlinear error is large, which also results in certain extra iterations counted by N_{ws}. For a new step, we do not apply the prediction scheme as in the linear cases to forecast h and m. The new h will be jointly determined by several values estimated by the current matrix exponential error, nonlinear error, and convergence condition (11), whichever is smaller.

Due to slower convergence of the fixed point iteration, MEXP generally requires more nonlinear iterations than TRAP using the Newton's method, which can be seen by counting the average per-step effective iteration $(N_{\text{it}} - N_{\text{ws}})/N_t$. This number ranges from 1.99 to 2.89 for TRAP while from 2.00 to 5.72 for MEXP. However, the separation of linear solution and nonlinear solution makes the error control in MEXP more straightforward than TRAP and largely avoids time-step reversal. In the calculation of LTE in TRAP, the contributions from linear elements and nonlinear elements are mixed together. Large error from either linear or nonlinear part will cause the violation of LTE and thus the time-step reversal. As seen in Table IV, a considerable portion of nonlinear iterations are wasted due to the time reversal, which in the worse case is nearly one half (D8). In MEXP, the numerical errors from linear and nonlinear solutions are separated. The nonlinear iteration is restarted only when the nonlinear error is large, which involves only the contribution from nonlinear elements. The error from solving linear elements is handled by the efficient error management unique for Krylov subspace approximation. It can be seen that the wasted nonlinear iterations in MEXP takes a much smaller fraction in total iterations than that in TRAP. The large number of time steps used for D7 is due to its stiffness (min capacitance $\sim 10^{-16}$), where the error of computing matrix exponential forces to adapt a small step size. With such small step, the nonlinear iteration converges fast, which is seen that there is no step reversal due to nonlinear error and only two per-step effective iterations. In terms of per-iteration runtime, MEXP also outperforms TRAP for large problem (D9) as seen in the linear case. The maximum overall speedup from MEXP is about 3.7X. The simulated responses of the two methods for the adder case (D8) are shown in Fig. 8.

Fig. 9 shows the runtime breakdown of EXP(h, m) in Table III and EXP in Table IV. The breakdown includes the runtime percentage of four main steps of Algorithm 2, which are regularization, computation of $\mathbf{H}_m$ and $\mathbf{V}_m$ by Arnoldi process, *a posteriori* error estimation, and the calculation of $\mathbf{x}_{\text{new}}$. Since the runtime of "findParameter" and "prediction" are insignificant, the figure does not include both the

TABLE III
PERFORMANCE COMPARISON WITH ADAPTIVE h AND m FOR LINEAR CASES ($Tol = 10^{-3}$)

Case	T	h_{init}	m_{init}	TR(h)			EXP(h)		EXP(h, m)	
				N_t	t	N_{ex}	N_t	t	N_t	t
D1	100 ps	0.01 ps	30	416	11.3 s	117	759	201.5 s	120	72.6 s
D2	10 ps	0.01 ps	30	99	394.1 s	46	41	214.7 s	40	161.4 s
D3	100 ps	0.1 ps	20	130	3118.2 s	5	70	200.2 s	55	112.3 s
D4	10 ps	0.01 ps	40	187	33802.1 s	29	181	3004.1 s	174	2135.2 s
D5	100 ps	0.1 ps	20	N/A	N/A	N/A	75	4927.2 s	45	3075.9 s

TABLE IV
PERFORMANCE COMPARISON FOR NONLINEAR CASES

Case	T	h_{init}	TR				EXP				
			N_t	N_{it}	N_{ws}	t	N_t	N_{it}	N_{ws}	t	m
D6	1 ns	1 ps	259	1062	357	35.6 s	222	1438	166	164.1 s	20
D7	100 ps	0.1 ps	242	670	187	68.4 s	533	1070	0	292.6 s	30
D8	100 ps	0.1 ps	371	1996	923	671.4 s	114	708	108	408.7 s	20
D9	100 ps	0.1 ps	451	1512	501	8244.5 s	285	1299	101	2252.3 s	30

TABLE V
SUMMARY OF THE FEATURES OF EXPLICIT, IMPLICIT, AND MEXPS

Methods	Nature	Stability for Passives	Matrix to Inverse	Main Cost	Memory[a]	Adaptivity[b]	Cost of Adaption[c]	Error Origin
Implicit	Poly. approx. order ≤ 10	High	$\mathbf{C} + h\mathbf{G}$	Linear solve	$(N_{\mathbf{C}+\mathbf{G}})^{1.5}$	h only	High	Taylor truncation
Polynomial explicit	Poly. approx. order ≤ 10	Weak	$\mathbf{C}$	Matrix-vector multiplication	$N_{\mathbf{C}}^{1.5}$	h only	Low	Taylor truncation
Matrix exponential	Analytical	High	$\mathbf{C}$	Arnoldi process	$\max(N_{\mathbf{C}}^{1.5}, mN)$	h and m	Low	Matrix EXP computation

[a]Estimated fill-in factor of 1.5 is used for sparse LU factorization, $N_{\mathbf{C}}$ is the number of nonzeros of $\mathbf{C}$, N the dimension of $\mathbf{C}$. [b]Variable order BDF is not considered here. [c]Cost of reevaluation for a new step size.

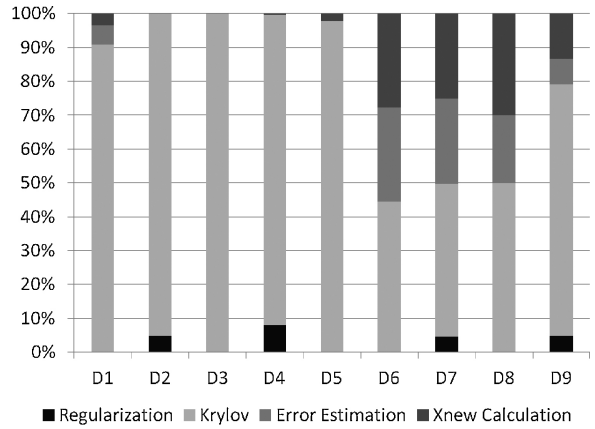

Fig. 9. Runtime breakdown for main steps in Algorithm 2.

operations. As we can see, only cases D2, D4, D7, and D9 require regularization, and the corresponding runtimes in these cases only take 4.7%, 8%, 4.5%, and 4.8%, which demonstrates practicability of the regularization process. The computation of Krylov subspace method generally dominates the performance for large cases, which takes more than 70% for the cases with size larger than 1K. In contrast, the computation time of small cases is more relevant to the number of error estimations and calculations of $\mathbf{x}_{\text{new}}$, which is larger in the nonlinear cases, such as D6, D7, and D8.

As a final remark, Table V compares the major characteristics of (traditional) explicit methods, implicit methods, and MEXP within the context of circuit simulation. Each method is shown to have its own strength and weakness, and thus its own appropriate range of application. Explicit methods has the best per-step performance but the worst stability problem, rendering it is more suitable for designs known to be nonstiff. Implicit methods are the most robust approach for general situations, although with a relatively low scalability and adaptivity. MEXP to some extent fills in the gap between explicit and implicit methods, by eliminating the stability difficulty of the former and providing better scalability than the latter, for a wide range of application with small to intermediate stiffness.

VII. CONCLUSION

An explicit numerical integration method has been presented for accurate and efficient time-domain circuit simulation. Different from conventional linear multistep method, MEXP solves the linear differential equation analytically via the matrix exponential operator. The computation of matrix exponential was significantly accelerated by the Krylov subspace method. The proposed method alleviates the stability bottleneck of explicit methods and enables great adaptivity for time-step size control. Numerical experiments have confirmed the superiority of the proposed method.

VIII. ACKNOWLEDGMENT

The authors would like to thank the reviewers for their valuable comments that led to significant improvement of this paper.

REFERENCES

[1] L. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Dept. Electric. Eng. Comput. Sci., College Eng., Univ. California, Berkeley, Tech. Rep. UCB/ERL M520, 1975.

[2] A. Devgan and R. A. Rohrer, "Event driven adaptively controlled explicit simulation of integrated circuits," in *Proc. ICCAD*, 1993, pp. 136–140.

[3] W. Dong, P. Li, and X. Ye, "Wavepipe: Parallel transient simulation of analog and digital circuits on multi-core shared-memory machines," in *Proc. DAC*, 2008, pp. 238–243.

[4] W. Dong and P. Li, "Parallelizable stable explicit numerical integration for efficient circuit simulation," in *Proc. DAC*, 2009, pp. 382–385.

[5] R. Griffith and M. Nakhla, "A new high-order absolutely-stable explicit numerical integration algorithm for the time-domain simulation of nonlinear circuits," in *Proc. ICCAD*, 1997, pp. 276–280.

[6] E. Lelarasmee, A. Ruehli, and A. Sangiovanni-Vincentelli, "The waveform relaxation method for time-domain analysis of large scale integrated circuits," *IEEE Trans. Comput.-Aided Des.*, vol. 1, no. 3, pp. 131–145, Jul. 1982.

[7] H. Peng and C.-K. Cheng, "Parallel transistor level full-chip circuit simulation," in *Proc. DATE*, 2009, pp. 304–307.

[8] J. E. Schutt-Ainé, "Latency insertion method (LIM) for the FST transient simulation of large networks," *IEEE Trans. Circuits Syst. I*, vol. 48, no. 1, pp. 81–89, Jan. 2001.

[9] T. Sekine and H. Asai, "Generalized leapfrog scheme for large-scale circuit simulation," in *Proc. EPEPS*, 2009, pp. 81–84.

[10] X. Ye, W. Dong, P. Li, and S. Nassif, "MAPS: Multi-algorithm parallel circuit simulation," in *Proc. ICCAD*, 2008, pp. 73–78.

[11] Z. Zhu, R. Shi, C.-K. Cheng, and E. S. Kuh, "An unconditional stable general operator splitting method for transistor level transient analysis," in *Proc. ASP-DAC*, 2006, pp. 428–433.

[12] Y. Saad, *Iteravite Methods for Sparse Linear Systems*. Philadelphia, PA: SIAM, 2003.

[13] T. A. Davis, *Direct Method for Sparse Linear Systems*. Philadelphia, PA: SIAM, 2006.

[14] J. Certaine, "The solution of ordinary differential equations with large time constants," *Math. Meth. Dig. Comp.*, 1960, pp. 129–132.

[15] G. Beylkin, J. M. Keiser, and L. Vozovoi, "A new class of time discretization schemes for the solution of nonlinear PDEs," *J. Comput. Phys.*, vol. 147, no. 2, pp. 362–387, 1998.

[16] Y. Ban, T. Endo, A. Yamamoto, and Y. Yamane, "Explicit time integration scheme using Krylov subspace method for reactor kinetics equation," *J. Nuclear Sci. Technol.*, vol. 48, no. 2, pp. 243–255, 2011.

[17] F. Aluffi-Pentini, V. De Fonzo, and V. Parisi, "A novel algorithm for the numerical integration of systems of ordinary differential equations arising in chemical problems," *J. Math. Chem.*, vol. 33, no. 1, pp. 1–15, 2003.

[18] C. Moler and C. V. Loan, "Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later," *SIAM Rev.*, vol. 45, no. 1, pp. 3–49, 2003.

[19] S.-H. Weng, P. Du, and C.-K. Cheng, "A fast and stable explicit integration method by matrix exponential operator for large scale circuit simulation," in *Proc. ISCAS*, 2011, pp. 1467–1470.

[20] S.-H. Weng, Q. Chen, and C.-K. Cheng, "Circuit simulation by matrix exponential method," in *Proc. ASICON*, 2011, pp. 462–465.

[21] C. Ho, A. Ruehli, and P. Brennan, "The modified nodal approach to network analysis," *IEEE Trans. Circuits Syst.*, vol. 22, no. 6, pp. 504–509, Jun. 1975.

[22] M. Hochbruck and C. Lubich, "On Krylov subspace approximations to the matrix exponential operator," *SIAM J. Numeric. Anal.*, vol. 34, no. 5, pp. 1911–1925, 1997.

[23] Y. Saad, "Analysis of some Krylov subspace approximations to the matrix exponential operator," *SIAM J. Numeric. Anal.*, vol. 29, no. 1, pp. 209–228, Feb. 1992.

[24] L. O. Chua and P. M. Lin, *Computer-Aided Analysis of Electronic Circuits*. Upper Saddle River, NJ: Prentice-Hall, 1975, ch. 8.

[25] Q. Nie, Y.-T. Zhang, and R. Zhao, "Efficient semi-implicit schemes for stiff systems," *J. Computat. Phys.*, vol. 214, no. 2, pp. 521–537, May 2006.

[26] Q. Chen, S.-H. Weng, and C.-K. Cheng, "A practical regularization technique for modified nodal analysis in large-scale time-domain circuit simulation," *IEEE Trans. Comput.-Aided Des.*, to be published.

[27] S. Natarajan, "A systematic method for obtaining state equations using MNA," *IEE Proc.-G Circuits, Devices Syst.*, vol. 138, no. 3, pp. 131–162, 1991.

[28] D. E. Schwarz and C. Tischendorf, "Structural analysis for electric circuits and consequences for MNA," *Int. J. Circ. Theor. Appl.*, vol. 28, no. 2, pp. 131–162, 1998.

[29] M. Güther and U. Feldmann, "The DAE-index in electric circuit simulation," *Math. Comput. Simul.*, vol. 39, nos. 5–6, pp. 573–582, 1995.

[30] S. Bächle and F. Ebert, "Graph theoretical algorithms for index reduction in circuit simulation," Inst. Mathematik, Technische Universität Berlin, Berlin, Germany, Tech. Rep. 245, 2005.

[31] D. Joyner, M. V. Nguyen, and N. Cohen. *Graph-Theory-Algorithms-Book* [Online]. Available: http://code.google.com/p/graph-theory-algorithms-book

[32] J. Niesen and W. M. Wright, "A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators," *ACM Trans. Math. Softw.*, to be published.

[33] A. H. Al-Mohy and N. J. Higham, "Computing the action of the matrix exponential, with an application to exponential integrators," *SIAM J. Sci. Comput.*, vol. 33, no. 2, pp. 488–511, 2011.

[34] N. Higham, "The scaling and squaring method for the matrix exponential revisited," *SIAM J. Matrix Anal. Applicat.*, vol. 26, no. 4, pp. 1179–1196, 2005.

[35] B. N. Bond. (2010). *SMORES: A Matlab Tool for Simulation and Model Order Reduction of Electrical Systems* [Online]. Available: http://bnbond.com/software/smores

[36] *HSPICE Simulation and Analysis User Guide, Version Y-2006.03*, Synopsys, Inc., Mountain View, CA, 2006.

Shih-Hung Weng (S'10) received the B.S. and M.S. degrees in computer science from National Tsing Hua University, Hsinchu, Taiwan, in 2006 and 2008, respectively. He is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, University of California at San Diego, La Jolla.

His current research interests include parallel circuit simulation, clock-gating optimization, and power-grid noise analysis in the very large-scale integration systems.

Quan Chen (S'09–M'11) received the B.S. degree in electrical engineering from Sun Yat-Sen University, Guangzhou, China, in 2005, and the M.Phil. and Ph.D. degrees in electronic engineering from the University of Hong Kong, Pokfulam, Hong Kong, in 2007 and 2010, respectively.

From 2010 to 2011, he was a Post-Doctoral Research Fellow with the Department of Computer Science and Engineering, University of California at San Diego, San Diego. He is currently a Post-Doctoral Research Fellow with the Department of Electrical and Electronic Engineering, University of Hong Kong. His current research interests include multiphysics and multiscale simulation, parallel computation in computer-aided design, and applied electromagnetics.

Chung-Kuan Cheng (S'82–M'84–SM'95–F'00) received the B.S. and M.S. degrees in electrical engineering from National Taiwan University, Taipei, Taiwan, and the Ph.D. degree in electrical engineering and computer sciences from the University of California at Berkeley, Berkeley, in 1984.

From 1984 to 1986, he was a Senior CAD Engineer with the Advanced Micro Devices, Inc., Sunnyvale, CA. In 1986, he joined the University of California at San Diego, San Diego, where he is currently a Professor with the Department of Computer Science and Engineering and an Adjunct Professor with the Department of Electrical and Computer Engineering. In 1999, he was the Chief Scientist with Mentor Graphics, Wilsonville, OR. He was an Honorary Guest Professor with Tsing Hua University, Hsinchu, Taiwan, from 2002 to 2008 and a Visiting Professor with National Taiwan University in 2011. His current research interests include medical modeling and analysis, network optimization, and design automation on microelectronic circuits.

Dr. Cheng was a recipient of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award in 1997 and 2002, the NCR Excellence in Teaching Award from the School of Engineering, University of California at San Diego, in 1991, and the IBM Faculty Award in 2004, 2006, and 2007. He was an Associate Editor of the IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN from 1994 to 2003.