UNIVERSITY OF CALIFORNIA, SAN DIEGO

Circuit Simulation via Matrix Exponential Method

A dissertation submitted in partial satisfaction of the
requirements for the degree Doctor of Philosophy
in
Computer Science (Computer Engineering)

by

Shih-Hung Weng

Committee in charge:

Professor Chung-Kuan Cheng, Chair
Professor Li-Tien Cheng
Professor Fan Chung Graham
Professor Bo Li
Professor Bill Lin

2013

.

Dedication

To my parents, Jon-Yi Weng and Li-Hua Wu

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

Four years ago, I left my country, Taiwan, to the United States alone to pursue my Ph.D. degree. During the journey of my Ph.D., I was first frustrated, then encouraged, and finally made a breakthrough. I cannot overcome those difficulties in both research and life without helps, supports and encouragements from all the people. The Ph.D. dissertation could not complete in such short four-year period without them. I would like to express my sincere gratitude here to people who makes the journey wonderful.

First of all, I would like to thank my adviser, Professor Chung-Kuan Cheng, who guided me a clear path in research. I learned not only the way conducting high quality research, but also passion, perseverance and humility from his personality. I always remember the days when Professor Cheng and I were in National Taiwan University as a visiting scholar and student. During that time, we were in the same office and polished some ideas everyday. All lessons learned from Professor Cheng will be my invaluable wealth of my life.

Secondly, I would also like to thank Professor Li-Tien Cheng, Professor Fan Chung-Graham, Professor Bo Li and Professor Bill Lin who serve as my Ph.D. committee members. They provided many valuable suggestions for my research and improved the quality of the dissertation.

Furthermore, I am grateful to all the students in Professor Cheng's group, including Ryan Coutts, Peng Du, Xiang Hu, Jingwei Lu, He Peng, Amirali Shayan, Renshen Wang, Xiang Zhang, Yulei Zhang, Hao Zhuang and many other students and research fellows. We shared different research ideas and collaborated on different projects. It is my pleasure to work with them. Your brilliant minds encouraged and helped me during the journey. I especially want to thank Quan Chen, who collaborated with me to conduct this research and provided me many insightful advices.

Finally, I owe my deepest gratitude to my family. They have been supportive along this journey. I also want to thank Annie, Paul, Jessica and Clifford for their taking care when I first came to the states. The dissertation is dedicated to them.

Chapter III, in part, is a reprint of the material as it appears in "Time-Domain Analysis of Large-Scale Circuits by Matrix Exponential Method with Adaptive Control," by Shih-Hung Weng, Quan Chen and Chung-Kuan Cheng, in *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems.* The dissertation author was the primary investigator and author of the paper.

Chapter IV, in part, is a reprint of the material as it appears in "Circuit Simulation using Matrix Exponential Method for Stiffness Handling and Parallel Processing," by Shih-Hung Weng, Quan Chen and Chung-Kuan Cheng, in the proceeding of *IEEE Int. Conf. on Computer-Aided Design, 2012.* The dissertation author was the primary investigator and author of the paper.

Chapter V in part, is a reprint of as it will appear in "Adaptive Time Stepping for Power Grid Simulation using Matrix Exponential Method," by Shih-Hung Weng, Hao Zhuang and Chung-Kuan Cheng, which has been submitted to *IEEE Int. Conf. on Computer-Aided Design, 2013.* The dissertation author was the primary investigator and author of the paper.

VITA

2006	B.S. in Computer Science National Tsing Hua University, Hsinchu, Taiwan.
2008	M.S. in Computer Science National Tsing Hua University, Hsinchu, Taiwan.
2013	Ph.D. in Computer Science (Computer Engineering) University of California San Diego, La Jolla, CA., U.S.A.

PUBLICATIONS

S.-H. Weng, H. Zhuang and C. K. Cheng, "Adaptive Time Stepping for Power Grid Simulation using Matrix Exponential Method," *Proceedings of IEEE Int. Conf. on Computer-Aided Design (ICCAD 2013)*, submitted.

X. Hu, P. Du, S.-H. Weng and C. K. Cheng, "Worst-Case Noise Prediction With Non-zero Current Transition Times for Power Grid Planning," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, accepted.

S.-H. Weng, Y. Zhang, J. F. Buckwalter and C. K. Cheng, "Eergy Efficiency Optimization through Co-Design of the Transmitter and Receiver in High-Speed On-Chip Interconnects," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, accepted.

C.-C. Chou, H.-H. Chuang, T.-L. Wu, S.-H. Weng, and C. K. Cheng, "Eye Prediction of Digital Driver with Power Distribution Network Noise", *Proceedings of IEEE Electrical Performance of Electronic Packaging and Systems (EPEPS 2012)*

S.-H. Weng, Q. Chen and C. K. Cheng, "Circuit Simulation using Matrix Exponential Method for Stiffness Handling and Parallel Processing," *Proceedings of IEEE Int. Conf. on Computer-Aided Design (ICCAD 2012)*

Q. Chen, W. Schoenmaker, S.-H. Weng, C. K. Cheng, G.-H. Chen, L.-J. Jiang and N. Wong, "A Fast Time-Domain EM-TCAD Coupled Simulation Framework via Matrix Exponential," *Proceedings of IEEE Int. Conf. on Computer-Aided Design (ICCAD 2012)*

Y.-C. Li, Q. Chen, S.-H Weng, C. K Cheng and N. Wong, "Globally Stable, Highly Parallelizable Fast Transient Circuit Simulation via Faber Series," *Proceedings of IEEE Int. New Circuits and Systems Conf. (NEWCAS 2012)*

G. Sun, S.-H. Weng, C. K. Cheng, B. Lin and L. Zeng, "An On-Chip Global Broadcast Network Design with Equalized Transmission Lines in the 1024-Core Era," *Proceedings of System Level Interconnect Prediction (SLIP 2012)*

S.-H. Weng, Q. Chen and C. K. Cheng, "Time-Domain Analysis of Large-Scale Circuits by Matrix Exponential Method with Adaptive Control" *IEEE Trans. on Computer-Aided Design of Integrated Circuit and Systems (CAD)*, Aug., 2012

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. on Computer-Aided Design of Integrated Circuits and Systems (CAD)*, Jul. 2012

C. K. Cheng, P. Du, A.B. Kahng and S.-H. Weng, "Low-Power Gated Bus Synthesis for 3D IC via Rectilinear Shortest-path Steiner Graph," *Proceedings of ACM Int. Symposium of Physical Design (ISPD 2012)*

P. Du, W. Zhao, S.-H. Weng, C. K. Cheng and R. L. Graham, "Character Design and Stamp Algorithms for Character Projection Electron-Beam Lithography," *Proceedings of IEEE Asia and South Pacific Design Automation Conf. (ASPDAC 2012)*

S.-H Weng, Y.-M. Kuo and S.-C. Chang, "Timing Optimization in Sequential Circuit by Exploiting Clock-Gating Logic," *ACM Trans. on Design Automation of Electronic Systems (DAES)*, April 2012.

P. Du, S.-H. Weng, X. Hu and C. K. Cheng, "Power Grid Sizing via Convex Programming," *Proceedings of IEEE Int. Conf. on ASIC (ASICON 2011)*

S.-H. Weng, Q. Chen and C. K. Cheng, "Circuit Simulation by Matrix Exponential Method," *Proceedings of IEEE Int. Conf. on ASIC (ASICON 2011)*

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" *Proceedings of IEEE Symposium on Circuits and Systems (ISCAS 2011)*

P. Du, X. Hu, S.-H. Weng, A. Shayan, X. Chen, A. E. Engin and C. K. Cheng, "Worst-Case Noise Prediction with Non-zero Current Transition Times for Early Power Distribution System Verification," *Proceedings of IEEE Int. Symposium on Quality Electric Design (ISQED 2010)*

Y.-M. Kuo, S.-H. Weng, and S.-C. Chang, "A Novel Sequential Circuit Optimization with Clock Gating Logic," *Proceedings of IEEE Int. Conf. on Computer-Aided Design (ICCAD 2008)*

S.-H. Weng, Y.-M. Kuo, S.-C. Chang, and M. Marek-Sadowska, “Timing Analysis Considering IR Drop Waveforms in Power Gating Designs,” *Proceedings of IEEE Int. Conf. on Computer Design (ICCD 2008)*

FIELDS OF STUDY

Major Field: Computer Science (Computer Engineering)
    Studies in VLSI CAD
    Professor Chung-Kuan Cheng

ABSTRACT OF THE DISSERTATION

Circuit Simulation via Matrix Exponential Method

by

Shih-Hung Weng
Doctor of Philosophy in Computer Science (Computer Engineering)
University of California, San Diego, 2013
Professor Chung-Kuan Cheng, Chair

The trend of higher integration density in VLSI has made the time-domain circuit simulation a bottleneck in today's IC design flows. For designs with millions of elements, SPICE-like simulation 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. A different aspect of solving the simulation problem is proposed in this dissertation.

The proposed method, called Matrix Exponential Method (MEXP), is based on the analytical solution of the ordinary different equations. The MEXP method has the benefits of accuracy, stability, scalability, parallelizability and adaptivity due to its analytical nature and simple kernel computation. To utilize the adaptivity of MEXP, an algorithm is proposed to dynamically determine locally optimal step size and order of Krylov subspace at each time step to reduce the overall computation in the simulation.

Moreover, the scaling effect of MEXP is observed. The scaling effect enables MEXP to simulate a circuit with larger step size as the time instant moves forward. With the scaling effect, MEXP demonstrates the comparable capability of stiffness handling as the widely adopted backward Euler and trapezoidal methods. The parallelism of MEXP is also demonstrated. The GPU implementation is shown in this dissertation. The restarted MEXP is also proposed for GPU implementation to mitigate the memory usage on resource restricted environment.

Finally, the rational Krylov subspace method is investigated. Instead of using $\mathbf{A}$ as the basis, the rational Krylov subspace utilizes $(\mathbf{I}-\gamma\mathbf{A})^{-1}$ as the basis. The rational basis transforms the spectrum of a matrix to relax the stiffness constraint in the Krylov subspace method. Furthermore, the rational Krylov subspace can avoid the regularization process of singular $\mathbf{C}$. This dissertation applies MEXP with the rational Krylov subspace on the power distribution network (PDN) design. Combining with the capability of adaptivity and accuracy of MEXP, MEXP with the rational Krylov subspace could exploit a large step size in the low-frequency response to further improve the performance of the power grid simulation.

I

Introduction

I.1 Circuit Simulation

Circuit simulation plays an important role in the integrated circuit (IC) design flow. Many design tools rely on the pre-characterized logic gates by the circuit-level simulation for *logic synthesis*, *placement*, *timing analysis* and *routing*. The electronic behavior of a pre-characterized gate will be stored as a look-up table such that design tools can quickly obtain information of power, timing, parasitic resistance and capacitance. With such abstraction provided by the circuit simulation, design tools can reduce the complexity and improve the overall performance.

Moreover, with continuous scaling of the technology, the complicated interaction among nano-scale transistors and interconnects causes unpredictable electronic behavior, e.g., power noise and signal noise. A pre-characterized single gate is unable to contain those interactions into a look-up table. Hence, a nano-scale design requires the help from the circuit simulation to analyze and verify the potential hazards before taping out for manufacturing. Nowadays, the circuit simulation is an essential procedure from the front-end stage (e.g., logic synthesis) to the back-end stage (e.g., post-layout verification or power distribution network verification).

Figure I.1 shows an overall flow of the circuit simulation, which includes two major operations—*device evaluation* and *numerical integration*. The device evaluation is first to calculate induced charges and currents of transistors given the nodal voltages from the previous time instant. The evaluation is based on either empirical equations fitted from measurement data, e.g., BSIM3 [16], or real physical formulation, e.g., surface potential model [32]. After the evaluation process, the operation will linearize all those devices to compute equivalent resistance and capacitance from currents and charges. With those resistance and capacitance, a circuit can be formulated into a system of ordinary differential equations (ODEs). Note that this stage is parallel ready because each device can be evaluated independently.

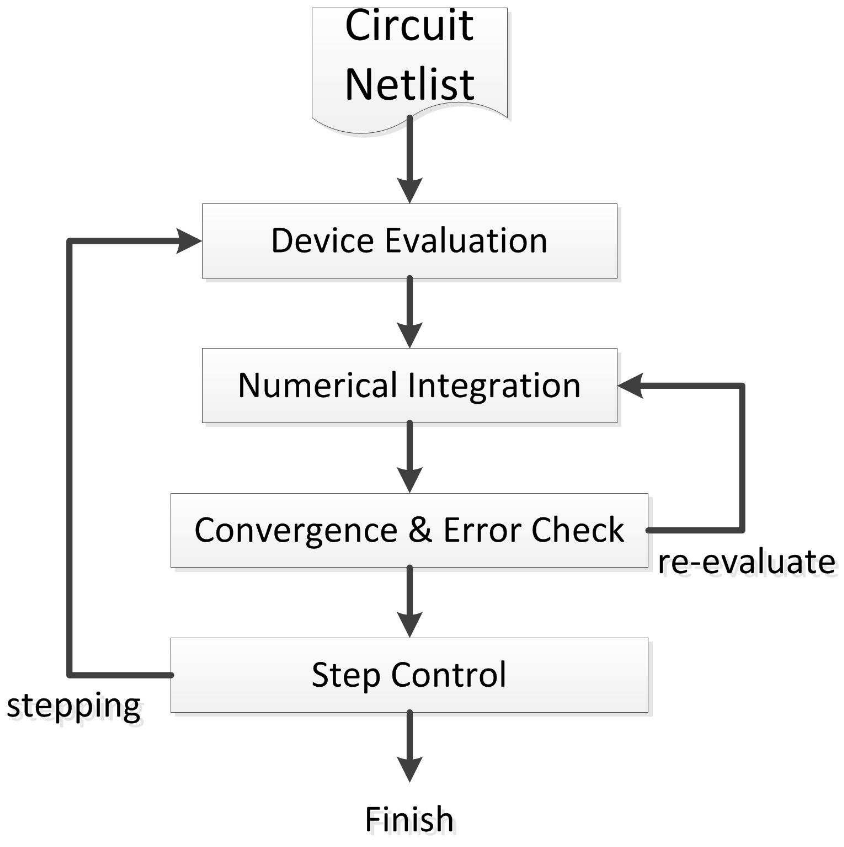


Figure I.1: Circuit simulation flow.

The numerical integration operation will solve the ODEs formed from previous stage with a chosen step size. Then, we can obtain the nodal voltages at the present time instant. The solving technique in the numerical integration can be classified into implicit and explicit methods. Implicit methods require solving a linear system but have better stability and can march large step size. On the other hand, explicit methods take only simple sparse matrix-vector multiplication but the poor stability drags the overall performance with the demand of a tiny step size. Even though implicit methods are more practical for the circuit simulation, the size of unknowns in a circuit could up to billions that makes solving a linear system impossible.

The *convergence and error check* component of the flow will verify 1) if the solution from implicit numerical integration converges, and 2) if the numerical error from previous two operations is under a certain threshold. Otherwise, the simulation will reverse back to the numerical integration process to solve the ODE again with a shrunk step size for less the numerical error.

Finally, the *step control* component will increase or reduce the step size for the next time step according to the numerical error. The flow will step into the next time instant with new nodal voltages until it reaches the end of simulation time.

Although the necessity of the circuit simulation has emerged in the deep sub-micron era, the device evaluation and the numerical integration processes have become prohibitive tasks, which consume days or even weeks to complete, due to the ever increasing size of circuitry in the nano-scale process technology. Even if the device evaluation can be parallelized in a multi-core or GPGPU environments, the serial nature of the numerical integration, which involves solving ODEs with billion unknowns, still restrains the performance of the circuit simulation. Such performance limitation makes the circuit simulation a bottleneck of the overall design and verification flow. Therefore, efficient yet accurate circuit simulation has always been one of the major demands in industry. In this dissertation, we will focus on how to accelerate the numerical integration operation of the circuit simulation.

I.2 Current Research Efforts

There are two directions to accelerate the circuit simulation. One is to speedup the performance of the device evaluation, which is often relied on parallel techniques. The other aspect is to improve the numerical integration operation to reduce the time of solving the solution or to increase allowed step size during the simulation.

Many researchers [33] [41] [54] [58] [59] [69] [74] [76] [77] worked on acceleration of device evaluation. The idea is to distribute the devices into different cores, machines or threads. Since each device is independent, the evaluation process can be performed simultaneously. Researchers [33] [76] [77] utilized the multi-cores environment in GPGPU to massively evaluate devices at the same time. On the other hand, researchers [58] [59] [54] [69] [74] took more general purpose cluster machine to parallize the evaulation. The cluster machine usually has more cores than GPGPU. However, the communication overhead on the ethernet will decrease the speedup while the on-chip communication within GPGPU is negligible. Kapre *et al.* implemented the device evaluation onto FPGA and also compared the performance of device evaluation over cluster, multi-core processor and GPGPU environments.

In terms of numerical integration, those proposed numerical approaches could be classified into explicit and implicit methods. Schutt-Ainé [60] proposed an explicit method to avoid matrix inverse by alternatively updating voltages and currents with Kirchhoff's Circuit and Voltage Laws. The coupling capacitor is regarded as a branch by the companion model [18]. To ensure both equations are not divided by zero, every node and branch must have branch inductances and grounded capacitances. That means a small latency has to be introduced among elements to enable alternative updating between voltage and current. Although such *latency insertion method* avoids matrix inverse, those fictitious elements cause either stability issue (with too small values) or inaccurate result (with too large values).

In order to improve the stability of the explicit method, Dong and Li [22] [23] utilized *telescopic projective integration* to enable large step size in the explicit methods without instability. The method is first to march several smaller steps to avoid the unstable result and then "project" one large step size using extrapolation. Gear and Kevrekidis [31] proved the stability region is extended with such scheme. The ratio of the small and the large step size, however, has to be less than 3, and thus, the overall speedup is limited. To improve the speedup, a multi-level projection is devised. Devgan and Rohrer [20] [21] utilized the concept of the steady state and avoid calculating those steady stiff elements for improving the stability. Therefore, the stiffness of circuit does not affect the computational cost because once a stiff element enters the steady state; it no longer controls the step size. However, both techniques in [20] [21] [22] [23] are still limited because 1) projected step size is still relatively small compared to the allowed step size in the implicit methods and 2) the steady state of stiff elements can only observed once the latency among the is large enough. Overall, the stability as well as limited step size are still challenges in explicit methods for the circuit simulation.

Researchers [30] [39] [44] [54] [65] proposed circuit partitioning technique to reduce a large-scale matrix into several matrices with size of tens of thousands, which can be solved within a reasonable time by the LU decomposition. By the concept of divide and conquer, the overall simulation performance is accelerated under affordable memory requirement. Furthermore, the partitioned sub-circuits also enable the ability of parallelization. To reduce the size of interface between sub-circuits and balance sizes of partitions, the circuit simulators TITAN [30] and Xyce [39] adopted the concept of placement in the physical design to develop an analytic partitioning method [55]. The circuit is transformed into a hyper-graph and partitioned by several “cuts”. The analytic partitioning method is to minimize the hyper-edges through cuts and to balance the number of vertices of groups such that the interconnect part (cuts) can be minimized, and sub-circuits (groups) are balanced. The waveform relaxation technique proposed by [44] simulates the sub-circuit individually for whole time period and then updates the interface values of a sub-circuit by the approximated waveforms of its adjacent sub-circuits iteratively. The initial interface waveforms of a sub-circuit are assumed to be the same as DC values, i.e., no transient behavior. Under the waveform relaxation technique, the circuit partitioning method has to exploit latency among sub-circuits so that the iteration number can be reduced. However, the convergence of nonlinear circuit cannot be guaranteed, and it is sensitive to the partitioning method and the order of solving sub-circuits. Researchers [54] [65] adopted overlapping sub-circuits partitioning in domain decomposition technique. The overlapping sub-circuits can be solve by the iterative methods, e.g., GMRES, with preconditioner (*additive Schwarz* preconditioner). The overlapping partitioning improves the convergence rate of the iterative method for solving matrix and mitigates effect of solving order. However, a circuit usually contains feed back loop, e.g., flip-flop in a sequential circuit. The feed back loop will increase the number of iterations and even unable to converge. The partition-based circuit simulation performs well only to certain type of circuit.

I.3 Dissertation Outline

Chapter II introduces the background of numerical integration in the circuit simulation application. The basic concepts and properties of numerical integration methods are briefly presented. Two basic yet widely adopted implicit and explicit methods in the circuit simulation are also shown.

Chapter III tackles the scalability and stability issues in today’s numerical integration approaches in the circuit simulation. The matrix exponential method is presented. The proposed method is free from the stability constraint, but does still preserve the highly scalable sparse matrix-vector multiplication operation of conventional explicit methods. The computation of the matrix exponential is via Krylov subspace approximation, which can significantly reduce the computational effort by projecting into a much smaller space. We also develop an adaptive step size scheme to accelerate the performance.

Chapter IV investigates and mitigates the issue of stiffness of an electronic circuit. We show that the Krylov subspace approximation has the *scaling effect*, and the step size can be enlarged after few steps. We also demonstrate the parallelism in the matrix exponential method in the GPGPU environment.

In Chapter V, we study the rational Krylov subspace method. The matrix exponential method with the Krylov subspace method could significantly relax the stiffness constraint and eliminate the regularization process. We demonstrate that the matrix exponential method with the rational Krylov subspace especially fits for the power grid simulation. The adaptivity of the matrix exponential method can exploit the low-frequency response of the power grid design to speedup the simulation.

Chapter VI summarizes the main contributions of the dissertation. Future research directions are also discussed.

II

Background of Numerical Integration in Circuit Simulation

In this chapter, we first present the mathematical formulation for the circuit simulation as well as the related background of numerical integration. Then, we show two widely-adopted implicit and explicit methods, i.e., backward Euler and forward Euler, in the circuit simulation application.

II.1 Circuit Formulation

In general, we can formulate a circuit into a system of differential algebraic equations as below

$$\dot{\mathbf{q}}(\mathbf{x}(t)) + \mathbf{C}^l\dot{\mathbf{x}}(t) = \big(-\mathbf{G}^l\mathbf{x}(t) + \mathbf{i}(\mathbf{x}(t))\big) + \mathbf{B}\mathbf{u}(t), \tag{II.1}$$

where sparse matrices $\mathbf{C}^l$ and $\mathbf{G}^l$ describe the linear capacitances, inductances and conductances, $\mathbf{B}$ indicates the locations of independent input sources, and $\mathbf{q}$ and $\mathbf{i}(t)$ denote the charges and currents induced by the nonlinear components, e.g., MOSFET or diode. Vector $\mathbf{x}(t)$ is the nodal voltages and branch currents at time t, and $\mathbf{u}(t)$ is the input voltage and current sources. In the device evaluation operation, we will linearize nonlinear devices, i.e., $\mathbf{q}$ and $\mathbf{i}$, into their companion models [18], and use modified nodal analysis (MNA) [37] to construct a circuit as a system of ODEs:

$$\mathbf{C}\dot{\mathbf{x}}(t) = -\mathbf{G}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{II.2}$$

where $\mathbf{C}$ includes the linear capacitors, linear inductors, and effective capacitance of $\mathbf{q}$, and $\mathbf{G}$ represents the linear resistors, effective resistance from $\mathbf{i}$ and the incidence between voltages and currents. Note that the simulator will apply Newton's method to the linearized nonlinear circuit to solve for the solution iteratively. The linearization process during the device evaluation stage is actually the first order derivative of nonlinear devices with respect to voltage, which is called Jacobian matrix. The required Jacobian matrix can be constructed by device-wise inspection.

Given the initial value $\mathbf{x}(0)$, which can be obtained by DC analysis, Eqn. (II.2) can be solved numerically. First, we discretize the continuous time span $[t_0, t_f]$ into several discrete time instants:

$$t_0 < t_1 < t_2 < \cdots < t_{n-1} < t_n < \cdots < t_f.$$

Then, instead of directly solving the ODE for $\mathbf{x}(t)$, we could solve the ODE implicitly or explicitly in a step-by-step fashion. A method is call implicit when the solution of current time instant depends on itself; otherwise, it is called the explicit method. For example, provided $\mathbf{x}_n$ is given, the following is a first-order explicit method (forward Euler):

$$\mathbf{x}_{n+1} = \mathbf{x}_n + h\dot{\mathbf{x}}_n + O(h^2), \tag{II.3}$$

where $\mathbf{x}_n \approx \mathbf{x}(t_n)$, h is the step size from t_n to t_{n+1}, and $\dot{\mathbf{x}}_n$ is the approximated derivative at time t_n.

The numerical error of the above approximation is $O(h^2)$, which is referred as *local truncation error* (LTE). Any numerical method based on polynomial expansion, e.g., Taylor's expansion, will generate such error. A higher order expansion can result in a more accurate result. Since the LTE depends on the step size, simulators usually can control the LTE by changing step size h adaptively such that

$$LTE \leq h\frac{E}{T},$$

where E is a user-defined error for entire simulation process, and T is the total simulation time. By doing so, the simulation can be accelerated using a large step size when nodal voltages are steady, and can still maintain the accuracy at rapid fluctuation period with a small step size.

In addition to LTE, we need to consider the *stability* issue of numerical methods. An unstable numerical method will amplify the error at time t_n to indefinitely large when n steps to infinity. Hence, even though we could calculate $\mathbf{x}_{n+1}$ with high-order numerical methods to have a smaller LTE, the stability of a method still limits the step size. We illustrate this issue with a test function as follows:

$$\dot{x}(t) = \lambda x(t) \quad , \quad x(0) = 1, \tag{II.4}$$

where λ is a constant complex number. Using the forward Euler method, we have the relation between x_{n+1} and x_0.

$$\begin{aligned} x_{n+1} &= x_n + h\lambda x_n \\ &= (1 + h\lambda)x_n \\ &= (1 + h\lambda)^{n+1}x_0. \end{aligned} \tag{II.5}$$

In order to maintain the stability, $1 + h\lambda$ has to be less than 1. As a result, the implicit methods generally have better stability than the explicit methods. For example, with the same test function, the stability constraint of backward Euler, which can be derived by approximating $\dot{\mathbf{x}}(t)$ as $\dot{\mathbf{x}}_{n+1}$, is $(\frac{1}{1-h\lambda})^n \leq 1$. The stability regions of both forward and backward Euler methods are shown in Figures II.1 and II.2, respectively. Note that the backward Euler method is also an *absolutely stable* (A-stable) method because it is stable as long as λ is negative. As we can see, the backward Euler method has much larger stability region than the forward Euler.

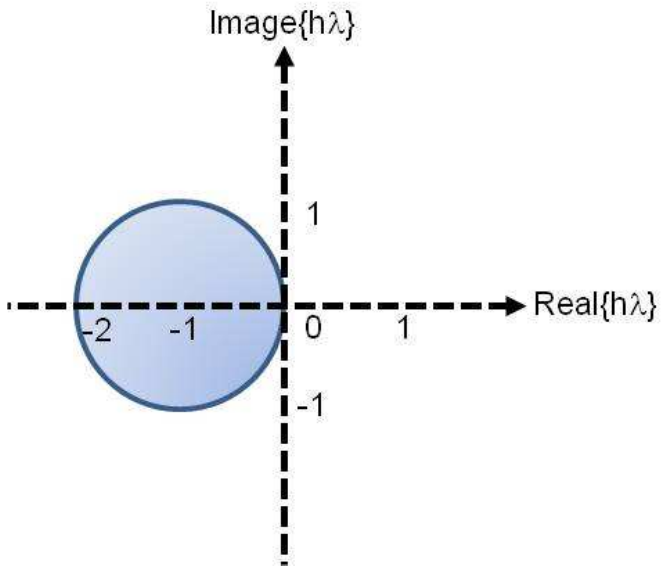


Figure II.1: Stability region of forward Euler.

Most of modern circuit simulators use implicit methods because the ODE system of circuit is usually *stiff*, i.e., the magnitude of elements in a circuit varies in a wide range. For example, a capacitor usually ranges from 10^{-16} to 10^{-12}. The implicit methods, e.g., the backward Euler and the trapezoidal methods, have a larger stability region that can handle the stiff ODE system. In addition, the implicit methods used in simulators can only up to second-order because lower-order implicit methods usually have larger stability range as the stability region decreases with higher order.

II.2 Implicit and Explicit Methods

The implicit methods have better stability to handle the stiffness of the circuit so that they could use larger step size, but the complicated computation involved in these methods poses performance and scalability issues. The backward

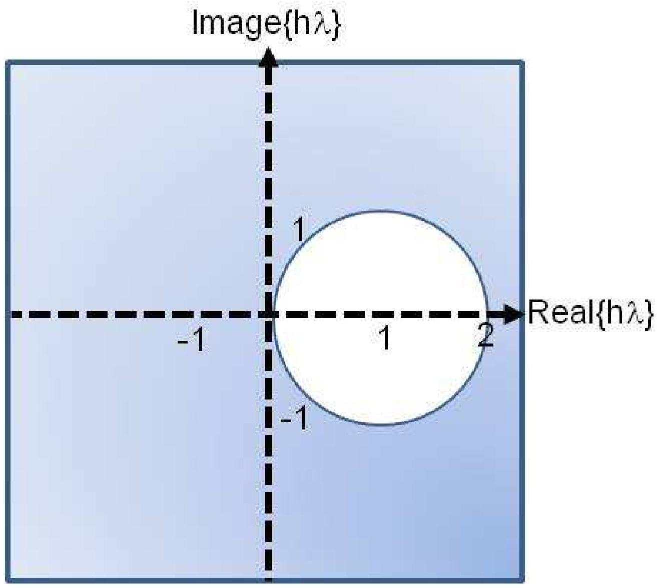


Figure II.2: Stability region of backward Euler.

Euler for the circuit simulation is expressed as follows.

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n + h\dot{\mathbf{x}}_{n+1} \\ &= \mathbf{x}_n + h\mathbf{C}^{-1}(-\mathbf{G}\mathbf{x}_{n+1} + \mathbf{B}\mathbf{u}_{n+1}) \\ &= \left(\frac{\mathbf{C}}{h} + \mathbf{G}\right)^{-1}\left(\frac{\mathbf{C}}{h}\mathbf{x}_n + \mathbf{B}\mathbf{u}_{n+1}\right).\end{aligned} \tag{II.6}$$

The trapezoidal method is also widely adopted in the circuit simulation. Its equation for the circuit simulation is expressed as below

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n + \frac{h}{2}\left(\dot{\mathbf{x}}_{n+1} + \dot{\mathbf{x}}_n\right) \\ &= \mathbf{x}_n + \frac{h}{2}\mathbf{C}^{-1}(-\mathbf{G}\mathbf{x}_{n+1} + \mathbf{B}\mathbf{u}_{n+1} - \mathbf{G}\mathbf{x}_n + \mathbf{G}\mathbf{u}_n) \\ &= \left(\frac{2\mathbf{C}}{h} + \mathbf{G}\right)^{-1}\left(\left(\frac{2\mathbf{C}}{h} - \mathbf{G}\right)\mathbf{x}_n + \mathbf{B}(\mathbf{u}_{n+1} + \mathbf{u}_n)\right).\end{aligned}$$

As we can see in the above backward Euler or trapezoidal methods for the circuit simulation, calculating $\mathbf{x}_{n+1}$ involves solving $(\mathbf{C}/h + \mathbf{G})$ or $(2\mathbf{C}/h + \mathbf{G})$ that usually requires a LU decomposition. For the circuit simulation, the time and space complexities of LU decomposition [19] are $\mathrm{O}(n^{1.5})$ where n is number of unknowns.

Such complexities lead to the scalability problem in terms of runtime and memory for the implicit methods when the number of unknowns could up to billions in today's design. Furthermore, the LU decomposition operation is required for every iterative of Newton's method in the nonlinear circuit. Despite the impressive achievements in iterative matrix solving methods [57], the ill-conditioned matrix constructed from the implicit methods significantly degrades the performance of the iterative technique. Another drawback in the implicit methods is the costly adaptive step control. Once the step size h changes, the LU factorization for $(\mathbf{C}/h + \mathbf{G})$ has to be re-evaluated and thus degrades the overall efficiency of the simulation.

The explicit methods usually require only the sparse matrix-vector multiplication when performing numerical integration because $\mathbf{x}_{n+1}$ depends only on the solution at previous time instant. However, in the circuit simulation application, the following forward Euler equation derived from Eqn. (II.3) shows that $\mathbf{x}_{n+1}$ still needs to solve $\mathbf{C}^{-1}$.

$$\begin{aligned}\mathbf{x}_{n+1} &= \mathbf{x}_n + h\dot{\mathbf{x}}_n \\ &= \mathbf{x}_n + h\mathbf{C}^{-1}(-\mathbf{G}\mathbf{x}_n + \mathbf{B}\mathbf{u}_n) \\ &= (\mathbf{I} - h\mathbf{C}^{-1}\mathbf{G})\mathbf{x}_n + h\mathbf{C}^{-1}\mathbf{B}\mathbf{u}_n. \qquad \text{(II.7)}\end{aligned}$$

Although $\mathbf{C}$ usually has better structure, which leads to a faster LU factorization or a faster convergence rate in the iterative methods, such extra cost still undermines the efficiency of the explicit methods. Note that although $\mathbf{C}$ is possible to be singular, previous works [6] [14] have proposed several approaches to regularize $\mathbf{C}$ with affordable cost while maintaining the sparsity.

Another issue of the explicit methods is the stability. Although the computation of the explicit method is simple and scalable, this method suffers from the stability issue. As we can see in Eqn. (II.5), the step size h is limited to make sure $h\lambda$ is less than 1, where λ in the circuit simulation application relates to the smallest time constant. For an electronic circuit, the time constant usually is around

$10^{-16} \sim 10^{-18}$. Hence, the step size of the explicit methods must be small enough to keep the methods stable. Thus, the explicit methods take more time steps to perform the simulation and the benefit of simple computation is deteriorated.

III

Matrix Exponential Method with Adaptive Step Control

In this chapter, 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.

III.1 Background

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. 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 semi-analytical 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 dated back to 1960s [13] and has been "re-invented" over the years in some areas, such as computational physics [10] [7] and chemistry [3]. 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 [45], 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 $\mathbf{A}$ [45].

In this chapter, 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) [71] [70]. Our method directly computes the analytical solution of the differential equations resulting from modified nodal analysis (MNA) [37]. We utilize the Krylov subspace method [38] [56] to approximate the matrix exponential operator, which significantly reduces the complexity of matrix 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 chapter for adaptive control of the step size and the order of Krylov subspace approximation during the numerical integration.

III.2 Matrix Exponential Formulation

In this section, we present the primary formulation of MEXP in linear and nonlinear circuit simulation.

III.2.A Linear Circuit

In MNA, a linear circuit is represented by a system of linear differential algebraic equations (DAEs) as Eqn (II.2). Provided $\mathbf{C}$ is invertible, (II.2) is reducible to a system of ordinary differential equations (ODEs). Given an initial condition $\mathbf{x}(0)$ of the circuit (e.g., from DC analysis), one can obtain the analytical solution [17] of (II.2) 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 $\mathbf{C}^{-1}\mathbf{G}$ in the Arnoldi process), 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

$$\mathbf{x}(t+h) = e^{\mathbf{A}h}\mathbf{x}(t) + \int_0^h e^{\mathbf{A}(h-\tau)}\mathbf{b}(t+\tau)d\tau. \tag{III.1}$$

Following the convention of SPICE-like simulators, we assume that the given input $\mathbf{u}(t)$ is piece-wise linear (PWL), i.e., $\mathbf{u}(t)$ is linear within every time step. The integral term in (III.1) can be computed analytically, turning (III.1) to (III.2) involving three functions with matrix exponential operators

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

We call the solution scheme based on this formulation as *matrix exponential method.* 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 (II.2) analytically provided the matrix exponential is computed exactly and the PWL assumption of input is satisfied. This is of theoretical difference from linear multi-step methods, such as the forward, backward Euler and trapezoidal methods. These methods approximate (II.2) by polynomial expansion and drop high-order terms, which is the source of LTE. Therefore, for linear circuits, the step size of matrix exponential method is not restricted by LTE or stability, but instead solely by the computation error of matrix exponential, which will be detailed in Section III.4.

III.2.B Nonlinear Circuit

For nonlinear circuits, the differential equation is given as Eqn. (II.1). 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 (II.2)

$$\mathbf{C}_n\dot{\mathbf{x}}(t) = -\left(\mathbf{G}^l\mathbf{x}(t) + \mathbf{i}(\mathbf{x}(t))\right) + \mathbf{B}\mathbf{u}(t) \tag{III.3}$$

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 (III.3) is therefore in an analogous form with (III.1)

$$\begin{aligned} \mathbf{x}(t_n + h) &= e^{\mathbf{A}_n h}\mathbf{x}(t_n) \\ &+ \int_0^h e^{\mathbf{A}_n(h-\tau)} \left[\mathbf{F}(\mathbf{x}(t_n+\tau)) + \mathbf{b}(t_n+\tau)\right] d\tau, \end{aligned} \tag{III.4}$$

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 $\mathbf{F}(\tau) = \left(\mathbf{F}(\mathbf{x}_n) + \mathbf{F}(\mathbf{x}_{n+1})\right)/2$, which is A-stable [10], leads to an algebraic nonlinear system

$$\begin{aligned} \mathbf{x}_{n+1} = &\frac{(e^{\mathbf{A}h} - \mathbf{I})}{2}\mathbf{A}^{-1}\mathbf{F}\left(\mathbf{x}_{n+1}\right) + e^{\mathbf{A}h}\mathbf{x}_n \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\left(\frac{\mathbf{F}_n}{2} + \mathbf{b}_n\right) \\ &+ (e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I})\mathbf{A}^{-2}\Delta\mathbf{b}_n, \end{aligned} \tag{III.5}$$

where $\Delta\mathbf{b}_n = \frac{\mathbf{b}_{n+1} - \mathbf{b}_n}{h}$, $\mathbf{x}_{n+1}$ is $\mathbf{x}(t+h)$, and $\mathbf{x}_n$ is $\mathbf{x}(t)$. 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 re-evaluated by the Krylov subspace approximation in every iteration.

To decouple the linear and nonlinear terms in the above equation, we adapt the scheme developed in [51], 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 (III.4). The second order implicit approximation is then of the form

$$\begin{aligned} \mathbf{x}_{n+1} = &\frac{h}{2}\mathbf{F}\left(\mathbf{x}_{n+1}\right) + e^{\mathbf{A}h}\left(\mathbf{x}_n + \frac{h}{2}\mathbf{F}_n\right) \\ &+ (e^{\mathbf{A}h} - \mathbf{I})\mathbf{A}^{-1}\mathbf{b}_n \\ &+ (e^{\mathbf{A}h} - \mathbf{A}h - \mathbf{I})\mathbf{A}^{-2}\Delta\mathbf{b}_n, \end{aligned} \tag{III.6}$$

in which an upper bound of the local truncation error is estimated as

$$-\frac{h^3}{12}\left((\mathbf{A}h)^2\mathbf{F}_n + (\mathbf{A}h)\dot{\mathbf{F}}_n + \ddot{\mathbf{F}}_n\right) \tag{III.7}$$

In (III.6), the nonlinear function of $\mathbf{x}_{n+1}$ is only multiplied by a scalar coefficient, other than the matrix exponential function in (III.5). 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.

To provide higher capability for handling nonlinearity, we utilize Newton's method [53] to solve Eqn. (III.6), which can be rearranged as

$$\mathbf{x}_{n+1} - \frac{h}{2}\mathbf{F}(\mathbf{x}_{n+1}) - \mathbf{L}_n = 0, \tag{III.8}$$

where

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

$\mathbf{L}_n$ includes terms only related to current time t. By multiplying $\mathbf{C}_n$ at both sides of the above equation, we have a nonlinear function $\mathbf{f}(\mathbf{x})$ as below

$$\mathbf{f}(\mathbf{x}_{n+1}) = \mathbf{C}_n\mathbf{x}_{n+1} - \frac{h}{2}\mathbf{i}(\mathbf{x}_{n+1}) - \mathbf{C}_n\mathbf{L}.$$

In Newton's method, solution $\mathbf{x}_{n+1}$ in nonlinear function $\mathbf{f}$ is derived as

$$\mathbf{x}_{n+1}^{k+1} = \mathbf{x}_n^k + \Delta\mathbf{x}_{n+1}^{k+1},$$

where k is the k-th iteration in Newton's method, $\mathbf{x}_{n+1}^0$ is set as $\mathbf{x}_n$, and

$$\Delta\mathbf{x}_{n+1}^{k+1} = -\frac{\mathbf{f}\left(\mathbf{x}_{n+1}^k\right)}{\mathbf{f}'\left(\mathbf{x}_{n+1}^k\right)}. \tag{III.9}$$

Newton's method will update $\mathbf{x}_{n+1}$ until Eqn. (III.9) is small than a given tolerance. The derivative of $\mathbf{f}(\mathbf{x}_{n+1})$ with respective of (x) is

$$\mathbf{f}'(x_{n+1}) \quad = \quad \mathbf{C}_{n+1} - \frac{h}{2}\frac{\delta \mathbf{i}}{\delta \mathbf{x}}\bigg|_{\mathbf{x}_{n+1}} = \mathbf{C}_{n+1} - \frac{h}{2}\mathbf{G}^{nl}_{n+1},$$

where Jacobian matrix $\frac{\partial \mathbf{i}}{\partial \mathbf{x}}$ is actually equivalent to the effective conductance of nonlinear components at t_n, which is denoted as $\mathbf{G}^{nl}$. The update delta of every iteration for $\mathbf{x}_{n+1}$ by Newton's method is expressed as

$$\Delta\mathbf{x}^{k+1}_{n+1} = -\frac{\mathbf{f}(\mathbf{x}^{k}_{n+1})}{\mathbf{f}'(\mathbf{x}^{k}_{n+1})} \quad = \quad -\left(\mathbf{C}_{n+1} - \frac{h}{2}\mathbf{G}^{nl}_{n+1}\right)^{-1}\left(\mathbf{C}_{n}\mathbf{x}_{n+1} - \frac{h}{2}\mathbf{i}(\mathbf{x}_{n+1}) - \mathbf{C}_{n}\mathbf{L}\right)$$

It should be noticed that instead of deriving the Jacobian matrix directly, as most SPICE-like simulators, we can construct $\mathbf{G}^{nl}$ via the inspection of linearized nonlinear components. In comparison with the implicit methods, e.g., backward Euler, the Jacobian matrix $(\mathbf{C}_n/h + \mathbf{G}^{l} + \mathbf{G}^{nl})$ will burden the linear system solver with much more non-zeros from $\mathbf{G}^{l}$.

The convergence rate of Newton's method is shown in [53]. As long as the following two criteria satisfy, Newton's method can always converge in quadratic rate.

- $\left(\mathbf{C}_{n+1} - \frac{h}{2}\mathbf{G}^{nl}_{n+1}\right)^{-1}$ is non-singular. This criterion is for the existence of $\Delta\mathbf{x}^{k+1}_{n+1}$.
- Initial guess of $\mathbf{x}_{n+1}$ is close to $\mathbf{x}_{n+1}$ enough.

The first criterion is always true since $\mathbf{C}_n$ is non-singular after the regularization process, and for the second criterion, our simulation sets $\mathbf{x}_n$ as the initial guess of $\mathbf{x}_{n+1}$ that will be close enough under an appropriate step size.

The primary version of matrix exponential method (III.2) has two limitations when applied in large-scale circuit simulation. First, the system of equations (II.2) 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 **A**, however, needs not to be nonsingular, since the three matrix exponential functions in their Taylor expansion form are just power series of **A**. Second, direct computation of matrix exponential is prohibitive [45] as the dimension of matrices in modern circuit simulation easily exceeds 1 million. The two problems are addressed in the following two sections.

III.3 Regularization

The most common cause of singular **C** matrix in (II.2) is the empty rows in **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 **C** non-invertible or ill-conditioned even though it has no zero rows [17].

We have reported in a separate work [14] a two-phase regularization technique to construct from the original MNA system with singular **C** matrix an equivalent system with invertible **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.

III.3.A Systematic Elimination

We utilize the regularization flow developed by Natarajan [49], which reduces $\mathbf{C}$ to its the row echelon form

$$\begin{bmatrix}\mathbf{C}_{11} & \mathbf{C}_{12}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\dot{\mathbf{x}}_1\\ \dot{\mathbf{x}}_2\end{bmatrix} = -\begin{bmatrix}\mathbf{G}_{11} & \mathbf{G}_{12}\\ \mathbf{G}_{21} & \mathbf{G}_{22}\end{bmatrix}\begin{bmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{bmatrix} + \begin{bmatrix}\mathbf{B}_1\\ \mathbf{B}_2\end{bmatrix}\mathbf{u}. \tag{III.10}$$

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 Gaussian elimination (BGE) is applied to obtain a reduced system of equations

$$\mathbf{C}_r\dot{\mathbf{x}}_1 = -\mathbf{G}_r\mathbf{x}_1 + \mathbf{B}_{0r}\mathbf{u} + \mathbf{B}_{1r}\dot{\mathbf{u}}, \tag{III.11}$$

where

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

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

$$\mathbf{x}_2 = -\,\mathbf{G}_{22}^{-1}\left(\mathbf{G}_{21}\mathbf{x}_1 - \mathbf{B}_2\mathbf{u}\right). \tag{III.12c}$$

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 (II.2)

$$\mathbf{C}_r\dot{\mathbf{x}}_r = -\mathbf{G}_r\mathbf{x}_r + \mathbf{B}_r\mathbf{u}, \tag{III.13}$$

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

The regularization of (III.10) 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 $\mathbf{C}_r$ 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 [49]. 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 [61]. Hence, we propose to reduce the index-2 circuit to its index-1 equivalent *prior to* the elimination process of (III.10) by detecting and breaking all CV-loops and LI-cutsets in the topology. The index-1 system are then fed into (III.10) which is guaranteed to stop after *one* iteration

III.3.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 [34]. 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 (non-datum) 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 (III.14a) and the corresponding elimination flow follows from (III.14b) to (III.14f) (assume v_i and i_v are eliminated). Dependent voltage sources are eliminated analogously.

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} G_{ii} & G_{ij} & 1 \\ G_{ji} & G_{jj} & -1 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i} \\ I_{s_j} \\ \hline V_s \end{array}\right] \tag{III.14a}$$

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline V_s \end{array}\right] \tag{III.14b}$$

$$\left[\begin{array}{cc|c} C_{ii} & C_{ij} & 0 \\ C_{ji} & C_{jj} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] = -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} 0 \\ 0 \\ \hline -\dot{V}_s \end{array}\right] \tag{III.14c}$$

$$\begin{aligned} \left[\begin{array}{cc|c} 0 & C_{ij}+C_{ii} & 0 \\ 0 & C_{jj}+C_{ji} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] &= -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji} & -1 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] + \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s \\ \hline 0 \end{array}\right] \\ &+ \left[\begin{array}{c} -C_{ii}\dot{V}_s \\ -C_{ji}\dot{V}_s \\ \hline -\dot{V}_s \end{array}\right] \end{aligned} \tag{III.14d}$$

$$\begin{aligned} \left[\begin{array}{cc|c} 0 & C_{ij}+C_{ii} & 0 \\ 0 & C_{jj}+C_{ji}+C_{ij}+C_{ii} & 0 \\ \hline -1 & 1 & 0 \end{array}\right]\left[\begin{array}{c} \dot{v}_i \\ \dot{v}_j \\ \hline \dot{i}_v \end{array}\right] &= -\left[\begin{array}{cc|c} 0 & G_{ij}+G_{ii} & 1 \\ 0 & G_{jj}+G_{ji}+G_{ij}+G_{ii} & 0 \\ \hline 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} v_i \\ v_j \\ \hline i_v \end{array}\right] \\ &+ \left[\begin{array}{c} I_{s_i}+G_{ii}V_s \\ I_{s_j}+G_{ji}V_s+I_{s_i}+G_{ii}V_s \\ \hline 0 \end{array}\right] + \left[\begin{array}{c} -C_{ii}\dot{V}_s \\ -C_{ji}\dot{V}_s-C_{ii}\dot{V}_s \\ \hline -\dot{V}_s \end{array}\right] \end{aligned} \tag{III.14e}$$

$$\begin{aligned} \left[C_{jj}+C_{ji}+C_{ij}+C_{ii}\right]\left[\dot{v}_j\right] &= -\left[G_{jj}+G_{ji}+G_{ij}+G_{ii}\right]\left[v_j\right] \\ &+\left[I_{s_j}+G_{ji}V_s+I_{s_i}+G_{ii}V_s\right]+\left[-C_{ji}\dot{V}_s-C_{ii}\dot{V}_s\right] \end{aligned} \tag{III.14f}$$

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 [5]. Once the inductor is chosen for a given LI-cutset (denoted as L_1), a similar process to (III.14b) 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 KCL of the LI-cutset (III.15) and its differential version are applied to eliminate i_{L_1} and $\dot{i}_{L_1}$ in $\mathbf{G}$ and $\mathbf{C}$, respectively.

$$i_{L_1} + \sum_{j=2}^{N_L} i_{L_j} + \sum_{j=1}^{N_{I_s}} I_{s_j} = 0. \tag{III.15}$$

III.3.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 capacitors”, a group of connected capacitors isolated by noncapacitive 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 [14]. 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. III.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 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 which affects only a small portion of the entire matrices.

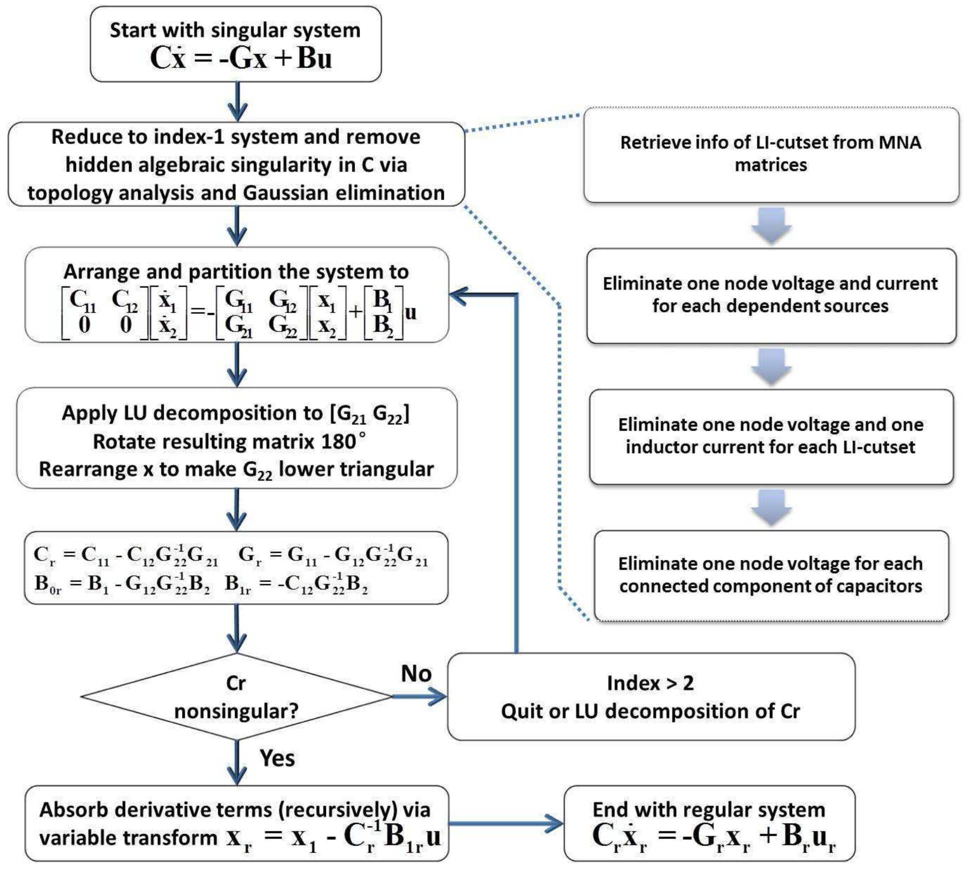

Start with singular system $C\dot{x} = -Gx + Bu$
Reduce to index-1 system and remove hidden algebraic singularity in C via topology analysis and Gaussian elimination
Arrange and partition the system to $\begin{bmatrix} C_{11} & C_{12} \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = -\begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} B_1 \\ B_2 \end{bmatrix}u$
Apply LU decomposition to $[G_{21}\ G_{22}]$
Rotate resulting matrix 180°
Rearrange x to make G_{22} lower triangular
$C_r = C_{11} - C_{12}G_{22}^{-1}G_{21}$ $G_r = G_{11} - G_{12}G_{22}^{-1}G_{21}$
$B_{0r} = B_1 - G_{12}G_{22}^{-1}B_2$ $B_{1r} = -C_{12}G_{22}^{-1}B_2$
Cr nonsingular?
No
Yes
Index > 2
Quit or LU decomposition of Cr
Absorb derivative terms (recursively) via variable transform $x_r = x_1 - C_r^{-1}B_{1r}u$
End with regular system $C_r\dot{x}_r = -G_rx_r + B_ru_r$
Retrieve info of LI-cutset from MNA matrices
Eliminate one node voltage and current for each dependent sources
Eliminate one node voltage and one inductor current for each LI-cutset
Eliminate one node voltage for each connected component of capacitors

Figure III.1: Flow of regularization process.

III.3.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($N_g log N_g$) [40] where N_g is the size of a reduced graph [5]. 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 includes mainly 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.

III.4 Computation of Matrix Exponential

III.4.A Merge of Three Functions into One Matrix Exponential

The analytical solution (III.2) has three matrix exponential functions, which are generally referred as φ-functions of the zero, first and second order [52]. Al-Mohy and Higham [2, Theorem 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 (III.2). Therefore, we only need to calculate the exponential of a slightly larger matrix to obtain the analytical solution (III.2), which can be rewritten into

$$\mathbf{x}(t+h) = \begin{bmatrix} \mathbf{I}_n & 0 \end{bmatrix} e^{\mathbf{A}'h} \begin{bmatrix} \mathbf{x}(t) \\ e_2 \end{bmatrix}, \tag{III.16}$$

with

$$\begin{aligned} \mathbf{A}' &= \begin{bmatrix} \mathbf{A} & \mathbf{W} \\ \mathbf{0} & \mathbf{J} \end{bmatrix}, \quad \mathbf{W} = \begin{bmatrix} \frac{\mathbf{b}(t+h)-\mathbf{b}(t)}{h} & \mathbf{b}(t) \end{bmatrix} \\ \mathbf{J} &= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \qquad e_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix} \end{aligned} \tag{III.17}$$

Using (III.16) only one matrix exponential evaluation is needed in each time step, and the problem boils down to how to compute (III.16) efficiently.

III.4.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 which 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 [38] [56].

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 $\mathbf{V_m} = [\mathbf{v}_1, \dots, \mathbf{v}_m]$
$\mathbf{v_1} = \mathbf{v}/\|\mathbf{v}\|_2$;
for $j = 1, 2, \dots, m$ do
    $\mathbf{w} = \mathbf{A}\mathbf{v}_j$;
    for $i = 1, 2, \dots, j$ 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
```

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 below:

$$e^{\mathbf{A}}\mathbf{v} \equiv (\mathbf{I} + \mathbf{A} + \frac{\mathbf{A}^2}{2!} + \dots + \frac{\mathbf{A}^k}{k!} + \dots)\mathbf{v}. \tag{III.18}$$

$$\begin{aligned}
e^{\mathbf{A}'h}\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix} &= \exp\left(\begin{bmatrix}-\mathbf{C}^{-1}\mathbf{G} & \mathbf{C}^{-1}\mathbf{W}_u\\ 0 & \mathbf{J}\end{bmatrix}h\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}\\
&= \exp\left(\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}\begin{bmatrix}-\mathbf{G} & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}\end{bmatrix}\frac{h}{\alpha}\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}\\
&= \exp\left(\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}\begin{bmatrix}-\mathbf{G}-(\mathbf{C}/\alpha) & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}-\mathbf{I}_2\end{bmatrix}\frac{h}{\alpha}+\begin{bmatrix}\mathbf{I}_n & 0\\ 0 & \mathbf{I}_2\end{bmatrix}\frac{h}{\alpha}\right)\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}\\
&= \exp\left(\frac{1}{\alpha}\underbrace{\begin{bmatrix}\mathbf{C}/\alpha & 0\\ 0 & \mathbf{I}_2\end{bmatrix}^{-1}}_{\tilde{\mathbf{C}}}\underbrace{\begin{bmatrix}-\mathbf{G}-(\mathbf{C}/\alpha) & \mathbf{W}_u\\ 0 & \alpha\mathbf{J}-\mathbf{I}_2\end{bmatrix}}_{\tilde{\mathbf{G}}}h\right)\underbrace{\begin{bmatrix}\mathbf{x}(t)\\ e_2\end{bmatrix}e^{\frac{h}{\alpha}}}_{\mathbf{v}} = e^{\tilde{\mathbf{A}}h}\mathbf{v}
\end{aligned} \tag{III.19}$$

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

$$\mathrm{K_m}\left(\mathbf{A},\mathbf{v}\right) = span\{\mathbf{v},\mathbf{A}\mathbf{v},\ldots,\mathbf{A}^{m-1}\mathbf{v}\}$$

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 $\mathrm{K_m}$. Note that in each Arnoldi iteration we compute $\mathbf{A}\mathbf{v}$ as $-\mathbf{C}^{-1}\left(\mathbf{G}\mathbf{v}\right)$. 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

$$\mathbf{H_m} = \mathbf{V_m^{\mathsf{T}}}\mathbf{A}\mathbf{V_m}. \tag{III.20}$$

Then we project $\mathbf{A}$ onto the Krylov subspace, and with (III.20) derive the approximation of $e^{\mathbf{A}}\mathbf{v}$ as [56]

$$\begin{aligned} e^{\mathbf{A}}\mathbf{v} &\approx \mathbf{V_m}\mathbf{V_m^T}e^{\mathbf{A}}\mathbf{v} = \|\mathbf{v}\|_2\mathbf{V_m}\mathbf{V_m^T}e^{\mathbf{A}}\mathbf{V_m}e_1 \\ &= \|\mathbf{v}\|_2\mathbf{V_m}e^{\mathbf{H_m}}e_1. \end{aligned} \tag{III.21}$$

To use the Krylov subspace method to compute (III.16), we first rewrite it into (III.19) where

$$\tilde{\mathbf{A}} = \frac{1}{\alpha}\tilde{\mathbf{C}}^{-1}\tilde{\mathbf{G}}, \qquad \mathbf{W}_u = \mathbf{B}\left[\begin{array}{cc} \frac{\mathbf{u}(t+h)-\mathbf{u}(t)}{h} & \mathbf{u}\left(t\right) \end{array}\right]. \tag{III.22}$$

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 (III.21), the overall solution of a new time step is

$$\mathbf{x}\left(t+h\right) = \left[\begin{array}{cc} \mathbf{I}_n & 0 \end{array}\right] \|\mathbf{v}\|_2\, \mathbf{V_m}e^{\mathbf{H_m}h}e_1. \tag{III.23}$$

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 [36, 45], 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 [57], e.g. CG or GMRES, to speed up the solution of linear system arising from, e.g. implicit numerical integration methods. It has been proved in [38] that convergence of Krylov subspace method for matrix exponential operator is faster than that for the iterative solution of linear systems.

III.4.C Stability

The stability region of matrix exponential formulation (III.19) 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 (III.23) is also A-stable when $\tilde{\mathbf{A}}$ is normal by the following theorem

THEOREM III.4.1 *For passive circuits, MEXP computed by the Krylov subspace approximation is A-stable when $\tilde{\mathbf{A}}$ is normal.*

Proof 1 *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, equation (III.16) 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

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

Ref. [56] proves 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 [7] 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 III.4.D). In our experiment, MEXP by Krylov subspace approximation with m ranged from 10 to 100 is stable for all test cases.

III.4.D Error Analysis

A priori error bound of computing the matrix exponential (III.21) by the Krylov subspace projection is given by

$$err \leq 2\|\mathbf{v}\|_2 \frac{\rho^{m+1}e^{\rho}}{(m+1)!}, \tag{III.24}$$

where $\rho = \|\tilde{\mathbf{A}}h\|_2$ [38,56]. 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 matrix exponential method 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 [56] is commonly adopted to determine the error of (III.23), which reads

$$err = \|\mathbf{v}\|_2 \mathbf{H}(m+1,m)|e_m^{\mathsf{T}} \varphi_1(\mathbf{H_m}) e_1|, \tag{III.25}$$

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

III.5 Adaptivity

One pleasing feature of MEXP lies in the ease of adaptively adjusting h during the numerical integration. According to (III.20), 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 re-evaluate (III.23) 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 re-evaluation process of adjusting time step involves scaling of $\mathbf{H}_m$ and re-computing 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 re-evaluation process is insignificant. In contrast, the implicit methods have to re-solve 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 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 (III.25), we require the error at $n+1$th step ϵ_{n+1} to meet the following inequality

$$\frac{\epsilon_{n+1}}{\epsilon_n} \leqslant \gamma \frac{\epsilon_{n+1}^{\max}}{\epsilon_n} = \gamma \frac{h_{n+1} Tol}{t_f \epsilon_n} = \gamma \frac{h_n Tol}{t_f \epsilon_n} \frac{h_{n+1}}{h_n} = \frac{\gamma}{w} \frac{h_{n+1}}{h_n} \tag{III.26}$$

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

$$w = \frac{\epsilon_n}{\epsilon_n^{\max}} = \frac{t_f \epsilon_n}{h_n Tol}. \tag{III.27}$$

III.5.A Prediction of h and m

Unlike the separate changes of h and m in [52], 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 (III.24), as

$$\frac{\epsilon_{n+1}}{\epsilon_n} = \left(\frac{h_{n+1}}{h_n} \right)^{\frac{m_{n+1}}{\beta}+1} \kappa^{-(m_{n+1}-m_n)}, \tag{III.28}$$

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 (III.26), i.e.,

$$\min \ \frac{t_f - t_n}{h} Q(m), \tag{III.29a}$$

$$s.\,t. \ \left(\frac{h}{h_n} \right)^{\frac{m}{\beta}} \kappa^{-(m-m_n)} \leqslant \frac{\gamma}{w}. \tag{III.29b}$$

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 w.r.t. m of (III.29b) is negative, which gives

$$h/h_n \leqslant \kappa^{\beta}. \tag{III.30}$$

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

$$Q(m) = c_1 m^2 + c_2 m + c_3, \tag{III.31}$$

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 (III.29a) achieves minimum when the constraint (III.29b) takes equality and postpone the proof later in this subsection. With this assumption, we solve h from (III.29b) as

$$\begin{aligned} \ln h &= \beta\left(\log\left(\frac{\gamma}{w}\right) - m_n \log\kappa\right)\frac{1}{m} + \left(\beta\log\kappa + \log h_n\right) \\ &= c_4 m^{-1} + c_5 = P(m). \end{aligned} \tag{III.32}$$

Substituting (III.32) into (III.29a), the objective function becomes a function of m only, namely, $(t_f - t_n)\, Q(m) e^{-P(m)}$, and the extreme value is obtained when the function derivative is zero, yielding

$$2c_1 m^3 + \left(c_2 + c_1 c_4\right) m^2 + c_2 c_4 m + c_3 c_4 = 0, \tag{III.33}$$

whose positive roots are the solution of m. The corresponding h is then obtained by (III.32). The new h is restricted by the negative derivative constraint (III.30) 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 (III.29)

Lemma III.5.1 *Provided (III.30) holds, the polynomial in (III.33) has one and only one positive root.*

Proof 2 *It is trivial to show $c_1 > 0$, $c_2 > 0$ and $c_3 > 0$. If (III.30) holds, we have $c_4 < 0$ from (III.32). The number of sign changes between the coefficients of the polynomial in (III.33) is one regardless of the sign of the second coefficient. Determined by Descartes' rule of signs, the polynomial has exactly one positive root.*

THEOREM III.5.1 *Given (III.30), the m_{opt} and h_{opt} computed by (III.33) and (III.32) are the optimal solution of (III.29).*

Proof 3 *We prove it by contradiction. Denote (III.29a) and (III.29b) by $F(h, m)$ and $C(h, m) \leqslant \frac{\gamma}{w}$. We assume there exists another pair of (h', m') ($h' \neq h_{opt}, m' \neq m_{opt}$) being a solution no worse than h_{opt}, m_{opt} for (III.29), i.e.,*

$$F(h', m') \leqslant F(h_{opt}, m_{opt}), C(h', m') \leqslant C(h_{opt}, m_{opt}). \tag{III.34}$$

If $C(h', m') < C(h_{opt}, m_{opt})$, 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 $F(\tilde{h}', m') < F(h', m') \leqslant F(h_{opt}, m_{opt})$, which is contradictory to the fact that $F(h_{opt}, m_{opt})$ is at its minimum for the equality constraint. If $C(h', m') = C(h_{opt}, m_{opt})$ (and $F(h', m') = F(h_{opt}, m_{opt})$), it is equivalent that m' is another positive solution of (III.33), which is in contradiction to the Lemma 5.1.

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 (LS) fitting of (III.28) with the above five data points.

With only moderate accuracy requirement, we solve the LS fitting problem by taking logarithm on both sides of (III.28)

$$\left(\frac{m}{\beta} + 1\right) \log \frac{h}{h_n} - (m - m_m) \log \kappa = \log \frac{\epsilon}{\epsilon_n}. \tag{III.35}$$

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

$$\begin{bmatrix} y_1^{(1)} y_2^{(1)} & m_n - y_2^{(1)} \\ y_1^{(2)} y_2^{(2)} & m_n - y_2^{(2)} \\ \vdots & \vdots \\ y_1^{(5)} y_2^{(5)} & m_n - y_2^{(5)} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} z^{(1)} - y_1^{(1)} \\ z^{(2)} - y_1^{(2)} \\ \vdots \\ z^{(5)} - y_1^{(5)} \end{bmatrix}, \tag{III.36}$$

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

III.5.B Correction of h Based on 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 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 posteriori error falls into an interval of $[w_{min}, w_{max}]$. When enlarging h, the two constraints defined in Section III.5.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 re-evaluation of a small matrix exponential, once can apply the largest permissible step size right in the same step.

3. Varying order of approximation, e.g., automatic switch between 1st, 2nd 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.

III.6 Experimental Results

We prototype MEXP in MATLAB and integrate with a SPICE-like circuit simulator SMORES developed in MIT [12]. The BSIM3 [16] compact model is also

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 posteriori error $\epsilon$ by (III.25) and new $w$;
    end
    while $w > w_{max}$ do
        scale down $h$ and $\mathbf{H}_m$;
        compute posteriori error $\epsilon$ and new $w$;
    end
    calculate $\mathbf{x}_{new}$ by (III.23);
    [$\beta$, $\kappa$] = findParameter($h$,$m$,$\epsilon$,$\mathbf{H}$,$\mathbf{V}$);
    [$h_{new}$, $m_{new}$] = prediction($t_f$,$t$,$h$,$m$,$w$,$\beta$,$\kappa$);
    $t = t + h$;
    $\mathbf{x}(t) = \mathbf{x}_{new}$;
    $h = h_{new}$;
    $m = m_{new}$;
end
```

integrated into MATLAB environment via MEX interface for nonlinear devices. Experiments are performed on a server with Intel Xeon 3.0GHz CPU and 16GB memory, with testbench circuits of different sizes and characteristics.

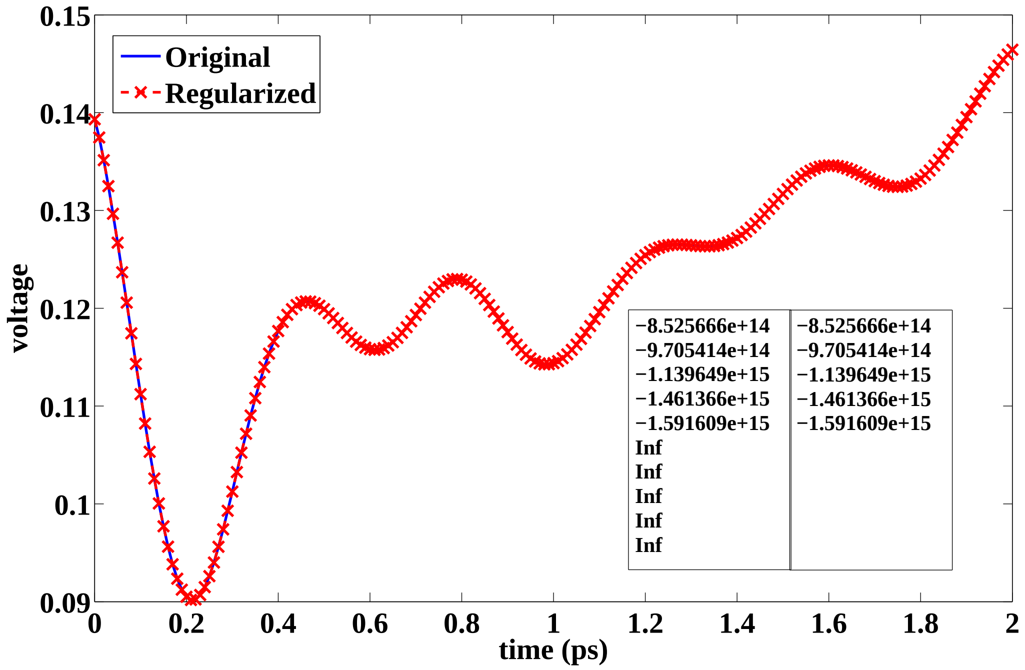
Table III.1 provides detailed specifications. Type indicates linear (L) or nonlinear (NL) circuits. Index gives the DAE index of the MNA systems. The numbers of nonzeros in $\mathbf{C} + \mathbf{G}$ before and after regularization are also shown. For fairness, a MATLAB implementation of TRAP is used to provide benchmarks for accuracy and performance comparisons. All linear systems are solved by the direct solver (backslash) in MATLAB.

Table III.1: 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	16bit adder	NL	579	0	1	3.6K	3.6K
D9	ALU	NL	10K	373	1	44.3K	43.4K

III.6.A Results for Regularization

Fig. III.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. III.2 also demonstrates an exact equivalence between the original and the regularized systems.



−8.525666e+14	−8.525666e+14
−9.705414e+14	−9.705414e+14
−1.139649e+15	−1.139649e+15
−1.461366e+15	−1.461366e+15
−1.591609e+15	−1.591609e+15
Inf	
Inf	
Inf	
Inf	
Inf	

Figure III.2: Accuracy of regularization process.

III.6.B Performance of Krylov Subspace Method in Computing Matrix Exponential

In this subsection, we show the numerical advantage of Krylov subspace method over traditional Taylor's expansion (III.18). While Taylor's expansion can

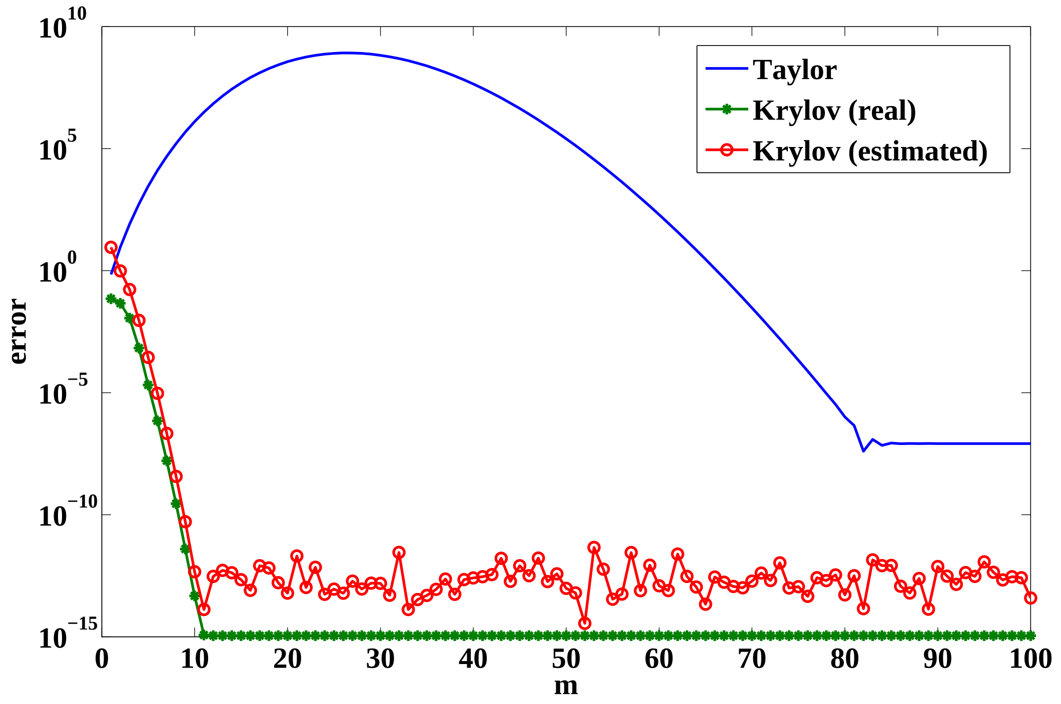


Figure III.3: Errors of computing $e^{\mathbf{A}h}v$ by Taylor's expansion and Krylov subspace method w.r.t. m. Reference solution is obtained by $expm(\mathbf{A}h)v$ $(h = 0.1ps)$. Both real error and posterior error estimate of Krylov method are shown.

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. III.3 shows the advantage of using the Krylov subspace method (III.21) to calculate $e^{\mathbf{A}h}v$, compared with Taylor's expansion (III.18). The reference result is computed via the MATLAB built-in function *expm* [36], 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 (III.18), i.e., how fast the factorial in denominator can dominate the nominator. In Fig. III.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 (III.20), which is major source of the approximation error, the Krylov subspace method is more accurate than Taylor's expansion under the same dimensions. Fig. III.3 shows that the error of Krylov subspace method saturates to 10^{-15} at $m = 11$. The error estimated by the posteriori formula (III.25) 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. III.3, which is limited by the double precision.

III.6.C Performance of Uniform Matrix Exponential Method

Fig. III.4 shows the transient response of D3 simulated by MEXP, TRAP (TR) and forward Euler method (FE). We apply fixed step sizes $1ps$ and $5ps$ for both MEXP and TRAP method, and $1ps$ 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 $5ps$, MEXP can till have its waveform "jump" to the correct points (the yellow crosses) at the waveform of $1ps$. The point-wise 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 $5ps$. 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 $1ps$.

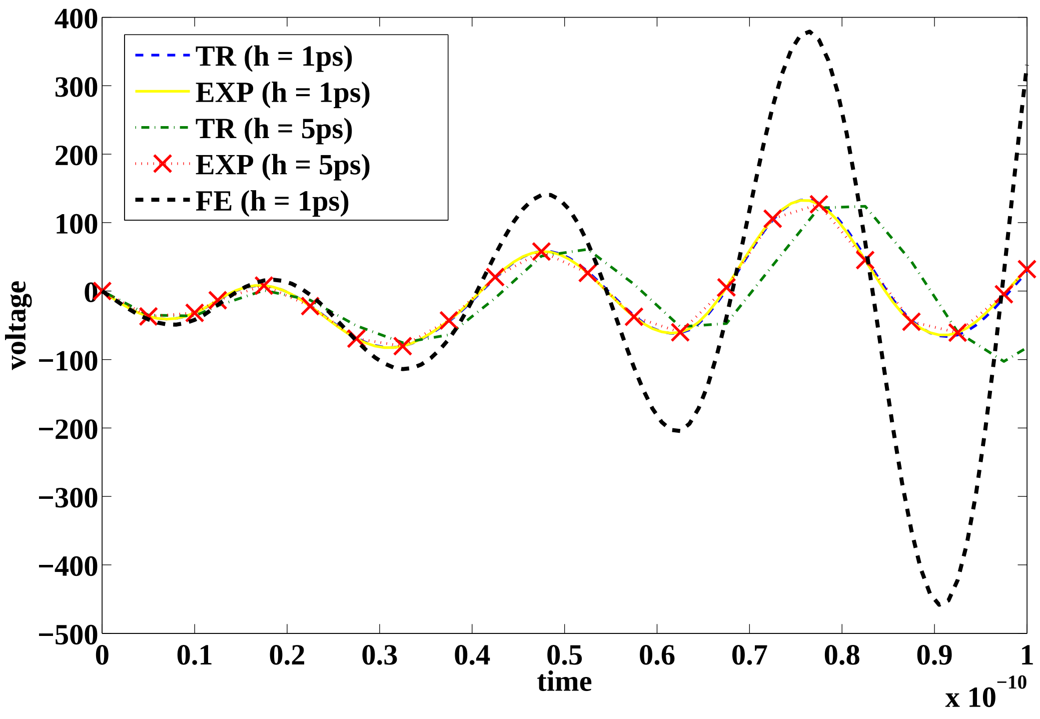


Figure III.4: Accuracy comparison among matrix exponential, trapezoidal and forward Euler methods for different step size (linear cases).

Table III.2 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 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 $h = 10h_{ref}$ and $h = 100h_{ref}$. 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 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 ($\sim 4X$ 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.

Table III.2: 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	0.01ps	100ps	1,957.3s	0.1ps	34.5s	3.0×10^{-4}	0.1ps	180.3s	8.6×10^{-3}	30
				1ps	1.9s	4.4×10^{-3}	1ps	20.6s	4.0×10^{-1}	
D2	0.01ps	10ps	2,728.3s	0.1ps	282.2s	7.1×10^{-2}	0.1ps	589.4s	3.7×10^{-3}	30
				1ps	43.5s	2.1×10^{-1}	1ps	90.8s	3.8×10^{-2}	
D3	0.1ps	100ps	27,064.5s	1ps	2,907.1s	2.8×10^{-3}	1ps	1,190.2s	2.8×10^{-5}	20
				10ps	426.6s	2.1×10^{-1}	10ps	176.8s	3.2×10^{-5}	
D4	0.01ps	10ps	1.8×10^{5}s (est.)	0.1ps	14,760.2s	N/A	0.1ps	3,796.2s	N/A	40
				1ps	2,102.5s	N/A	1ps	565.8s	N/A	
D5	0.1ps	100ps	N/A	0.1ps	N/A	N/A	0.1ps	6,491.1s	N/A	20
				1ps	N/A	N/A	1ps	1,168.7s	N/A	

III.6.D Performance of Adaptive Matrix Exponential Method

This subsection demonstrates the advantage of using the adaptive scheme in Section III.5 in MEXP. We first verify that the formula (III.28) provides a usable error prediction for matrix exponential computation. Fig. III.5 compares the predicted error and the real posteriori error by (III.25) 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 (III.31). 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. III.6 indicates a good match between the predicted and real optimal (h, m) pairs, and thus the effectiveness of our prediction scheme.

Table III.3 compares the performance of TRAP and MEXP with adaptive control for the four linear cases. The error is measured by w in (III.27) 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 current step (if $w > 1.2$) or for the next step (if $w < 0.6$), i.e., $h_{new} = \sqrt{w} h_{old}$. We also implement two versions of adaptive MEXP: one adjusts h only using the correction scheme; the other adjusts both h and m by the prediction-correction scheme described in Section III.5. 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.

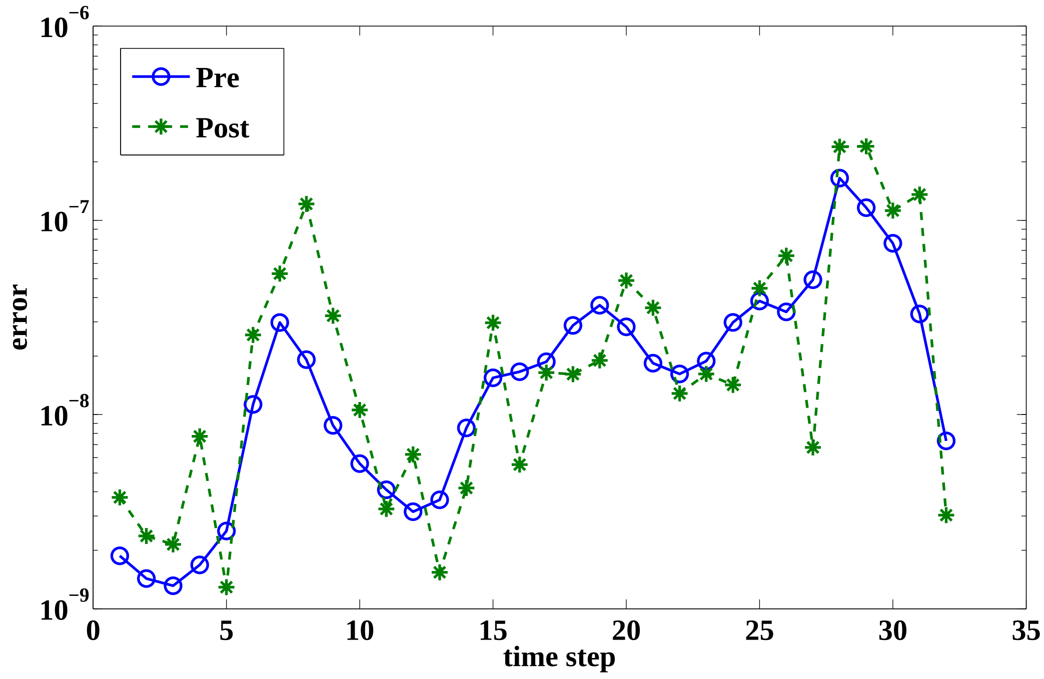


Figure III.5: Predicted and posteriori errors comparison.

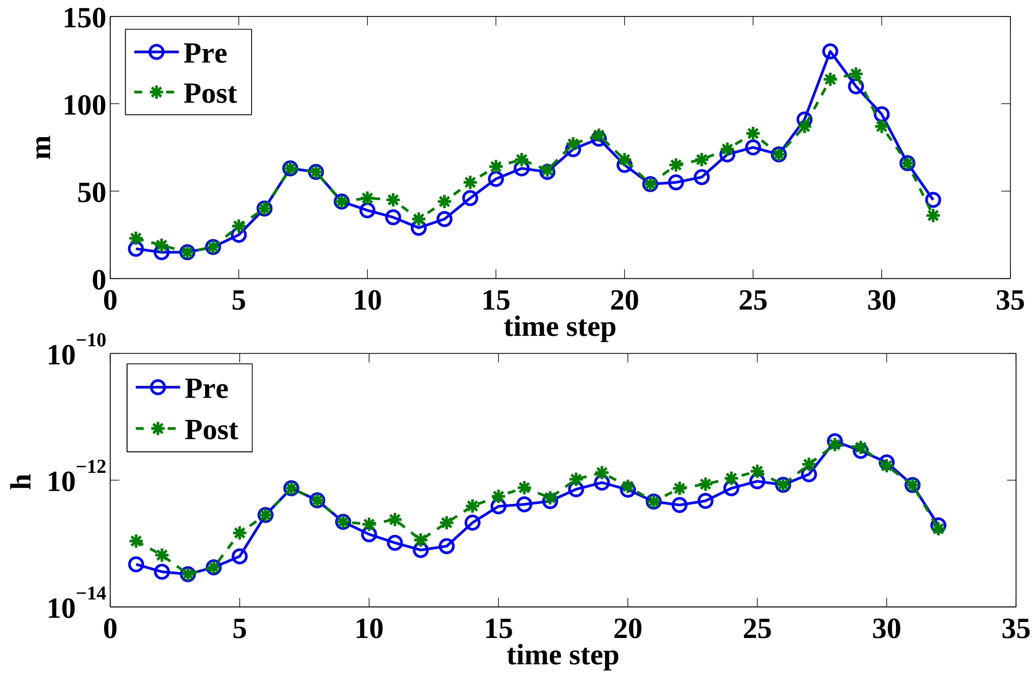


Figure III.6: Predicted and real optimal (h, m) pair.

Table III.3: 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	100ps	0.01ps	30	416	11.3s	117	759	201.5s	120	72.6s
D2	10ps	0.01ps	30	99	394.1s	46	41	214.7s	40	161.4s
D3	100ps	0.1ps	20	130	3,118.2s	5	70	200.2s	55	112.3s
D4	10ps	0.01ps	40	187	33,802.1s	29	181	3,004.1s	174	2,135.2s
D5	100ps	0.1ps	20	N/A	N/A	N/A	75	4,927.2s	45	3,075.9s

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. 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. III.7 shows the difference in point selection of $EXP(h)$ and $EXP(h,m)$ for D1 with a two-square wave input.

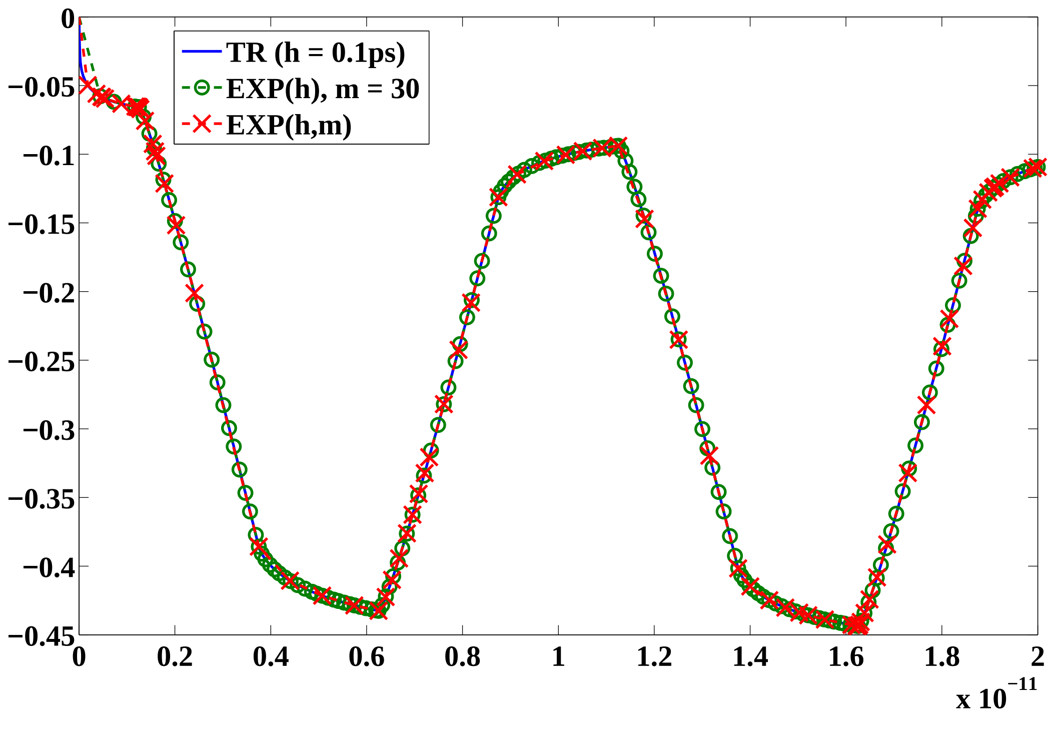


Figure III.7: Selection of time points for the two adaptive matrix exponential methods (D1,$Tol = 10^{-3}$).

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 III.4 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 in MEXP, both with the same convergence criteria. In TRAP, we follow the HSPICE convention [66] 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 (III.7) and the convergence of Newton's method. Posterior correction is used to reduce h when the accuracy of Krylov subspace approximation is not sufficient. The Newton's method is repeated until the solution converges, 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, whichever is smaller.

Table III.4: 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	1ns	1ps	259	1062	357	35.6s	222	1438	166	164.1s	20
D7	100ps	0.1ps	242	670	187	68.4s	533	1070	0	292.6s	30
D8	100ps	0.1ps	371	1996	923	671.4s	114	708	108	408.7s	20
D9	100ps	0.1ps	451	1512	501	8244.5s	285	1299	101	2252.3s	30

Due to the approximation in Eqn. (III.6), MEXP generally requires more nonlinear iterations than TRAP, which can be seen by counting the average per-step effective iteration $(N_{it} - N_{ws})/N_t$. 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 III.4, 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 2 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. III.8.

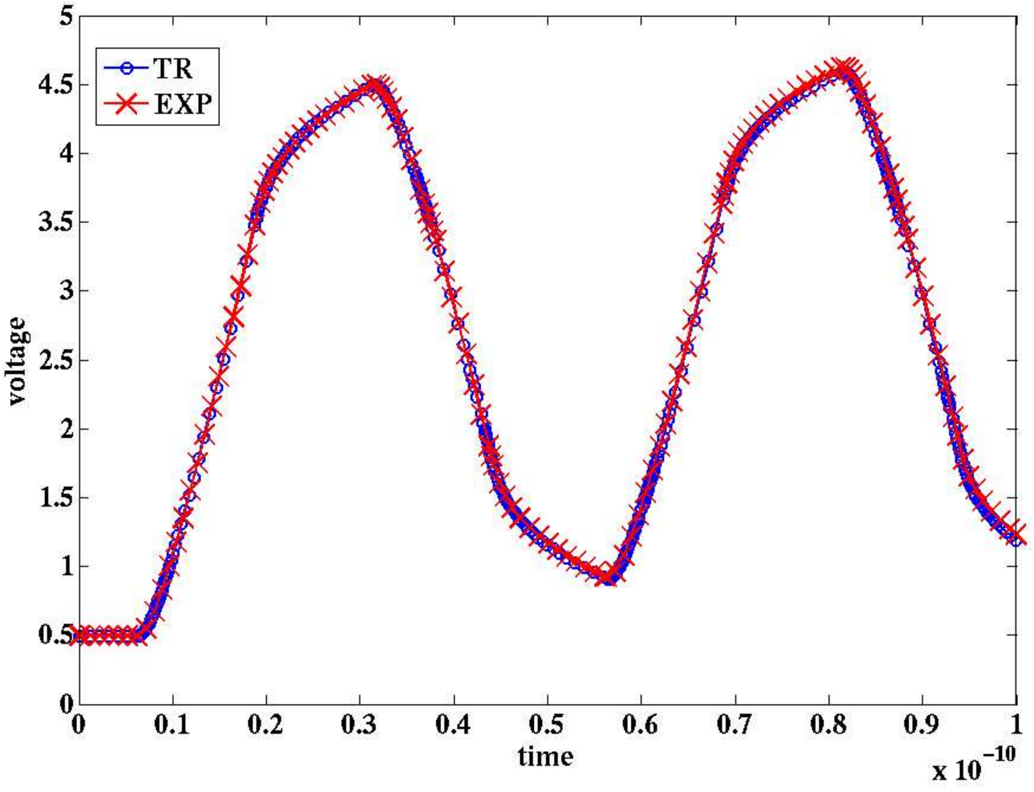


Figure III.8: Accuracy and time point selection of matrix exponential method for nonlinear circuit (D8).

Table III.5: A summary of the features of explicit, implicit and matrix exponential methods.

Methods	Nature	Stability for passives	Matrix to inverse	Main cost	Memory[1]	Adaptivity[2]	Cost of adaption[3]	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

[1]: An 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 $\mathbf{C}$.

[2]: Variable order BDF is not considered here. [3]: the cost of re-evaluation for a new step size.

Fig. III.9 shows the runtime breakdown of $EXP(h, m)$ in Table III.3 and EXP in Table III.4. 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, posteriori error estimation and the calculation of $\mathbf{x}_{new}$. Since the runtime of "findParameter" and "prediction" are insignificant, the figure does not include both the 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 $\mathbf{x}_{new}$, which is larger in the nonlinear cases, such as D6, D7 and D8.

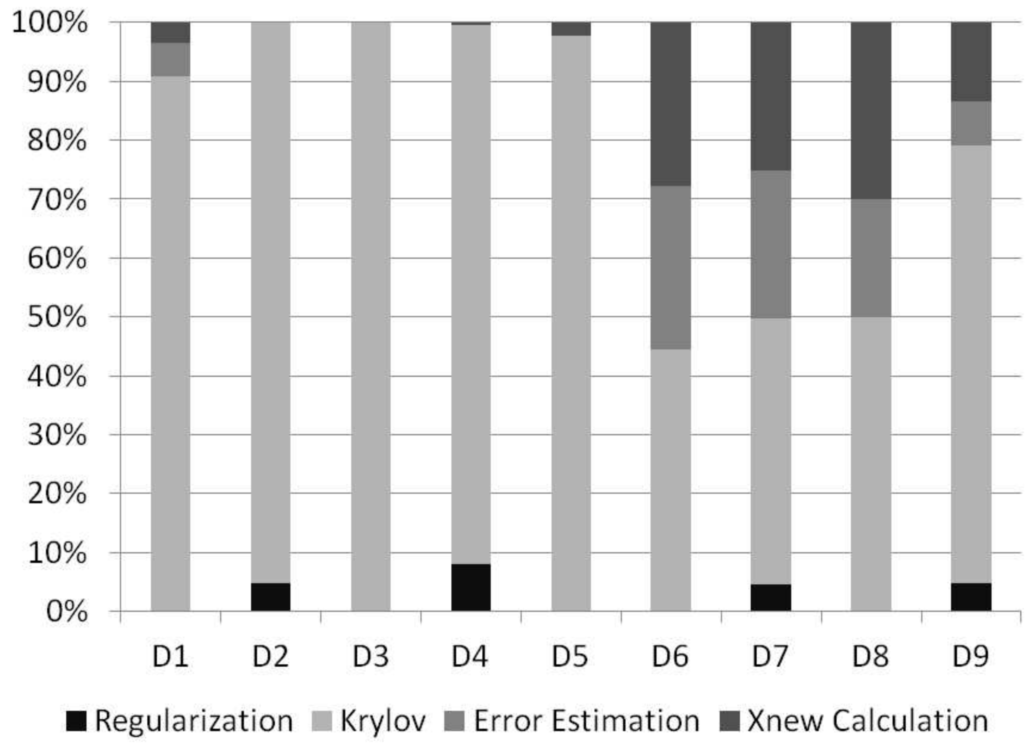


Figure III.9: Runtime breakdown for main steps in Algorithm 2.

As a final remark, Table III.5 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, though 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.

III.7 Summary

An explicit numerical integration method have been presented for accurate and efficient time-domain circuit simulation. Different from conventional linear multi-step method, MEXP solves the linear differential equation analytically via the matrix exponential operator. The computation of matrix exponential is 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.

Chapter III includes the content of one published journal paper, “Time-Domain Analysis of Large-Scale Circuits by Matrix Exponential Method with Adaptive Control,” by Shih-Hung Weng, Quan Chen and Chung-Kuan Cheng, in *IEEE Transaction on Computer-Aided Design of Integrated Circuit and Systems*, pp. 1180–1193, 2012. The dissertation author was the primary investigator and author of the paper.

IV

Stiffness Handling and Parallel Processing in Matrix Exponential Method

In this chapter, we propose an advanced matrix exponential method to handle the transient simulation of stiff circuits and enable parallel simulation. We analyze the rapid decaying of fast transition elements in Krylov subspace approximation of matrix exponential and leverage such *scaling effect* to leap larger steps in the later stage of time marching. Moreover, matrix-vector multiplication and restarting scheme in our method provide better scalability and parallelizability than implicit methods. Specifically, the advantages of the proposed MEXP are

- MEXP reveals the scaling effect of the Krylov subspace method to enable the use of a larger step size when stepping forward.
- MEXP utilizes the restarting scheme to mitigate the memory usage when a large m is needed to strengthen the scaling effect.
- MEXP has the parallelizability and scalability on the GPU platform and for large-scale cases.

The experimental results show that the performance of MEXP for highly stiff circuits is improved up to 4.8 times by exploiting the scaling effect, and is accelerated in the GPU environment up to another 11 times. Furthermore, we demonstrate that MEXP is able to handle the circuit with up to 12 million nodes.

IV.1 Background

Beyond the traditional explicit and implicit methods, a new class of explicit methods called exponential time differencing (ETD) has been drawing attention in the numerical community. The ETD methods analytically solve an ODE system within every discretized time step by directly computing the exponential of a matrix. In theory, the ETD methods avoid the local truncation error (LTE) of the polynomial expansion approximation, and the stability is the same as the trapezoidal method for passive systems. In application, the exponential of a matrix required in the ETD methods can be efficiently approximated by the Krylov subspace method, which involves mainly matrix-vector multiplication and has the advantages of scalability and parallelizability.

In Chapter III, we have embraced the idea of ETD for the circuit simulation and proposed an adaptive step control scheme to enhance the performance. Although the ETD methods demonstrate their advantages, there are still two major limitations for ETD methods. Firstly, stiff circuits enforce the ETD methods to use small step sizes for reducing the approximation error of the Krylov subspace method. Recall the error bound of the matrix exponential approximated by the Krylov subspace in Eqn. (III.24), which is written again as below.

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

where $\rho = \|\tilde{\mathbf{A}}h\|_2$. As we can see, the error dependes on $2-$norm of $\tilde{\mathbf{A}}h$. In a stiff circuit, $2-$ norm of $\tilde{\mathbf{A}}$ could be large, i.e., $\sim 10^{16}$. Hence, we require much smaller h $(\leq 0.1ps)$ to restrain the error, and thus the performance is damaged.

The Krylov subspace method needs a large m or a small h to provide sufficient resolution to the spectrum of $\mathbf{A}h$. As the consequence, a large m will cause the second limitation. That is larger Krylov subspace bases required by stiff circuits, which usually need > 100 bases, pose a memory bottleneck for simulation involving millions of unknowns.

In this chapter, we propose a *matrix exponential method* (MEXP) utilizing the *scaling effect* and restarting scheme to address these two limitations in stiff circuits in the following sections.

IV.2 Scaling Effect and Restarted Matrix Exponential Method for Stiff Circuits

The scaling effect manifests the rapid decaying nature of fast transition components. Ideally, MEXP with the scaling effect can eventually step an arbitrarily large size even for stiff circuits. However, the interpolation error from the Krylov subspace method will prevent the optimal scaling effect and thus restrict the maximal step size.

In general, large m increases the number of interpolation points and reduces the interpolation error. We apply the restarting scheme to the Krylov subspace construction process so that the number of interpolation points increases effectively without adding the memory usage.

IV.2.A Scaling Effect in Matrix Exponential Method

With the one-exp formulation in Eqn. (III.16), time stepping in MEXP can be regarded as a series of product of matrix exponential, i.e., the solution at the $n+1$th step is related to the initial condition $\mathbf{x}_0$ by

$$\mathbf{x}_{n+1} = e^{\mathbf{A}h_{n+1}} e^{\mathbf{A}h_n} ... e^{\mathbf{A}h_1} \mathbf{x}_0 \tag{IV.1}$$

With the eigenvalue decomposition $\mathbf{A} = \mathbf{Q}_A \mathbf{\Sigma}_A \mathbf{Q}_A^{-1}$, (IV.1) becomes

$$\mathbf{x}_{n+1} = \mathbf{Q}_A e^{\mathbf{\Sigma}_A h_{n+1}} e^{\mathbf{\Sigma}_A h_n} ... e^{\mathbf{\Sigma}_A h_1} \mathbf{y}_0 = \mathbf{Q}_A e^{\mathbf{\Sigma}_A h_{n+1}} \mathbf{y}_n, \tag{IV.2}$$

where $\mathbf{y}_i = \mathbf{Q}_A^{-1} \mathbf{x}_i, i = 0, 1, ...$ is the components of $\mathbf{x}_i$ on the eigenvectors space of $\mathbf{A}$ (which are referred as the eigencomponents of vector $\mathbf{x}_i$ hereafter).

Since a circuit intrinsically contains fast and slow transition elements, e.g. small and large capacitors, that correspond to different eigenvalues of $\mathbf{A}$, one can group the elements of $\mathbf{y}_n$, with a given threshold, and use $\mathbf{y}_n^f$ and $\mathbf{y}_n^s$ to represent the corresponding eigencomponents for the fast mode (negative eigenvalues with large magnitude) and the slow mode (negative eigenvalues with small magnitude), respectively. The effect of exponential shows the rapid decaying nature of fast transition elements (i.e., fast damping of negative eigenvalue with large magnitude in the exponent), and is reflected in the eigencomponents $\mathbf{y}_n^f$, which attenuate drastically as stepping forward.

In our MEXP, the Krylov subspace method still preserves such attenuation of the fast mode. The kth basis of the Krylov subspace at time T and in nth step can be represented as

$$\begin{aligned} (\mathbf{A} h_n)^k \mathbf{x}_n &= \mathbf{Q}_A (\mathbf{\Sigma}_A h_n)^k \mathbf{Q}_A^{-1} \mathbf{x}_n \\ &= \mathbf{Q}_A (\mathbf{\Sigma}_A h_n)^k \mathbf{y}_n \\ &= \mathbf{Q}_A (\mathbf{\Sigma}_A h_n)^k e^{\mathbf{\Sigma}_A T} \begin{bmatrix} \mathbf{y}_0^f \\ \mathbf{y}_0^s \end{bmatrix}. \end{aligned} \tag{IV.3}$$

The fast mode eigencomponents $\mathbf{y}_0^f$ are attenuated rapidly by $e^{\mathbf{\Sigma}_A T}$. Although the power of $\mathbf{\Sigma}_A h_n$ acts as a counter force that brings those eigencomponents back to stage, the damping rate of exponential surpasses the increase by the power, so that the Krylov subspace method still benefits from the attenuation of $\mathbf{y}_0^f$ (as shown in Section IV.4.A). The rapid attenuation of eigencomponents implies that MEXP can alleviate the effect of stiffness, caused by the fast mode, and enables the use of larger h in the later stage of time marching where the step size should be more dominated by the slow mode of a circuit. We call this phenomenon as "scaling effect".

With the scaling effect, components of fast transition elements can be rapidly attenuated after a few steps. Smaller m, or larger h, can then be used in the Krylov subspace method at later steps. It should be noticed that for the nonlinear case, even though $\mathbf{A}$ is different every time step due to the nonlinear components, our formulation can still exploit the scaling effect. Since only linear terms are associated with the matrix exponential operator, without affecting Newton's iteration, we can calculate $\mathbf{l}_{n+1}$ separately in the form of (IV.1) using the scaling effect.

Theoretically, according to (IV.3), MEXP can use extremely large h in the later stage of simulation. Nevertheless, due to the approximation error of the Krylov subspace method, the allowable scaling of step size is restricted in practice. To provide an in-depth analysis of the error, we re-interpret the Krylov subspace approximation (III.21) from an interpolation perspective [56].

Lemma IV.2.1 *The approximation (III.16) is mathematically equivalent to approximating* $\exp(\mathbf{A})\mathbf{v}$ *by* $p_{m-1}(\mathbf{A})\mathbf{v}$*, where* p_{m-1} *is the (unique) polynomial of degree* $m-1$*, which interpolates the exponential function in the Hermite sense on the set of Ritz values, the eigenvalues of* $\mathbf{H}_m$*, repeated according to their multiplicities.*

The exact local error vector of approximation (III.16) can be written as

$$\mathbf{r} = e^{\mathbf{A}}\mathbf{v} - p_{m-1}(\mathbf{A})\mathbf{v} = \mathbf{Q}_A \left[e^{\mathbf{\Sigma}_A} - p_{m-1}(\mathbf{\Sigma}_A) \right] \mathbf{y} \tag{IV.4}$$

Provided $\mathbf{A}$ is not highly nonnormal, i.e., the norm of $\mathbf{Q}_A$ remains reasonably bounded, the error mainly depends on $e^{\mathbf{\Sigma}_A} - p_{m-1}(\mathbf{\Sigma}_A)$, the mismatch between the exponential function and the interpolation function evaluated at the eigenvalues of $\mathbf{A}$, which we denote by the "interpolation error". The vector $\mathbf{y}$ denotes the eigencomponents of the starting vector $\mathbf{v}$.

IV.2.B Restarted Krylov Subspace Method

We adopt the restarted Krylov subspace method specific for matrix exponential computation [1,26,27]. Such restarting scheme mitigates the memory usage of the Krylov subspace method when a larger m is needed to strengthen the scaling effect. The Arnoldi process in computing matrix exponential is restarted every m iterations with the last basis vector from the previous cycle being the new starting vector.

$$\mathbf{v}_1^{(k)} = \mathbf{v}_{m+1}^{(k-1)} \tag{IV.5}$$

The approximation $\mathbf{f}$ of the matrix exponential (multiplied with a vector) is updated with a correction term in each restarting.

$$\mathbf{f}^{(k)} = \mathbf{f}^{(k-1)} + \beta \mathbf{V}_m^{(k)} \left[e^{\hat{\mathbf{H}}_{km}} e_1 \right]_{(k-1)m+1:km}, \tag{IV.6}$$

where $\hat{\mathbf{H}}_{km}$ collects all the $\mathbf{H}_m$ from the k cycles of restarting

$$\hat{\mathbf{H}}_{km} = \begin{bmatrix} \mathbf{H}_m^{(1)} & & & \\ \mathbf{E}_m^{(2)} & \mathbf{H}_m^{(2)} & & \\ & \ddots & \ddots & \\ & & \mathbf{E}_m^{(k)} & \mathbf{H}_m^{(k)} \end{bmatrix}, \tag{IV.7}$$

where $\mathbf{E}_m^{(k)} = \eta_k e_m^T e_1$. The posterior error of (IV.6) can be estimated by

$$e^{\mathbf{A}}\mathbf{v} - \mathbf{f}^{(k)} = \eta_k \sum_{j=1}^{2} \left[e_{km}^T \phi_j \left(\hat{\mathbf{H}}_{km} \right) e_1 \right] \omega_{j-1} (\mathbf{A}) \mathbf{v}_{m+1}^{(k)}, \tag{IV.8}$$

where ϕ_j is the divided differences of the exponential function and ω_j is the nodal function w.r.t. the minimal and maximal eigenvalues of $\mathbf{H}_m^{(k)}$.

It is shown in [27] that the k cycles of this restarting is equivalent to interpolating the exponential function (in the Hermite sense) at the *union* of the k set of eigenvalues of $\mathbf{H}_m$. This way, a much larger "effective" m is allowed without increasing the memory demand. In principle, given a sufficient number of restarting, the restarted method is able to work with arbitrary step size.

One major shortcoming of the above restarting scheme lies in calculation of the exponential of a matrix whose complexity increases with km. As a consequence, we would like to limit the number of restarting k at each step and adjust the step size adaptively to meet the accuracy requirement. This naturally calls for an integration of the restarted Krylov subspace method and the adaptive step control based on the scaling effect. Such integration utilizes the restarting scheme to reduce the interpolation error so that the scaling effect can be manifested and exploited by adaptively stepping.

IV.2.C Overall Restarted Algorithm

The context of restarted MEXP follows the MEXP described in Section III.2. The step size is adaptively adjusted according to the posterior error estimate. The two main distinctions lie in that: 1) the maximal number of restarting k_{max} is fixed to a small value, e.g., 5, in each step. It is intended to enlarge the effective m to roughly $k_{max}m$, without inducing too much overhead from the evaluation of $e^{\hat{\mathbf{H}}_{km}}$; 2) unlike in the unrestarted case, there is no fast re-evaluation once h is changed, since the basis vectors $\mathbf{V}_m$ from previous restarting cycles, which are not stored to save memory, are all required to generate an updated approximation $\mathbf{f}$. Hence, we only employ re-evaluation(s) when the error exceeds the tolerance. If the step size can be increased owing to a small error, we use the step size in the next step, instead of applying it right in the current step with re-evaluation. In this way, we could still benefit from the scaling effect while minimizing the number of re-evaluations. The posterior error estimate with a changed h is calculated by (IV.8), with the storage of some auxiliary quantities. The whole restarted MEXP is summarized in Algorithm 3.

Algorithm 3: Restarted MEXP

```
Input: $\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$, $\mathbf{u}(t)$, initial $h$, initial $m$, total error budget $TOL$
       and total time $t_f$
Output: $\mathbf{x}(t)$
$t = 0$; $\mathbf{x}(0) =$ DC_analysis;
while $t \leq T$ do
    $[\mathbf{C}_r, \mathbf{G}_r, \mathbf{B}_r] =$ regularization($\mathbf{C}, \mathbf{G}, \mathbf{B}$);
    Compute $\mathbf{x}_{new}$ by restarted Krylov subspace method (IV.6);
    Estimate $err$ by (IV.8);
    $tol = \frac{h}{t_f} TOL$;
    if $err > tol$ then
        Reduce $h$ and re-compute $\mathbf{x}_{new}$;
    else if $err < tol$ then
        Estimate $h_{new}$ by repeated tuning to fully use error margin;
    else
        $h_{new}$ is unchanged
    end
    $t = t + h$;
    $\mathbf{x}(t) = \mathbf{x}_{new}$;
    $h = h_{new}$;
end
```

IV.3 Parallel Matrix Exponential Method

In this section, we present the parallel version of MEXP. We focus on the sparse matrix-vector multiplication—one of the key components in restarted MEXP that shows strong potential for parallelism. The basic idea of parallel sparse matrix-vector multiplication is to simultaneously calculate each row of the product vector. Many researchers [8, 9, 50] have investigated and parallelized such matrix arithmetic on different environments, such as FPGA, cluster and GPU. In this chapter, we target the parallel restarted MEXP using the GPU platform for two reasons. First, GPU has better cost-toperformance ratio. Designers can adopt the parallel restarted MEXP with affordable cost. Second, the communication overhead of the sparse matrix-vector multiplication is mitigated since the communication is now inter-thread instead of intermachine.

Our parallel restarted MEXP uses a hybrid CPU-GPU implementation. We only parallelize Arnoldi process and matrix exponential operation of a smaller matrix while keeping other operations serial on CPU. For Arnoldi process, the parallel matrix-vector multiplication is implemented by [8]. Although the parallel sparse matrix-vector multiplication has up to an order of magnitude speedup comparing to CPU implementation, the limited memory on GPU ($2GB \sim 4GB$) imposes a restriction on the dimension of Krylov subspace method for large-scale circuits. Fortunately, with the restarting scheme, MEXP method can make the effective m sufficiently large under restricted memory resource.

In the computation of matrix exponential, even though the reduced matrix by Krylov subspace method can be efficiently evaluated on CPU, the restarting scheme would increase the dimension of matrix up to hundreds, and the performance of evaluation on CPU will significantly drop. We implement the parallel matrix exponential based on a scaling and squaring method [36], which involves only basic dense matrix arithmetic that has already been optimized in the GPU environment [50].

Besides the parallelization on GPU, we minimize the data transfer cost between GPU and CPU that is one potential performance hazard of the hybrid implementation. To minimize the memory transfer, we keep the intermediate matrices, e.g., $\mathbf{V}_m$ and $\mathbf{H}_m$, on GPU, and consecutively execute Arnoldi process and matrix exponential computation. Thus, we only transfer the solution vector of the next time step back to CPU. For linear circuits, we transfer **G** and **L** and **U** decomposed from **C** at the beginning of MEXP since those linear elements remain the same during simulation. For nonlinear circuits, even though **C** of every time step changes, we do not have to transfer matrices for every Newton's iteration. This is because we decouple the linear and nonlinear terms, and thus, at each time step, only one data transmission for **C** is required. The execution flow of Algorithm 3 between CPU and GPU are shown in Figure IV.1.

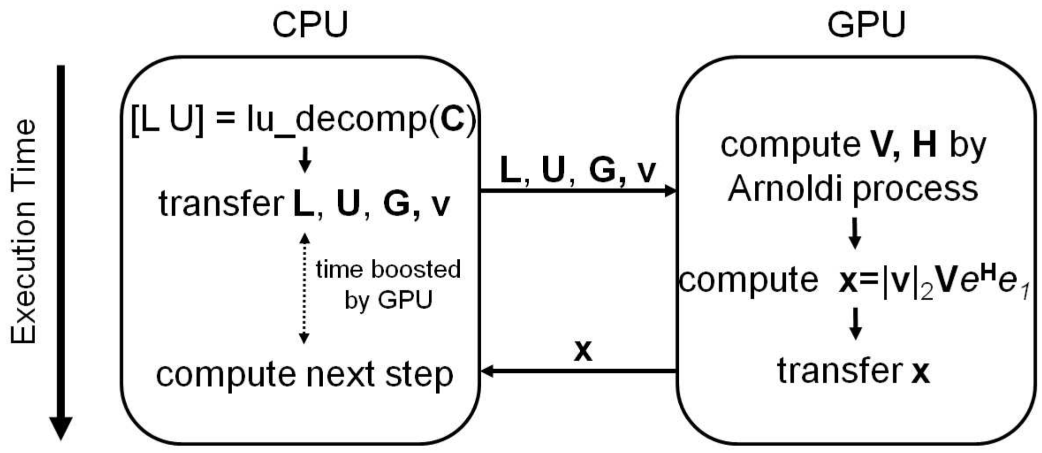


Figure IV.1: Execution flow between CPU and GPU.

We would like to mention that the backward and forward substitutions for **L** and **U** are also parallelized on GPU [50]. For large-scale circuit that cannot be decomposed, we adopt iterative approaches that are also based on matrix-vector product, and then solve **C** on GPU in parallel.

IV.4 Numerical Results

The experiments are performed on a Linux machine with an Intel i7 2.67GHz processor and 4GB memory. The parallel operations are implemented on a NVIDIA C1060 device with Tesla T10 architecture and maximum 77.6 GFLOPs for double precision. The MEXP method is prototyped in MATLAB, and the parallel restarting Krylov subspace method and matrix exponential computation are implemented in CUDA. The LU decomposition is performed by KLU package [19].

IV.4.A A Case Study of Scaling Effect

We analyze the scaling effect with the following simple model problem mainly for the ease of reproduction. One can also use a RC ladder and obtain similar observations. Let $\mathbf{A}$ be a 100×100 diagonal matrix with the diagonal elements range logarithmically in $-[10^{-2}, 10^{2}]$, i.e. $-logspace(-2, 2, 100)$, and $\mathbf{x}_0 = [1, 1, ..., 1]^T / \sqrt{100}$. We run 100 steps in the form of (IV.1), and record for each step the eigencomponents of the solution vector ($\mathbf{y}$) and $\mathbf{r}$ in (IV.4). Adaptive step is applied based on the posterior error estimate (constant local tolerance is used $tol = 10^{-6}$). The number of interpolation points $m = 10$. Fig. IV.2 shows the eigencomponents for 6 selected eigenvalues with different magnitudes.

In the upper subfigure of Fig. IV.2, the eigencomponents for fast mode (negative eigenvalues with large magnitude) generally attenuate rapidly, while the eigencomponents for slow mode (the two smallest eigenvalues) stay nearly constant. The relatively slow and oscillating drop of the eigencomponent of -100 is due to the oscillation of the maximal Ritz value [27], which induces large interpolation error and weaker attenuation when the two values are out of phase. In the lower subfigure of Fig. IV.2, the eigencomponents of large eigenvalues in the error vector are heavily attenuated, while the contribution from small eigenvalues remains at the same order, indicating that the error in later stage is largely determined by the small eigenvalues. The attenuation of some components in the error vector

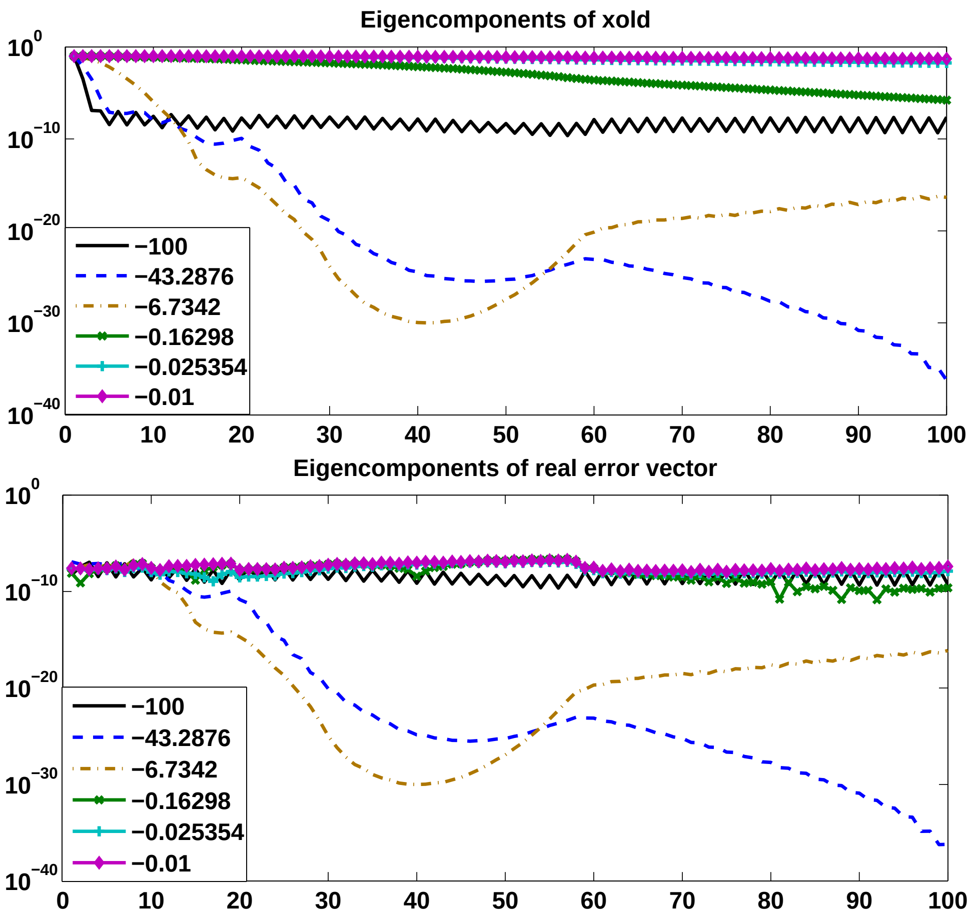


Figure IV.2: Eigencomponent vs. # of steps ($m = 10$, fixed $tol = 10^{-6}$)

reduces the norm of error and thus allows the usage of gradually increasing h in later steps under the same tolerance, which is shown in Fig. IV.3.

In Fig. IV.3, when m is set as 10, the sum of h over 100 steps is 68.42, and the ratio of the initial h to the largest allowable h is 22.55. When we increase m to 15, the increased m reduces the interpolation error and thus allows a larger step size in each step. The sum of h for $m = 15$ achieves 196.52, which is nearly triple of that for $m = 10$, with just a half increase of computational cost, while the ratio of the initial and the largest allowable h is still 22.55. Therefore, it is beneficial to use a larger m in large, stiff, problems, realized by the restarting of

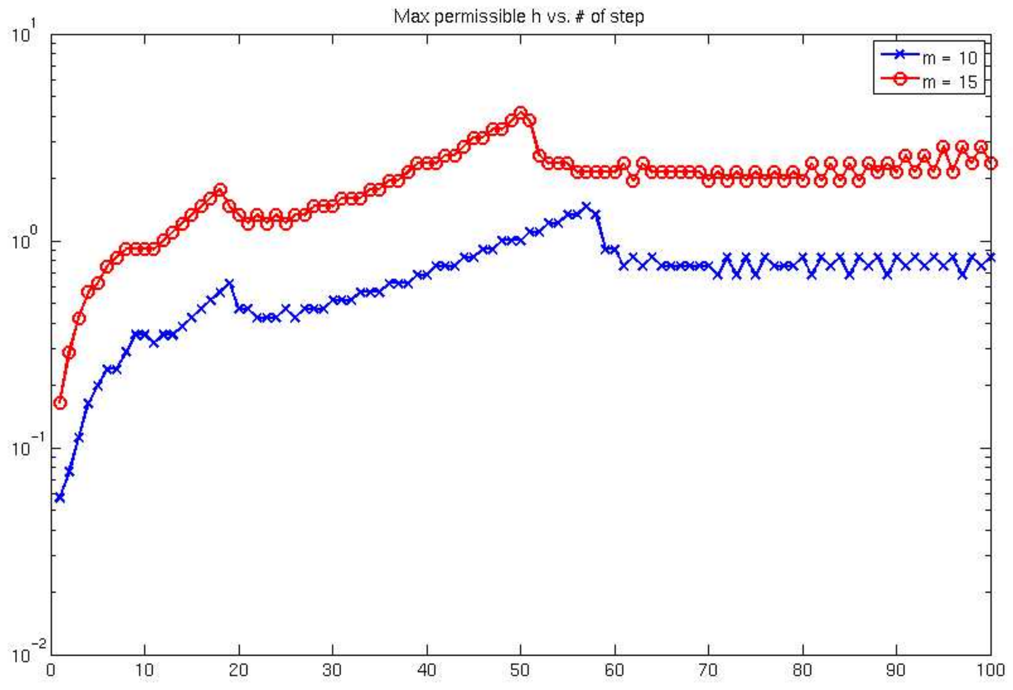


Figure IV.3: Max permissible h vs. steps for $m = 10$ and $m = 15$ (fixed $tol = 10^{-6}$)

Krylov subspace method.

IV.4.B Performance of Restarted MEXP

Table IV.1 details the functionality, size, type (L for linear and NL for nonlinear), stiffness of circuits and also the number of nodes without grounded capacitance for each benchmark circuit. We represent the stiffness of a circuit with the largest generalized eigenvalue of the matrix pencils ($\mathbf{G}$, $\mathbf{C}$). Highly stiff circuits have a value ranged from $10^{16} \sim 10^{20}$. Table IV.2 records the result of regularization process in Section III.3 for the cases (D2, D3, and D6). From the table, we can see that the number of nonzeros of $\mathbf{C} + \mathbf{G}$ before and after the regularization process is not affected significantly and even decreased because some elements are eliminated. Furthermore, the regularization process still maintains a reasonable condition number of $\mathbf{C}_r$ for inverse and spends acceptable runtime for each cases.

Table IV.1: Specifications of benchmark circuits

Design	Category	Type	Nodes	Nodes w/o Cap.	Stiffness
D1	power grid	L	2.5K	0	3.9×10^{17}
D2	trans. line	L	5.6K	431	1.6×10^{19}
D3	ALU	NL	10K	373	8.7×10^{18}
D4	IO	NL	630K	0	1.6×10^{20}
D5	power grid	L	800K	0	2.6×10^{14}
D6	power grid	L	1.6M	0.6M	1.6×10^{17}
D7	power grid	L	12M	0	2.6×10^{14}

Table IV.2: Result of regularization

Design	cond($\mathbf{C}_r$)	nnz($\mathbf{C} + \mathbf{G}$)	nnz($\mathbf{C}_r + \mathbf{G}_r$)	runtime
D2	1.1×10^{7}	0.9M	0.9M	5.9s
D3	4.4×10^{5}	44K	43K	1.2s
D6	1.4×10^{6}	5.4M	4.8M	191.2s

Table IV.3 shows the performance gained from the scaling effect. We compare the performance of ordinary MEXP with adaptive step size (MEXP) and restarted MEXP with both restarting and adaptive control (RMEXP) as outlined in Algorithm 3, where the number of restarting is 5. In addition, we implement the trapezoidal method (TRAP) with adaptive step control as the baseline performance of the circuit simulation. All three methods adopt the same adaptive scheme in Algorithm 3, and the total error budget TOL is 10^{-4} for all cases. The total simulation time and the initial step size for each case are denoted in columns t_f and "init. h", respectively.

For MEXP and TRAP, TRAP outperforms MEXP only in small case D1 because a linear system can be solved efficiently in such scale. As the size of circuit becomes large, TRAP is slowed down by solving a large linear system while MEXP benefits from the scaling effect and the sparse matrix-vector multiplication. For stiff cases D2, D3 and D4, MEXP achieves a 2.06$\times$ speedup over TRAP by the scaling effect on average. For moderately stiff cases (D5 and D7), since MEXP can use much larger step size than that in stiff cases, MEXP outperforms TRAP by over 100 times. MEXP also demonstrates the scalability in the cases with millions of nodes (D6 and D7), while TRAP encounters the scalability issues in runtime and memory. Notice that although both MEXP and TRAP have to perform Newton's method for nonlinear circuits, the Jacobian in our formulation has much fewer non-zeros than that in the traditional implicit methods. For example, in D4, solving the Jacobian in MEXP takes only 0.73 seconds whereas TRAP requires 6.86 seconds due to those extra non-zeros.

Table IV.3: Performance comparison (same adaptive step scheme with Tol = 10^{-4} for all methods and circuits)

Design	t_f	init. h	m	TRAP	# steps	MEXP	# steps	RMEXP	# steps	RMEXP-GPU	comm.
D1	1ns	1ps	30	67.85s	151	475.46s	2,595	199.10s	143	348.55s	0.8ms
D2	100ps	0.1ps	30	3,085.91s	390	2,113.51s	165	891.03s	56	982.14s	41.1ms
D3	100ps	0.1ps	30	8,053.45s	451	2,502.30s	285	572.54s	43	535.93s	53.7ms
D4	10ps	0.01ps	20	10,071.40s	357	6,646.38s	333	1,384.35s	92	194.11s	396.0ms
D5	1ns	1ps	20	16,431.12s	483	159.73s	249	83.20s	47	7.20s	121.5ms
D6	10ps	0.01ps	30	>1 day	n/a	12,626.88s	133	3,114.84s	34	239.46s	189.8ms
D7	1ns	1ps	20	fails	n/a	12,498.83s	223	7,821.65s	74	629.56s	1.5s

Our restarting scheme can further improve the scaling effect in RMEXP. For linear cases with high stiffness (D1, D2 and D6), RMEXP improves the performance over MEXP by up to 4.1×. For the nonlinear cases (D3 and D4), our nonlinear formulation in RMEXP still takes advantage of the scaling effect that improves the performance up to 4.8 times, because the decoupling scheme minimizes the calculation of the matrix exponential during Newton's iterations. Overall, RMEXP achieves 3.6× speedup over MEXP on average, and achieves an average of 8.25× speedup over TRAP on stiff cases D2, D3 and D4.

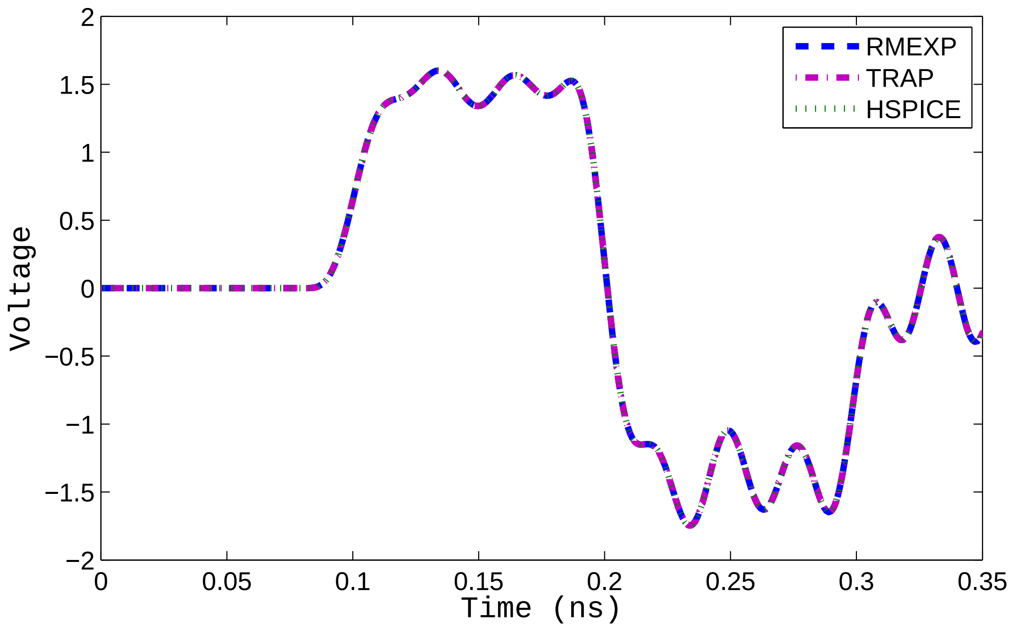


Figure IV.4: Results of D5 by RMEXP and HSPICE

Fig. IV.4 shows the waveforms by TRAP and RMEXP as well as the golden reference waveform by HSPICE for testcase D5. The errors to the golden reference for both waveforms of TRAP and RMEXP are 5.78×10^{-4} and 2.02×10^{-4}, respectively.

Fig. IV.5 shows the speedup of the parallel Krylov subspace method (Krylov-GPU) and computation of $e^{\mathbf{H_m}}$ ($\exp(\mathbf{H_m})$) over the serial version. For small matrices, Krylov-GPU and $\exp(\mathbf{H_m})$ accelerate little in the throughput-oriented GPU.

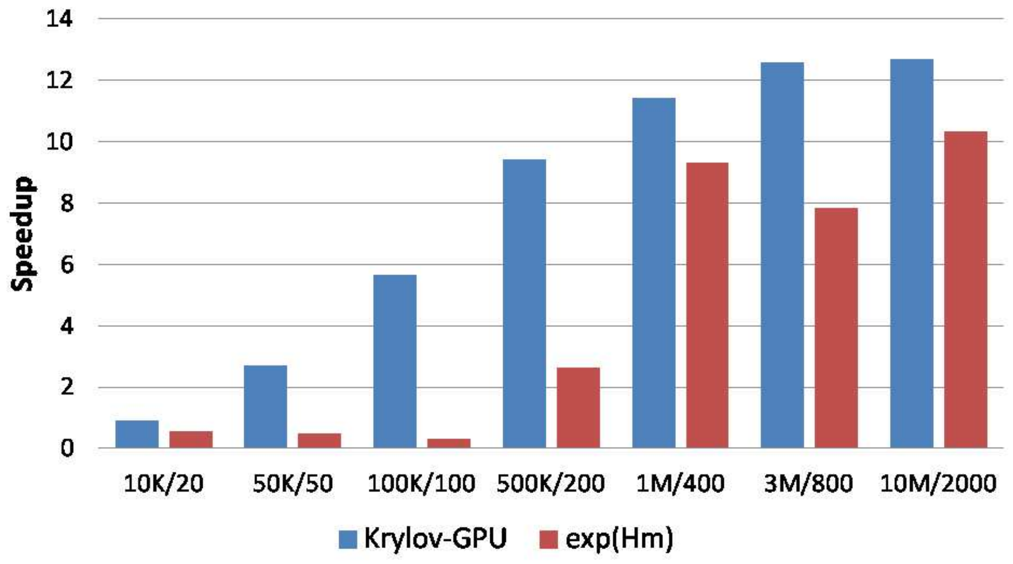


Figure IV.5: Speedup of core operations by GPU

For example, Krylov-GPU and exp($\mathbf{H_m}$) gain 0.9× and 0.5× speedup for the matrix size of 10K and 20. As the matrix size goes up, the massive parallelism of GPU surpasses faster floating point operations of CPU. Then, both operations can achieve 12× and 10× speedup for the size of 10M and 2000. Overall, with integrating both parallel operations into Algorithm 3, the performance of parallel restarted MEXP (RMEXP-GPU) (also shown in Table IV.3) presents an average of 11× speedup for large-scale cases, compared with the serial RMEXP.

Take the largest case D7 as an example. TRAP fails the simulation due to insufficient memory while MEXP and RMEXP require about 3.5 and 2.2 hours, respectively. RMEXP-GPU shows the simulation takes only 10 minutes using GPU. The speedup by GPU over MEXP and RMEXP are 19.9× and 12.4×, respectively.

Note that the communication overhead between CPU and GPU is negligible since we minimize the data transmission in the hybrid CPU-GPU architecture. Although, for nonlinear circuits, the change of $\mathbf{C}$ and $\mathbf{G}$ by nonlinear devices at every time step increases the communication between CPU and GPU, such overhead is still negligible. The column “comm.” in Table IV.3 shows the time of transferring **L**, **U** and **G** once from CPU to GPU. We can see that even the largest case D7 requires only about 1.5 seconds. This is because the sparsity of the matrices in the circuit simulation application greatly reduces the transferred data, and also KLU [19] maintains acceptable sparsity in both **L** and **U** matrices.

IV.5 Summary

In this chapter, we propose a parallel and restarted matrix exponential to utilize the scaling effect, which can overcome the stiffness of circuitry. The scaling effect enables the use of gradually larger step size as time frame marches forward. Moreover, our method has better scalability and parallelizability, which are the major limitations of existing implicit methods. The experimental results show that MEXP has a speedup over the trapezoidal method in orders of magnitude for cases with millions of nodes. For stiff circuits, our restarted MEXP improves the performance of MEXP by up to 4.8 times, and the performance can be further accelerated by up to another 11 times by GPU parallelization. For cases with millions of nodes, our method is able to simulate with acceptable runtime and memory usage while the trapezoidal method breaks down. The superior scalability and parallelizability of the proposed method enable a substantial expansion of computational capability for modern extremely large circuit simulation problems.

Chapter IV includes the content of one published conference paper, "Circuit Simulation using Matrix Exponential Method for Stiffness Handling and Parallel Processing," by Shih-Hung Weng, Quan Chen and Chung-Kuan Cheng, in Proceedings of *2012 IEEE International Conference on Computer-Aided Design*. The dissertation author was the primary investigator and author of the paper.

V

Matrix Exponential Method with Rational Krylov Subspace Approximation

In this chapter, we propose the matrix exponential method with the rational Krylov subspace approximation. The kernel operation in our method only demands one factorization and backward/forward substitutions. Moreover, the rational Krylov subspace approximation can relax the stiffness constraint and avoid regularization process shown in the previous chapters III and IV. The cheap computation of adaptivity in the matrix exponential method fits well for the power grid simulation. In such application, our method could exploit the long low-frequency response in a power grid and significantly accelerate the simulation. The experimental results show that our method achieves up to 18X speedup over the trapezoidal method with fixed step size. Our method also demonstrates a promising parallelism in the perspective of input sources.

V.1 Background

Power grid simulation plays an important role in IC design methodology. Given current stimulus and the power grid structure, designers could verify and predict the worst-case voltage noise through the simulation before signing off their design. However, with the huge size of modern design, power grid simulation is a time-consuming process. Moreover, manifesting effects from the package and the board would require longer simulation time, e.g., up to few μs, which worsens the performance of the power grid simulation. Therefore, an efficient power grid simulation is always a demand from industry.

Conventionally, the power grid simulation is by the trapezoidal method where the major computation is to solve a linear system by either iterative approaches [57] or direct methods [19]. Many previous works [15] [28] [42] [48] [64] utilize iterative approaches to tackle the scalability issue and enable adaptive stepping in the power grid simulation. However, the iterative methods usually suffer from the convergence problem because of the ill-conditioned matrix from the power grid design. On the other hand, the direct methods, i.e., Cholesky or LU factorizations, are more general for solving a linear system. Despite the huge memory demanding and computational effort, with a carefully chosen step size, the power grid simulation could perform only one factorization at the beginning while the rest of operations are just backward/forward substitutions. Since a power grid design usually includes board and package models, a long simulation time is required to manifest the low-frequency response. Hence, the cost of expensive factorization can be amortized by many faster backward/forward substitutions. Such general factorization and fixed step size strategy [73] [75] [78] [79] is widely adopted in industry.

In this chapter, we tailor MEXP using *rational Krylov subspace* for the power grid simulation with adaptive time stepping. The rational Krylov subspace uses $(\mathbf{I}-\gamma\mathbf{A})^{-1}$ as the basis instead of $\mathbf{A}$ used in the conventional Krylov subspace.

The rational basis limits the spectrum of a circuit so that the exponential of a matrix can be accurately approximated even under a large step size. Moreover, the rational avoids $\mathbf{C}^{-1}$ and thus eliminates the regularization process. As a result, MEXP with rational Krylov subspace is free from the stiffness constraint and the regularization process. The simulation can purely enjoy benefits of the adaptivity and the accuracy of MEXP. Even though the rational Krylov subspace still needs to solve a linear system as the trapezoidal method does, MEXP can factorize the matrix only once and then constructs the rest of rational Krylov subspaces by backward/forward substitutions. Therefore, MEXP can utilize its capability of adaptivity to accelerate the simulation with the same kernel operations as the fixed step size strategy. Overall, our MEXP for the power grid simulation has the following advantages:

- Enabling adaptive time stepping for the power grid simulation with only one LU factorization.
- Eliminating the regularization process for the singular power grid design.
- Allowing scaling large step size without compromising the accuracy.

The experimental results demonstrate the effectiveness of MEXP with adaptive step size. The power grid simulation for IBM benchmark suite [47] and industrial power grid designs can be accelerated 6X and 15X on average compared to the trapezoidal method.

V.2 Power Distribution Network Modeling

A power distribution network (PDN) is shown in Figure V.1. The purpose of the network is to supply power to integrated circuits. A PDN includes interconnect with decoupling capacitance (decap) on a printed circuit board (PCB), an integrated circuit packae and a circuit die. As illustrated in Figure V.1, the system also contains a voltage regulator module (VRM), the PDNs on PCB, on package and on chip. A voltage regulator is to adjust the supply power to the required voltage level of the integrated circuits. There are power and ground planes connected the VRM to the package on the PCB. The on-board PDN connects supply power to package through solder balls, and the on-package PDN connects the on-board PDN to chip. Finally, the on-package PDN connects chip through bonding wires or C4 (Controlled Collapse Chip Connection) bumps. The on-chip PDN delivers power to switching transistors across the die. Different decaps are placed at each of the PCB, package, and chip levels, which serve as charge reservoirs to supply power for local currents.

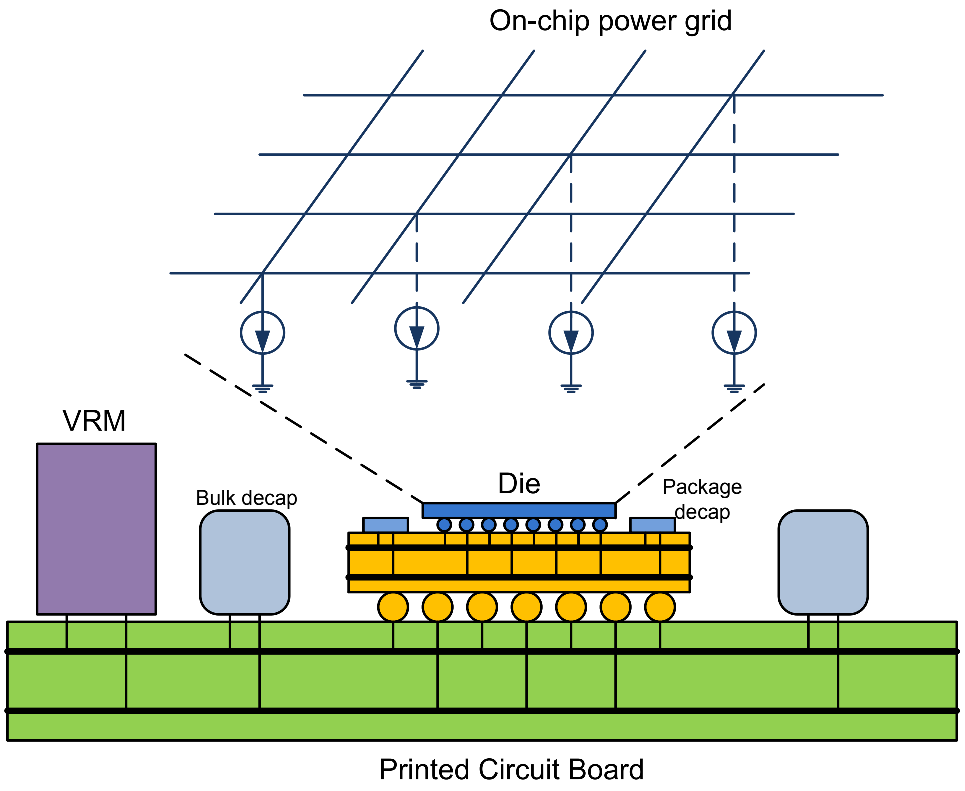


Figure V.1: A power distribution system from VRM to die. Courtesy of Xiang Hu, 2011.

A PDN can be formulated as a system of differential algebraic equations (DAEs) via modified nodal analysis (MNA) as Eqn. (II.2) where vector $\mathbf{u}(t)$ represents the corresponding supply voltage and current sources associated to different blocks. In order to stimulate the worst-case voltage noise, those current sources, which represent the behavior of underlying blocks of a power grid, are often pre-characterized with pre-designed vectors. Many previous works [11] [25] [29] have developed different methodologies to generate worst-case stimulus currents. In this work, we assume those input sources are given and in the format of *piece-wise linear*.

We would also like to mention that a PDN contains on-board, on-package and on-die components, which are ranged in different scale, e.g., capacitance is from pF to fF, inductance is from nH to pH, and resistance is ranged from $k\Omega$ to $m\Omega$. Hence, the electronic behavior of a PDN will have low-, mid- and high-frequency response. In order to manefist the low-frequency effect, such as "rogue wave" phenomenon [25], a long simulation time span ($\sim \mu s$) is necessary.

V.3 MEXP with Rational Krylov Subpsace

In chapters III and IV, Eqn. (III.16) is calculated via the Krylov subspace method using Arnoldi process. The subspace is defined as

$$\mathbf{K_m}(\widetilde{\mathbf{A}}, \mathbf{v}) = span\{\mathbf{v}, \widetilde{\mathbf{A}}\mathbf{v}, \cdots, \widetilde{\mathbf{A}}^{m-1}\mathbf{v}\}, \tag{V.1}$$

where $\mathbf{v}$ is an initial vector. The Arnoldi process approximates the eigenvalues with large magnitude well. But when handling a stiff circuit system, the formed matrix usually contains many eigenvalues with small magnitude. Besides, $e^{\widetilde{\mathbf{A}}h}$ is mostly determined by the eigenvalues with smallest real magnitudes and their corresponding invariant subspaces. In this scenario, due to the existence of eigenvalues with large magnitude in $\widetilde{\mathbf{A}}$, the Arnoldi process for Eqn. (V.1) requires large m to capture the important eigenvalues (small magnitudes) and invariant spaces for exponential operator. Therefore, the time steps in MEXP has to be small enough to capture the important eigenvalues. This suggests us transforming the spectrum to intensify those eigenvalues with small magnitudes and corresponding invariant subspaces. We make such transformation based on the idea of *rational Krylov subspace method* [46] [68]. The details are presented in the following subsections.

V.3.A Fast Rational Krylov Subspace Approximation of Matrix Exponential

For the purpose of finding the eigenvalues with smallest magnitude first, we uses a preconditioner $(\mathbf{I}-\gamma\widetilde{\mathbf{A}})^{-1}$, instead of using $\widetilde{\mathbf{A}}$ directly. It is known as the rational Krylov subspace [46] [68]. The formula for the rational Krylov subspace is

$$\begin{aligned}\mathbf{K_m}((\mathbf{I}-\gamma\widetilde{\mathbf{A}})^{-1},\mathbf{v}) = \mathit{span}\{&\mathbf{v},(\mathbf{I}-\gamma\widetilde{\mathbf{A}})^{-1}\mathbf{v},\cdots,\\ &(\mathbf{I}-\gamma\widetilde{\mathbf{A}})^{-(m-1)}\mathbf{v}\},\end{aligned} \tag{V.2}$$

where γ is a predefined parameter. The Arnoldi process constructs $\mathbf{V_m}$ and $\mathbf{H_m}$, and the relationship is given by

$$(\mathbf{I}-\gamma\widetilde{\mathbf{A}})^{-1}\mathbf{V_m} = \mathbf{V_m}\mathbf{H_m} + \mathbf{H}(m+1,m)\mathbf{v}_{m+1}e_m^T, \tag{V.3}$$

where e_m is the m-th unit vector with dimension $m \times 1$. Matrix $\mathbf{H_m}$ is an upper Hessenberg matrix, and $\mathbf{V_m}$ consists of $[\mathbf{v}_1,\mathbf{v}_2,\cdots,\mathbf{v}_m]$. After re-arranging (V.3, we can project $\widetilde{\mathbf{A}}$ onto the rational Krylov subspace as

$$\frac{1}{\gamma}(\mathbf{I}-\mathbf{H_m}^{-1}) \approx \mathbf{V_m}^T\widetilde{\mathbf{A}}\mathbf{V_m}. \tag{V.4}$$

Now given a time step h, the matrix exponential $e^{\widetilde{\mathbf{A}}h}\mathbf{v}$ can be calculated as

$$\begin{aligned} e^{\widetilde{\mathbf{A}}h}\mathbf{v} &\approx \mathbf{V_m}\mathbf{V_m}^T e^{\widetilde{\mathbf{A}}h}\mathbf{v} = \|v\|_2\,\mathbf{V_m}\mathbf{V_m}^T e^{\widetilde{\mathbf{A}}h}\mathbf{V_m}e_1 \\ &= \|v\|_2\,\mathbf{V_m}e^{\alpha\widetilde{\mathbf{H}}_\mathbf{m}}e_1, \end{aligned} \tag{V.5}$$

where $\widetilde{\mathbf{H}}_\mathbf{m} = \mathbf{I}-\mathbf{H_m}^{-1}$, $\alpha = \frac{h}{\gamma}$ is the tuning parameters for control of adaptive time step size in Section V.4. Note that in practice, instead of computing $\mathbf{A}^{-1}$

directly, we only need to solve $(\mathbf{C}+\gamma\mathbf{G})^{-1}\mathbf{C}\mathbf{v}$, which can be achieved by one LU factorization at beginning. Then the construction of the following subspaces is by backward/forward substitutions.

This strategy is also presented in the numerical analysis community [24] [46] [68]. Intuitively, the "shift-and-invert" operation would intensify the eigenvalues with small magnitudes and minify the eigenvalues with large magnitudes. By doing so, the Arnoldi process could capture those eigenvalues important to the exponential operator, which originally cannot be manifested with small m in the conventional Krylov subspace.

We would like to point out that the error bound for Eqn. (V.5) does not longer depend on $\|\widetilde{\mathbf{A}}h\|$ as Chapters III and IV. It is only the first (smallest magnitude) eigenvalue of $\widetilde{\mathbf{A}}$. The lemma is given as follow and the proof is given in [68].

Lemma V.3.1 *(Lemma 3.1 in [68]) Let μ be such that $-\widetilde{\mathbf{A}}-\mu\mathbf{I}$ is positive semi-definite. Then*

$$\left\| \|v\|_2 \mathbf{V}_m e^{\alpha\widetilde{\mathbf{H}}} e_1 - e^{\alpha\gamma\widetilde{\mathbf{A}}}\mathbf{v} \right\| \le 2e^{-\alpha\gamma\mu} E_{m-1}^{m-1}(\widetilde{\gamma}) \tag{V.6}$$

with $E_j^i(\widetilde{\gamma}) := \inf_{r\in R_i^j} \sup_{t\le 0} |r(t)-e^{-t}|$, $R_i^j = \{p(t)(1-\gamma t)^{-i} | p \in \Pi_j\}$, the space Π_{m-1} is the space of all polynomials of degree $m-1$ or less.

Note that E_{m-1}^{m-1} can be approximated as a constant upper bound here, and the eigenvalues of $-\widetilde{\mathbf{A}}$ are all larger than a positive μ. With increasing h (equivalently α) and predefined γ, it brings much tighter trend of error bound than smaller α. That explains, in numerical experiment, we observe that large α provides less error under the same dimension m. An intuitive explanation is also given by [68], the larger α combined with exponential operators, the relatively smaller portion of the eigenvalues with smallest magnitude determine the final vector. Actually, it is quite a nice feature, especially for power grid simulation, where the voltage noise usually contains low frequency behavior from board and package levels. Our method can step forward as much as possible to accelerate simulation, and still maintain the high accuracy. The sacrifice resides in the small time step when more eigenvalues determine the final vector. So we should choose a appropriate parameter γ or increase the order m to balance the accuracy and efficiency. Even though the increasing m results more backward/forward substitutions, the m is still quite small ($5 \sim 20$) in the power grid simulation. Therefore, it does not degrade our method too much. our experimental results present that our method achieves impressive speedups over the trapezoidal method while preserving low- to high-frequency responses in industrial power grid designs.

The formula of posterior error estimation is required for controlling adaptive step size. We use the formula derived from [68],

$$\begin{aligned} err(m,\alpha) = \frac{\|v\|_2}{\gamma}\widetilde{\mathbf{H}}(m+1,m)\left\|e_m^T\phi(\alpha\widetilde{\mathbf{H}}_m)e_1\right\| \\ \times\left\|(\mathbf{I}-\gamma\widetilde{\mathbf{A}})v_{m+1}\right\|, \end{aligned} \tag{V.7}$$

where $\phi(x) = \frac{e^x-1}{x}$. The formula provides a good approximation for the error trend with respect to m and α in our numerical experiment.

V.3.B Block LU factorization

In practical numerical implementation, in order to avoid direct inversion of $\mathbf{C}$ to form $\mathbf{A}$ in Eqn. (V.2), the Eqn. (V.8)

$$(\mathbf{C}+\gamma\mathbf{G})^{-1}\mathbf{C} \tag{V.8}$$

is used. Correspondingly, we uses the following equations for Eqn. (III.16),

$$(\widetilde{\mathbf{C}}-\gamma\widetilde{\mathbf{G}})^{-1}\widetilde{\mathbf{C}} \tag{V.9}$$

where

$$\widetilde{\mathbf{C}} = \begin{bmatrix} \mathbf{C} & 0 \\ 0 & \mathbf{I} \end{bmatrix}, \ \widetilde{\mathbf{G}} = \begin{bmatrix} -\mathbf{G} & \mathbf{W} \\ 0 & \mathbf{J} \end{bmatrix} \tag{V.10}$$

The Arnoldi process based on Eqn. (V.9) actually only requires to solve $\mathbf{v}_{k+1}$ with $\mathbf{v}_k$. The linear system is expressed as

$$(\widetilde{\mathbf{C}} - \gamma\widetilde{\mathbf{G}})\mathbf{v}_{k+1} = \widetilde{\mathbf{C}}\mathbf{v}_k, \tag{V.11}$$

where $\mathbf{v}_k$ and $\mathbf{v}_{k+1}$ are k-th and $k+1$-th basis in the rational Krylov subspace. If $\mathbf{W}$ changes with inputs during the simulation, the Arnoldi process has to factorize a matrix every time step. However, it is obvious that the majority of matrix components in Eqn. (V.10) stay the same for this linear system. To take advantage of this property, a block LU factorization is devised here to avoid redundant calculation.

The goal is to obtain two matrices: the lower triangular matrix $\mathbf{L}$ and the upper triangular matrix $\mathbf{U}$, such that

$$\widetilde{\mathbf{C}} - \gamma\widetilde{\mathbf{G}} = \mathbf{L}\mathbf{U}. \tag{V.12}$$

At the beginning of simulation, after LU factorization of $\mathbf{C} + \gamma\mathbf{G} = \mathbf{L}_{sub}\mathbf{U}_{sub}$ we obtains the lower triangular sub-matrix $\mathbf{L}_{sub}$, and upper triangular sub-matrix $\mathbf{U}_{sub}$. Then Eqn. (V.12) only needs updating via

$$\mathbf{L} = \begin{bmatrix} \mathbf{L}_{sub} & 0 \\ 0 & \mathbf{I} \end{bmatrix}, \quad \mathbf{U} = \begin{bmatrix} \mathbf{U}_{sub} & \mathbf{L}_{sub}^{-1}\mathbf{W} \\ 0 & \mathbf{I}_J \end{bmatrix}, \tag{V.13}$$

where $\mathbf{I}$ is an identity matrix, $\mathbf{I}_J = \mathbf{I} + \mathbf{J}$ is an upper triangular matrix. The following equations further reduce operation $\mathbf{L}_{sub}^{-1}$ and construct vector $\mathbf{v}_{k+1}$.

$$\mathbf{L}_{sub}\mathbf{U}_{sub}\,\mathbf{y}_1 = \mathbf{b}_1 - \mathbf{W}\,\mathbf{y}_2, \quad \mathbf{y}_2 = \mathbf{I}_J^{-1}\mathbf{b}_2, \tag{V.14}$$

where $\mathbf{v}_{k+1} = [\mathbf{y}_1,\ \mathbf{y}_2]^T, \mathbf{b}_1 = [\mathbf{C},\ 0]\mathbf{v}_k, \mathbf{b}_2 = [0,\ \mathbf{I}]\mathbf{v}_k$. By doing this, it only needs one LU factorization at the beginning of simulation, and with cheap updates for the $\mathbf{L}$ and $\mathbf{U}$ at each time step during transient simulation.

V.3.C Compatibility with Singular C without Regularization

Another contribution is that our method can handle singular $\mathbf{C}$ without regularization in the rational Krylov subspace. Even we cannot formulate Eqn. (III.1) explicitly, but the solution can be still represented as the combination of eigenvalues and eigenvectors [72]. To calculate such eigenvalues, the problem turns into the generalized eigenvalue problem for matrix pencil $(-\mathbf{G}, \mathbf{C})$. There are n eigenvalues $\lambda_1, \lambda_2, \cdots, \lambda_n$, whose real parts are negative in the power grid simulation. The solution can be expressed as $\mathbf{x}(t+h) = \mathbf{Q}_n e^{h\Sigma_n} \mathbf{P}_n \mathbf{x}(t),$, where $\mathbf{Q}_n \mathbf{P}_n = \mathbf{I}$, and $\mathbf{I}$ is an identity $n \times n$ matrix. The only hindrance is the existence of "$-\infty$" eigenvalues without regularization in Chapter III. However, we can prove these eigenvalues can be eliminated in the solution, which is given in the following lemma.

Lemma V.3.2 *When all generalized eigenvalues of matrix pencil $(-\mathbf{G}, \mathbf{C})$ are negative, there exist r bounded eigenvalues and $\forall i > r,\ \ \lambda_i = -\infty$. The formula,*

$$\mathbf{x}(t+h) = \mathbf{Q}_n e^{h\Sigma_n} \mathbf{P}_n \mathbf{x}(t),$$

can be written as

$$\mathbf{x}(t+h) = \mathbf{Q}_r e^{h\Sigma_r} \mathbf{P}_r \mathbf{x}(t),$$

where

$$\mathbf{\Sigma}_j = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_j \end{bmatrix}$$

$\mathbf{Q}_j = [\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_j]$, $\mathbf{P}_j = [\mathbf{p}_1; \mathbf{p}_2; \cdots; \mathbf{p}_j]$, *and* $\mathbf{q}_i$ *is* $n \times 1$ *vector,* $\mathbf{p}_i$ *is* $1 \times n$ *vector.*

Proof 4

$$\mathbf{x}(t+h) = \mathbf{Q}_n e^{h\Sigma_n} \mathbf{P}_n \mathbf{x}(t),$$

$$\begin{aligned}
\mathbf{x}(t+h) &= [\mathbf{Q}_r \mathbf{Q}_{n-r}] e^{h\begin{bmatrix} \mathbf{\Sigma}_r & 0 \\ 0 & \mathbf{\Sigma}_{n-r} \end{bmatrix}} \begin{bmatrix} \mathbf{P}_r \\ \mathbf{P}_{n-r} \end{bmatrix} \mathbf{x}(t) \\
&= (\mathbf{Q}_r e^{h\mathbf{\Sigma}_r} \mathbf{P}_r + \mathbf{Q}_{n-r} e^{h\mathbf{\Sigma}_{n-r}} \mathbf{P}_{n-r}) \mathbf{x}(t) \\
&= \mathbf{Q}_r e^{h\mathbf{\Sigma}_r} \mathbf{P}_r \mathbf{x}(t)
\end{aligned}$$

where

$$e^{h\mathbf{\Sigma}_{n-r}} = \begin{bmatrix} e^{-\infty} & 0 & \cdots & 0 \\ 0 & e^{-\infty} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & e^{-\infty} \end{bmatrix} = \mathbf{0}$$

By this lemma, we can take out the "$-\infty$" eigenvalues.

In addition, it is known that eigenvalues of Hessenberg matrix of Arnoldi process are often used as the estimation for the original matrix [63]. The rational Krylov subspace is also utilized to preserve the bounded eigenvalues of matrix pencil and corresponding eigenvectors as long as m is enough [43]. Therefore, by using Arnoldi process via the rational Krylov subspace for $(-\mathbf{G}, \mathbf{C})$ It holds

$$\begin{aligned}
\mathbf{x}(t+h) &= \mathbf{Q} e^{h\mathbf{\Sigma}_n} \mathbf{P} \mathbf{x}(t) = \mathbf{Q}_r e^{h\mathbf{\Sigma}_r} \mathbf{P}_r \mathbf{x}(t) \\
&\approx \mathbf{V_m} e^{\alpha \widetilde{\mathbf{H}}_\mathbf{m}} \mathbf{V_m}^T \mathbf{x}(t).
\end{aligned}$$

The above equation illustrates that, without modification of our method using the rational Krylov subspace, our method is compatible with the cases that contain singular $\mathbf{C}$.

Intuitively speaking, "$-\infty$" eigenvalues and corresponding eigenvectors do not effect the whole dynamical system via time derivatives here due to the exponential operator; it just forms the algebraic relations in the DAEs [35]. Previously, MEXP in Chapters III and IV cannot handle $\mathbf{C}$ without regularization is only because the Arnoldi process requires to inverse $\mathbf{C}$. Our algorithm never uses the inverse of $\mathbf{C}$ directly.

V.4 Adaptive Time Step Control

The proposed MEXP can significantly benefit from the adaptive time stepping because the rational Krylov subspace approximation relaxes the stiffness constraint as well as preserves the scaling invariant property. As a result, MEXP can effortlessly adjust the step size to different scale, e.g., from $1ps$ to $1ns$, during the simulation. Such adaptivity is particularly helpful in the power grid where the voltage noise includes the high- to low-frequency responses from die, package and board.

Our adaptive step scheme is to step forward as much as possible so that MEXP can quickly finish the simulation. With the insight from Eqn. (V.5), MEXP can adjust α to calculate results of required step sizes with only one Arnoldi process. However, even though the rational Krylov subspace could scale arbitrarily, the step size in MEXP is restrained from input sources. As shown in Eqn. (III.2), MEXP has to guarantee constant slope during a stepping, and hence the maximum allowed step size h_{max} at every time instant is limited. Our scheme will first determine h_{max} from inputs at time t and construct the rational Krylov subspace from $\mathbf{x}(t)$. Then, $\mathbf{x}$ within interval $[t, \quad t+h_{max}]$ are calculated through the step size scaling.

Algorithm 4 shows MEXP with adaptive step control. In order to comply with the required accuracy during the simulation, the allowed error err_k at certain time instant t_k is defined as

$$err_k \leq \frac{E_{Tol}}{T} \times h_k,$$

where E_{Tol} is the error tolerance in the whole simulation process, T is the simulation time span, h_k is the step size at time t_k, and err_k is the posterior error of MEXP from Eqn. (V.7). Hence, when we construct the rational Krylov subspace, we will increase m until the err_k satisfies the error tolerance.

The complexity of MEXP with adaptive time stepping is mainly determined by the total number of required backward/forward substitutions during the simu-

Algorithm 4: MEXP with Adaptive Step Control

```
Input: $\mathbf{C}$, $\mathbf{G}$, $\mathbf{B}$, $\mathbf{u}(t)$, $\tau$, error tolerance $E_{Tol}$ and simulation time
       $T$
Output: $\mathbf{x}(t)$
$t = 0$; $\mathbf{x}(0)$ = DC_analysis;
$[\mathbf{L}_{sub}, \mathbf{U}_{sub}]$ =LU$(\mathbf{C} + \gamma\mathbf{G})$;
while $t \leq T$ do
    Compute maximum allowed step size $h$ from $\mathbf{u}(t)$;
    Construct $\mathbf{H_m}$, $\mathbf{V_m}$, $err_k$ by Arnoldi process and (V.7) until
    $err_k \leq \frac{E_{Tol}}{T} h$;
    $\alpha = \frac{h}{\gamma}$ ;
    Compute $\mathbf{x}(t + h)$ by (V.5);
    $t = t + h$;
end
```

lation process. This is because the dimension of $\mathbf{H_m}$ is small, e.g., around $5 \sim 20$, and the computation overhead of the step size scaling is negligible. The number of total substitution operations is

$$\sum_{i=0}^{N} m_i,$$

where N is total time steps, and m_i is required dimension of the rational Krylov subspace at time step i. Compared to the trapezoidal method where the number of substitution operations depends only on the fixed step size, MEXP could use less substitution operations as long as the maximum allowed step size h_{max} is much larger than the fixed step size. Our experiments in the following section demonstrates it is usually the case for the power grid simulation.

V.5 Experimental Results

In this section, we compare performance of the power grid simulation by MEXP and the trapezoidal method (TRAP). MEXP with adaptive step size control follows Algorithm 4. We set γ as 10^{-10} and restrict the maximum allowed step size within $1ns$ to have enough time instants to plot the figure. It is possible to have more fine-grain time instants, e.g., $10ps$, with only negligible cost by adjusting α in Eqn. (V.5). TRAP is in fixed step size in order to minimize the cost of LU factorization. Both methods only perform factorization once, and rest of operations is mainly backward/forward substitution. We implement both methods in MATLAB and use UMFPACK package for LU factorization. Note that even though previous works [15] [64] show that using iterative approach in TRAP could also achieve adaptive step control, long simulation time span in power grid designs make direct method with fixed step size more desirable [73] [75] [79].

Our benchmark includes different industrial power distribution networks (PDNs), which contain the model of board and package, and also includes IBM power grid suite [47]. Designs in both benchmarks all have singular $\mathbf{C}$. The experiments are performed on a Linux workstation with an Intel i7 2.67GHz processor and 12GB memory. The details and simulation results of both industrial designs and IBM power grid suite are presented in the following subsections.

V.5.A Industrial PDN Designs

The industrial PDN design includes on-chip power grid, package and board. The power grid consists of four metal layers: M1, M3, M6 and RDL. The physical parameters of each metal layer is listed in Table V.1. The package is modeled as an RL series at each C4 bump, and the board is modeled as a lumped RLC network. The specification of each PDN design is listed in Table V.2 where the size of each design ranges from 45.7K to 7.40M.

In order to characterize a PDN design, designers can rely on the simulation

Table V.1: Widths and pitches of metal layers in the PDN design(μm).

M1		M3		M6		RDL	
pitch	width	pitch	width	pitch	width	pitch	width
2.5	0.2	8.5	0.25	30	4	400	30

Table V.2: Specifications of PDN Designs

Design	Area (mm^2)	#R	#C	#L	#Nodes
D1	0.35^2	23221	15193	15193	45.7K
D2	1.40^2	348582	228952	228952	688K
D3	2.80^2	1468863	965540	965540	2.90M
D4	5.00^2	3748974	2467400	2464819	7.40M

result of impulse response of the PDN design. Many previous works [25] [67] have proposed different PDN analysis based on the impulse response. The nature of impulse response of the PDN design, which contains low-, mid- and high-frequency components, can significantly enjoy the adaptive step size in MEXP. We would also like to mention that the impulse response based analysis is not only for the PDN design, but also for worst-case eye opening analysis in the high speed interconnect [4] [62].

The impulse response can be derived from the simulation result of a step input from 0V to 1V with a small transition time. Hence, we inject a step input to each PDN design and compare the performance of MEXP and TRAP. The transition time of the step input and the simulation time span is $10ps$ and $1\mu s$ for observing both high- and low-frequency responses. Table V.3 shows the simulation runtime of MEXP and TRAP where the fixed step size is set as $10ps$ to comply with the transition time. In the table, “DC”, “LU” and “Time” indicate the runtime for DC analysis, LU factorization and the overall simulation, respectively. DC analysis is also via the LU factorization. We can also adopt other techniques [73] [75] [78] to improve the performance of DC analysis for both methods. “Spdp” shows the speedup of MEXP over TRAP.

Table V.3: Simulation runtime of PDN designs

Design	DC(s)	TRAP ($h = 10ps$)		MEXP ($\gamma = 10^{-10}$)		
		LU(s)	Total	LU(s)	Total	Spdp
D1	0.71	0.67	44.85m	0.68	2.86m	15.73
D2	12.21	15.60	15.43h	15.48	54.57m	16.96
D3	69.59	91.60	76.92h	93.28	4.30h	17.91
D4	218.89	293.81	203.64h	298.83	11.26h	18.08

MEXP has significant speedup over TRAP because MEXP can exploit much large step size to simulate the design whereas TRAP can only step in $10ps$ for whole $1\mu s$ time span. The average speedup is 17X. Figure V.2 shows the simulation result of design D1 at a node on M1. As we can see, the result by MEXP and TRAP are very close to the result of HSPICE, which is as our reference result here. The errors of MEXP and TRAP to HSPICE are 7.33×10^{-4} and 7.47×10^{-4}. This figure also demonstrates that a PDN design has low-, mid- and high-frequncy response so that long simulation time span is necessary.

V.5.B IBM Power Grid Suite

In this subsection, we compare MEXP and TRAP using the IBM power grid suite [47]. The simulation time is $10ns$, and TRAP uses fixed step size in $10ps$. Table V.4 shows the details of each benchmark circuit of which size ranges from 54K up to 3M.

Table V.5 presents the performance gains from MEXP compared to TRAP. MEXP shows a little speedup over TRAP that is not significant as that of the industrial PDN designs. The average of the speedup is 1.42 times. The reason behind the little speedup in IBM benchmark suite is the fact that hundreds of thousands of input sources limits the maximum allowed step size. The simulation

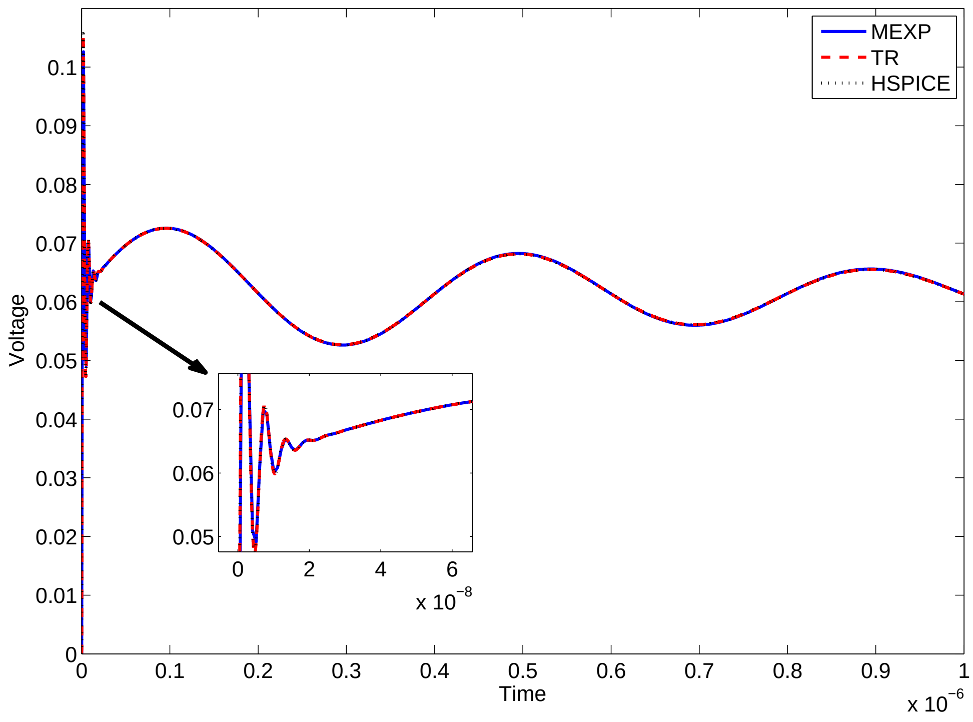


Figure V.2: Result of D1

results of ibmpg6t via MEXP and TRAP are shown in Figure V.3 where MEXP has many small step sizes so that the benefit of large step size does not improve the performance drastically. As we mentioned in Section V.4, the maximum allowed step size is actually determined by the slope of input sources over a time interval. Figure V.4 illustrates the concept where I_1 and I_2 are input sources and “max. step” shows the maximum allowed step size during the simulation. As we can see, ill alignment causes tiny step size because MEXP has to guarantee constant slopes within a step size so that Eqn. (III.2) can be held. The ibmpg4t has better alignment than the other benchmark circuits, and thus, MEXP achieves 2.95 speedup in ibmpg4t over TRAP.

We can avoid the ill alignment issue by simply grouping “similiar” input sources and then simulate them in separately. Due to the linearity of the power grid, the final result is just the superposition of the simulation result with each

Table V.4: Specifications of IBM power grid suite

Design	#R	#C	#L	#I	#V	#Nodes
ibmpg1t	40801	10774	277	10774	14308	54265
ibmpg2t	245163	36838	330	36838	330	164897
ibmpg3t	1602626	201054	955	201054	955	1043444
ibmpg4t	1826589	265944	962	265944	962	1214288
ibmpg5t	1550048	473200	277	473200	539087	2092148
ibmpg6t	2410486	761484	381	761484	836249	3203802

Table V.5: Performance comparison TRAP v.s. MEXP

Design	DC(s)	TRAP ($h = 10ps$)		MEXP ($\gamma = 10^{-10}$)		
		LU(s)	Total(s)	LU(s)	Total(s)	Spdp
ibmpg1t	0.28	0.24	11.02	0.24	9.53	1.16
ibmpg2t	1.12	1.31	48.19	1.29	41.81	1.15
ibmpg3t	17.72	18.05	493.97	18.41	413.90	1.19
ibmpg4t	22.04	30.32	675.78	31.01	229.13	2.95
ibmpg5t	12.40	16.16	657.13	16.48	649.97	1.01
ibmpg6t	17.66	23.99	965.53	34.60	915.62	1.05

group of inputs. By doing so, MEXP can simulate in much larger step size in each group, e.g. rising/falling slew and pulse width of I_1 in Figure V.4, than the step size originally in ill aligned input sources. Moreover, MEXP with grouping is parallelizable by simulating each group in multicore or cluster system.

In order to demonstrate the idea, we simply device a naive grouping scheme, which will take identical input sources together as a group. Table V.6 shows the performance of MEXP with grouping. In this table, “Groups” is the number of groups for each designs, “Max. Total” is the maximum runtime among the groups, and “Ideal Spdp” records the ideal speedup with parallelization. Design ibmpg4t has only 4 groups because the alignment is better. The performance of MEXP has

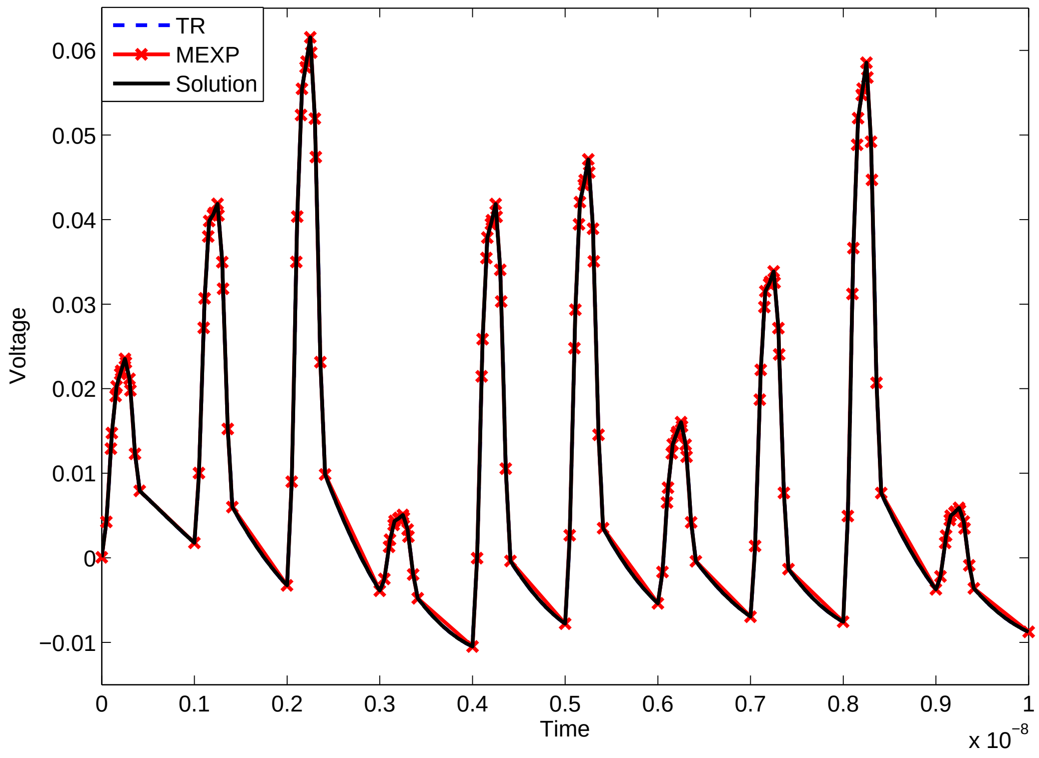


Figure V.3: Result of ibmpg6t

a significant improvement after grouping. The average ideal speedup over TRAP is 6 times. MEXP shows another parallelism in the power grid simulation from the perspective of input sources.

V.5.C Complexity

The performance of MEXP and TRAP are dominated by the number of backward/forward substitutions. This is because the time span is relatively long in a power grid, and therefore, the expensive LU fatorization is amortized during the simulation. In MEXP, the number of backward/forward substitutions at each time step is actually m, which is the number of vectors forming rational Krylov subspace. Since m is small, e.g., $5 \sim 6$ in IBM benchmark suite, and MEXP uses large step size than TRAP, MEXP has less numbers of substitution operations

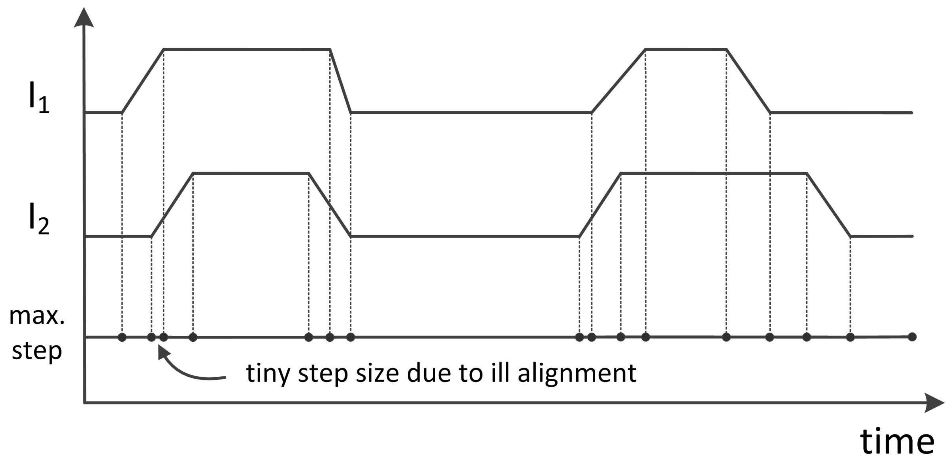


Figure V.4: Input sources alignment

Table V.6: Performance of MEXP w/ grouping

Design	DC(s)	MEXP w/ grouping ($\gamma = 10^{-10}$)			
		Groups	LU(s)	Max. Total(s)	Ideal Spdp
ibmpg1t	0.28	25	0.22	1.63	6.77
ibmpg2t	1.12	25	1.32	7.93	6.08
ibmpg3t	17.72	25	18.75	86.24	5.73
ibmpg4t	22.04	4	31.55	124.16	5.44
ibmpg5t	12.40	25	16.35	111.97	5.87
ibmpg6t	17.66	25	29.85	166.34	5.80

in general. Table V.7 presents number of backward/forward substitions for each method. "Sub." is the total number of substituion operations along the simulation. MEXP has less operations but close to TRAP because the ill alignment of input sources leads to many tiny step sizes whereas MEXP with grouping has much less operations due to well aligned inputs. These results also validate Tables V.5 and V.6 where MEXP would has less speedup due to ill alignment issue, and the grouping scheme could improve the performance.

Table V.7: Number of backward/forward substitutions

Design	TRAP	MEXP	MEXP w/ grouping
	# Sub.	# Sub.	# Max Sub.
ibmpg1t	1000	830	120
ibmpg2t	1000	830	120
ibmpg3t	1000	874	120
ibmpg4t	1000	349	143
ibmpg5t	1000	982	139
ibmpg6t	1000	932	134

V.6 Summary

We propose an adaptive time stepping method for large scale power grid simulation based on MEXP. This method utilizes rational Krylov subspace approximation, which solves stiffness constraint and eliminates the regularization requirement for singular **C**. For the time-consuming impulse response simulation for industrial PDN designs, the proposed method has more than 15 times speedup on average over the widely-adopted fixed-step trapezoidal method. Also, the proposed method shows a promising parallelism. The naive grouping scheme could further accelerate the power grid simulation about 6X on average for the IBM power grid benchmark.

Chapter V includes the content of one submitted conference paper, "Adaptive Time Stepping for Power Grid Simulation using Matrix Exponential Method," by Shih-Hung Weng, Hao Zhuang and Chung-Kuan Cheng. The paper is submitted to *IEEE International Conference on Computer-Aided Design, 2013.* The dissertation author was the primary investigator and author of the paper.

VI

Conclusion

VI.1 Summary of Contributions

In this dissertation, we study the circuit simulation and propose a new numerical integration method, the matrix exponential method. The proposed method is a general approach, which can be applied to either linear or nonlinear circuits. We utilize the Krylov subspace approximation to compute the kernel operation, the exponential of a matrix, in our approach. We discuss the adaptive step size control in both step size and dimension of the Krylov subspace to improve the performance. The stiffness constraint and the parallelism in the matrix exponential method is also investigated. Finally, we explore the rational basis in the Krylov subspace method to relax the stiffness constraint and eliminate the demanded regularization process. We demonstrate that the rational Krylov subspace approach fits well for the power grid simulation.

Chapter III presents the formulation of the matrix exponential method and the process of the Krylov subspace approximation. The detailed linear and nonlinear formulations are also presented. The proposed method solves the linear differential equation analytically via the matrix exponential operator and alleviates the stability bottleneck of explicit methods and enables great adaptivity for time step size control.

Chapter IV investigates the stiffness constraint and parallelism in the matrix exponential method. The scaling effect is observed and analyzed. With the scaling effect, the matrix exponential method is able to utilize larger step size as stepping forward. We also implemented parallel matrix exponential method on the GPGPU. The results show that the performance for highly stiff circuits is improved up to 4.8 times by exploiting the scaling effect, and is accelerated in the GPU environment up to another 11 times. Furthermore, we demonstrate that MEXP is able to handle the circuit with up to 12 million nodes.

Chapter V proposes an adaptive time stepping method for large scale power grid simulation based on the matrix exponential method. This method utilizes rational Krylov subspace approximation, which solves stiffness constraint and eliminates the regularization requirement for singular **C**. For the time-consuming impulse response simulation for industrial PDN designs, the proposed method has more than 15 times speedup on average over the widely-adopted fixed-step trapezoidal method. Also, the proposed method shows a promising parallelism. The naive grouping scheme could further accelerate the power grid simulation about 6X on average for the IBM power grid benchmark.

VI.2 Future Works

One possible direction to speedup circuit simulation is via partition. With a deliberately partition, we can simulation each sub-circuit simultaneously within few iterations to converge. Since the stiffness of a circuit comes from wide scale range in the electronic components, a well-devised partition strategy is able to separate components in different scales. Therefore, the matrix exponential method can utilize adaptive step size scheme to apply different step size to each sub-circuit. By doing so, sub-circuits with components of small magnitude are simulated in smaller step size yet short time span, and sub-circuits with components of large magnitude uses large step size but in long time span.

Another direction is to expand the matrix exponential method to different physics. In today's IC design, designers often need to consider not only the electronic behavior but also thermal effect and mechanical effect, especially for 3D IC technology. Although the thermal and mechanical effects are modeled in different equations, they can be integrated via finite element method. The matrix exponential method is possible to combine with the finite element method to simulate a circuit including its thermal and mechanical effects.

Bibliography

[1] Martin Afanasjew, Michael Eiermann, Oliver G. Ernst, and Stefan Guttel. Implementation of a restarted Krylov subspace method for the evaluation of matrix functions. *Linear Algebra and its Applications*, 429(10):2293 – 2314, 2008.

[2] A. H. Al-Mohy and N. J. Higham. Computing the action of the matrix exponential, with an application to exponential integrators. *SIAM Journal on Scientific Computing*, 33(2):488–511, 2011.

[3] 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. *Journal of Mathematical Chemistry*, 33:1–15, Jan 2003.

[4] Behnam Analui, James F Buckwalter, and Ali Hajimiri. Data-dependent jitter in serial communications. *IEEE Trans. on Microwave Theory and Techniques*, 53(11):3388–3397, 2005.

[5] S. Bächle and F. Ebert. Graph theoretical algorithms for index reduction in circuit simulation. Technical report, Inst. f. Mathematik, TU Berlin, Germany, 2005.

[6] S. Bächle and F. Ebert. Index reduction by element-replacement for electrical circuits. *Scientific Computing in Electrical Engineering*, 11:191–197, 2007.

[7] Y. Ban, T. Endo, A. Yamamoto, and Y. Yamane. Explicit time integration scheme using krylov subspace method for reactor kinetics equation. *Journal of Nuclear Science and Technology*, 48(2):243–255, 2011.

[8] M.M. Baskaran and R. Bordawekar. Optimizing sparse matrix-vector multiplication on gpus. *IBM research report RC24704, IBM*, 2009.

[9] N. Bell and M. Garland. Implementing sparse matrix-vector multiplication on throughput-oriented processors. In *SC*, page 18. ACM, 2009.

[10] G. Beylkin, J. M. Keiser, and Lev Vozovoi. A new class of time discretization schemes for the solution of nonlinear PDEs. *J. Comput. Phys.*, 147:362–387, Dec 1998.

[11] Srinivas Bodapati and Farid N Najm. High-level current macro-model for power-grid analysis. In *Proc. Design Automation Conference*, pages 385–390, 2002.

[12] B. N. Bond. *SMORES: A Matlab tool for Simulation and Model Order Reduction of Electrical Systems*, 2010. http://bnbond.com/software/smores/.

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

[14] Q. Chen, S. H. Weng, and C. K. Cheng. A practical regularization technique for modified nodal analysis in large-scale time-domain circuit simulation. submitted to *IEEE Trans. on CAD*, 2011.

[15] Tsung-Hao Chen and Charlie Chung-Ping Chen. Efficient large-scale power grid analysis based on preconditioned krylov-subspace iterative methods. In *Proc. Design Automation Conference*, pages 559–562, 2001.

[16] Yuhua Cheng and Chenming Hu. *MOSFET modeling and BSIM3 user's guide.* Springer, 1999.

[17] L. O. Chua and P. M. Lin. *Computer-Aided Analysis of Electronic Circuits*, chapter 8. Prentice-Hall, New Jersey, 1975.

[18] Leon O. Chua and Pen-Min Lin. *Computer Aided Analysis of Electric Circuits: Algorithms and Computational Techniques.* Prentice-Hall, 1975.

[19] Timothy A. Davis. *Direct Method for Sparse Linear Systems.* SIAM, 2006.

[20] A. Devgan and R.A. Rohrer. Adaptively controlled explicit simulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(6):746–762, 1994.

[21] Anirudh Devgan and Ronald A. Rohrer. Event driven adaptively controlled explicit simulation of integrated circuits. In *ICCAD*, pages 136–140, 1993.

[22] W. Dong and P. Li. Parallel circuit simulation with adaptively controlled projective integration. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 16(4):44, 2011.

[23] Wei Dong and Peng Li. Parallelizable stable explicit numerical integration for efficient circuit simulation. In *DAC*, pages 382–385, 2009.

[24] Vladimir Druskin, Leonid Knizhnerman, and Mikhail Zaslavsky. Solution of large scale evolutionary problems using rational krylov subspaces with optimized shifts. *SIAM Journal on Scientific Computing*, 31(5):3760–3780, 2009.

[25] Peng Du, Xiang Hu, Shih-Hung Weng, Amirali Shayan, Xiaoming Chen, A Ege Engin, and Chung Kuan Cheng. Worst-case noise prediction with non-zero current transition times for early power distribution system verification. In *Intl. Symposium on Quality Electronic Design*, pages 624–631, 2010.

[26] Michael Eiermann and Oliver G. Ernst. A restarted Krylov subspace method for the evaluation of matrix functions. *SIAM J. NUMER. ANAL*, 44:2481–2504, 2006.

[27] Michael Eiermann, Oliver G. Ernst, and Stefan Guttel. Deflated restarting for matrix functions. *SIAM J. MATRIX ANAL. APPL.*, 32(2):621–641, 2011.

[28] Zhuo Feng and Peng Li. Multigrid on gpu: tackling power grid analysis on parallel simt platforms. In *Proc. Intl. Conf. Computer-Aided Design*, pages 647–654, 2008.

[29] Imad A Ferzli, Eli Chiprout, and Farid N Najm. Verification and codesign of the package and die power delivery system using wavelets. *IEEE Trans. Computer-Aided Design*, 29(1):92–102, 2010.

[30] N. Frohlich, B.M. Riess, U.A. Wever, and Q. Zheng. A new approach for parallel simulation of vlsi circuits on a transistor level. *Circuits and Systems I: Fundamental Theory and Applications, IEEE Transactions on*, 45(6):601–613, 1998.

[31] CW Gear and I.G. Kevrekidis. Telescopic projective methods for parabolic differential equations. *Journal of Computational Physics*, 187(1):95–109, 2003.

[32] Gennady Gildenblat, Xin Li, Weimin Wu, Hailing Wang, Amit Jha, Ronald van Langevelde, Geert DJ Smit, Andries J Scholten, and Dirk BM Klaassen. Psp: An advanced surface-potential-based mosfet model for circuit simulation. *Electron Devices, IEEE Transactions on*, 53(9):1979–1993, 2006.

[33] Kanupriya Gulati, John F Croix, Sunil P Khatr, and Rahm Shastry. Fast circuit simulation on graphics processing units. In *Proceedings of the 2009 Asia and South Pacific Design Automation Conference*, pages 403–408. IEEE Press, 2009.

[34] M. Güther and U. Feldmann. The DAE-index in electric circuit simulation. *Mathematics and Computers in Simulation*, 39(5-6):573–582, Nov 1995.

[35] Ernst Hairer and Gerhard Wanner. *Solving Ordinary Differential Equations II: Stiff and Differential-Algebraic Problems*. Springer, 1996.

[36] N.J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1196, 2005.

[37] C.W. Ho, A. Ruehli, and P. Brennan. The modified nodal approach to network analysis. *IEEE Trans. on CAS*, 22(6):504–509, 1975.

[38] M. Hochbruck and C. Lubich. On Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 34(5):1911–1925, 1997.

[39] S. Hutchinson, E. Keiter, R.J. Hoekstra, H.A. Watts, A.J. Waters, R.L. Schells, and S.D. Wix. The xyce parallel electronic simulator-an overview. In *IEEE International Symposium on Circuits and Systems, Sydney (AU)*, 2000.

[40] D. Joyner, M. V. Nguyen, and N. Cohen. *graph-theory-algorithms-book*. Online resource. http://code.google.com/p/graph-theory-algorithms-book/.

[41] Nachiket Kapre and André DeHon. Performance comparison of single-precision spice model-evaluation on fpga, gpu, cell, and multi-core processors. In *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*, pages 65–72. IEEE, 2009.

[42] Joseph N Kozhaya, Sani R Nassif, and Farid N Najm. A multigrid-like technique for power grid analysis. *IEEE Trans. Computer-Aided Design*, 21(10):1148–1160, 2002.

[43] Jongwon Lee, Venkataramanan Balakrishnan, Cheng-Kok Koh, and Dan Jiao. A linear-time complex-valued eigenvalue solver for full-wave analysis of large-scale on-chip interconnect structures. *IEEE Trans. Microwave Theory and Techniques*, 57(8):2021–2029, 2009.

[44] E. Lelarasmee, A.E. Ruehli, and A.L. Sangiovanni-Vincentelli. The waveform relaxation method for time-domain analysis of large scale integrated circuits. *IEEE Trans. on CAD*, 1(3):131–145, 1982.

[45] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, Mar 2003.

[46] Igor Moret and Paolo Novati. Rd-rational approximations of the matrix exponential. *BIT Numerical Mathematics*, 44(3):595–615, 2004.

[47] Sani R Nassif. Power grid analysis benchmarks. In *Proc. Asia and South Pacific Design Automation Conference*, pages 376–381, 2008.

[48] Sani R Nassif and Joseph N Kozhaya. Fast power grid simulation. In *Proc. Design Automation Conference*, pages 156–161, 2000.

[49] S. Natarajan. A systematic method for obtaining state equations using MNA. *IEE Proceedings-G of Circuits, Devices and Systems*, 138(3):131–162, Jun 1991.

[50] R. Nath, S. Tomov, and J. Dongarra. Accelerating gpu kernels for dense linear algebra. *High Performance Computing for Computational Science–VECPAR 2010*, pages 83–92, 2011.

[51] Qing Nie, Yong-Tao Zhang, and Rui Zhao. Efficient semi-implicit schemes for stiff systems. *Journal of Computational Physics*, 214:537–541, 2006.

[52] J. Niesen and W. M. Wright. A Krylov subspace algorithm for evaluating the φ-functions appearing in exponential integrators. *ACM Trans. Math. Software.* in press.

[53] James M Ortega and Werner C Rheinboldt. *Iterative solution of nonlinear equations in several variables*, volume 30. Society for Industrial and Applied Mathematics, 1987.

[54] H. Peng and C.K. Cheng. Parallel transistor level full-chip circuit simulation. In *DATE*, pages 304–307. IEEE, 2009.

[55] B.M. Riess, K. Doll, and F.M. Johannes. Partitioning very large circuits using analytical placement techniques. In *Design Automation, 1994. 31st Conference on*, pages 646–651. IEEE, 1994.

[56] Y. Saad. Analysis of some Krylov subspace approximations to the matrix exponential operator. *SIAM Journal on Numerical Analysis*, 1992.

[57] Yousef Saad. *Iteravite Methods for Sparse Linear Systems.* SIAM, 2003.

[58] P Sadayappan and V Visvanathan. Circuit simulation on shared-memory multiprocessors. *Computers, IEEE Transactions on*, 37(12):1634–1642, 1988.

[59] Resve A Saleh, KYLE A Gallivan, M-C Chang, IN Hajj, D Smart, and TN Trick. Parallel circuit simulation on supercomputers. *Proceedings of the IEEE*, 77(12):1915–1931, 1989.

[60] José E. Schutt-Ainé. Latency insertion method (LIM) for the fst transient simulation of large networks. *IEEE Trans. on CAS I*, 48(1):81–89, Jan 2001.

[61] D. Estevez Schwarz and C. Tischendorf. Structural analysis for electric circuits and consequences for MNA. *INT. J. CIRC. THEOR. APPL*, 28:131–162, 1998.

[62] Rui Shi, Wenjian Yu, Yi Zhu, Chung Kuan Cheng, and Ernest S Kuh. Efficient and accurate eye diagram prediction for high speed signaling. In *Proc. Intl. Conf. Computer-Aided Design*, pages 655–661, 2008.

[63] Gerhard Starke and Richard S Varga. A hybrid arnoldi-faber iterative method for nonsymmetric systems of linear equations. *Numerische Mathematik*, 64(1):213–240, 1993.

[64] Haihua Su, Emrah Acar, and Sani R Nassif. Power grid reduction based on algebraic multigrid principles. In *Proc. Design Automation Conference*, pages 109–112, 2003.

[65] K. Sun, Q. Zhou, K. Mohanram, and D.C. Sorensen. Parallel domain decomposition for simulation of large-scale power grids. In *Computer-Aided Design, 2007. ICCAD 2007. IEEE/ACM International Conference on*, pages 54–59. IEEE, 2007.

[66] Synopsys. *HSPICE Simulation and Analysis User Guide*, y-2006.03 edition, Mar 2006.

[67] Wei Tian, Xie-Ting Ling, and Ruey-Wen Liu. Novel methods for circuit worst-case tolerance analysis. *IEEE Trans. on Circuits and Systems I: Fundamental Theory and Applications*, 43(4):272–278, 1996.

[68] Jasper van den Eshof and Marlis Hochbruck. Preconditioning lanczos approximations to the matrix exponential. *SIAM Journal on Scientific Computing*, 27(4):1438–1457, 2006.

[69] Donald M Webber and A Sangiovanni-Vincentelli. Circuit simulation on the connection machine. In *Proceedings of the 24th ACM/IEEE Design Automation Conference*, pages 108–113. ACM, 1987.

[70] S.-H. Weng, Quan Chen, and C.-K. Cheng. Circuit simulation by matrix exponential method. In *2011 IEEE ASIC Conference*, pages 462–465. IEEE, 2011.

[71] 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 *2011 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1467–1470. IEEE, 2011.

[72] James Hardy Wilkinson. Kronecker's canonical form and the qz algorithm. *Linear Algebra and its Applications*, 28:285–303, 1979.

[73] Xuanxing Xiong and Jia Wang. Parallel forward and back substitution for efficient power grid simulation. In *Proc. Intl. Conf. Computer-Aided Design*, pages 660–663, 2012.

[74] G-C Yang. Paraspice: a parallel circuit simulator for shared-memory multiprocessors. In *Design Automation Conference, 1990. Proceedings., 27th ACM/IEEE*, pages 400–405. IEEE, 1990.

[75] Jianlei Yang, Zuowei Li, Yici Cai, and Qiang Zhou. Powerrush: Efficient transient simulation for power grid analysis. In *Proc. Intl. Conf. Computer-Aided Design*, pages 653–659, 2012.

[76] Xiaoji Ye, Wei Dong, Peng Li, and Sani Nassif. Maps: Multi-algorithm parallel circuit simulation. In *ICCAD*, pages 73–78, 2008.

[77] Xiaoji Ye, Wei Dong, Peng Li, and Sani Nassif. Hierarchical multialgorithm parallel circuit simulation. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 30(1):45–58, 2011.

[78] Ting Yu, Martin Wong, et al. Pgt_solver: an efficient solver for power grid transient analysis. In *Proc. Intl. Conf. Computer-Aided Design*, pages 647–652, 2012.

[79] Min Zhao, Rajendran V Panda, Sachin S Sapatnekar, and David Blaauw. Hierarchical analysis of power distribution networks. *IEEE Trans. Computer-Aided Design*, 21(2):159–168, 2002.