Vectorized Efficient Computation of Padé Approximation for Semi-Analytical Simulation of Large-Scale Power Systems

Rui Yao, *Member, IEEE,* Kai Sun, *Senior Member, IEEE,* and Feng Qiu, *Senior Member, IEEE*

***Abstract*—Semi-analytical simulation (SAS) is a methodology that derives power series as an approximate solution in power system steady-state or dynamic analysis. Padé approximation is able to further improve the efficiency of SAS, while its computation for large-scale power systems is time-consuming. This paper proposes a vectorized fast algorithm for computing Padé approximation of a large-scale SAS. The Levinson algorithm is used to reduce the temporal and spatial complexities. Considering the structural homogeneity of computation, a vectorized version of Levinson algorithm is realized to achieve instruction-level parallelism. The novel approach is tested on the SAS of Polish 2383-bus system, which verifies the advantage of Levinson algorithm and vectorization in boosting the computation speed of SAS.**

***Index Terms*—Parallel computation, vectorization, power series, Padé, holomorphic embedding, power system, simulation**

I. Introduction

MODERN computers provide parallelism for accelerating computation. For large-scale power systems, there have been various parallel methods for accelerating the power flow or dynamic simulation [1]–[3]. These methods focus on accelerating computation by assigning the computation tasks to multiple processors. But the overall performance depends on both the number of the processors and the efficiency on each processor. Therefore, besides the multi-tasking on multiple processors, the exploitation of each processor is also important. One type of parallelism inside processors is vectorization, which is based on single instruction, multiple data (SIMD) feature. It is useful for accelerating large-scale homogeneous, but simple operations.

Semi-analytical simulation (SAS) is a novel methodology for power system analysis, which derives approximate solutions as high-order power series, and thus achieves longer time step and accelerates computation. Existing SAS methods mainly include holomorphic embedding (HE) [4], Adomian decomposition method (ADM) [5], Taylor expansion method (TEM) [6], and differential transformation method (DTM) [7]. Moreover, the Padé approximation (PA) is usually derived from power series to further expand the effective range of SAS [8], [9]. In SAS, the derivation of PAs is a performance bottleneck. To accelerate the SAS, an efficient method for calculating PAs based on Levinson algorithm is proposed, which improves both the temporal and spatial complexities as compared with LU factorization. Moreover, the vectorized version of Levinson algorithm significantly accelerates the computation. Such an approach can be generically utilized in any large-scale homogeneous computation of PAs.

This work was supported in part by the ERC Program of the NSF and DOE under NSF grant EEC-1041877, in part by NSF grant ECCS-1610025, and in part by DOE grant DE-OE0000875. (*Corresponding author: Kai Sun*)

R. Yao is with Argonne National Laboratory, Lemont, IL 60439 and the Department of EECS, University of Tennessee, Knoxville, TN 37996, USA (emails: yaorui.thu@gmail.com, ryao@anl.gov).

K. Sun is with the Department of EECS, the University of Tennessee, Knoxville, TN 37996, USA (email: kaisun@utk.edu).

F. Qiu is with Argonne National Laboratory, Lemont, IL 60439 (email: fqiu@anl.gov).

II. Padé approximants from power series in SAS

The power system analysis solves the following equations:

$$\mathbf{f}(\dot{\mathbf{x}}(\alpha), \mathbf{x}(\alpha), \mathbf{p}(\alpha)) = 0, \tag{1}$$

where $\mathbf{x}$ stands for state variables, and $\mathbf{p}$ is system parameters. α is a parameter representing time or an abstract path of system state transition. In steady-state analysis, (1) degenerates and $\dot{\mathbf{x}}(\alpha)$ does not appear. The SAS first generates approximate solutions as a truncated power series of α:

$$x_{PS}(\alpha, N) = c_0 + c_1\alpha + \cdots + c_N\alpha^N, \tag{2}$$

and the corresponding PA has the following form:

$$x_{PA}(\alpha, m, n) = \frac{a_0 + a_1\alpha + \cdots + a_m\alpha^m}{b_0 + b_1\alpha + \cdots + b_n\alpha^n}, \tag{3}$$

where $N = m + n$, and to ensure uniqueness of the coefficients, $b_0 = 1$ is enforced. To solve the coefficients in (3), let $x_{PS}(\alpha, N) = x_{PA}(\alpha, m, n)$, and according to (2) and (3), multiply the left by the denominator of the right-hand side:

$$(c_0 + c_1\alpha + \cdots + c_N\alpha^N)(b_0 + b_1\alpha + \cdots + b_n\alpha^n), \tag{4}$$

and expand the numerator of (3) as

$$a_0 + a_1\alpha + \cdots + a_m\alpha^m + a_{m+1}\alpha^{m+1} + \cdots + a_N\alpha^N, \tag{5}$$

where $a_{m+1} = \cdots = a_N = 0$. By equating the coefficients in (4) and (5), the following equations can be obtained:

$$\left[\begin{array}{c} a_0 \\ \vdots \\ a_m \\ \hline 0 \\ \vdots \\ 0 \end{array}\right] = \left[\begin{array}{c} c_0 \\ \vdots \\ c_m \\ \hline c_{m+1} \\ \vdots \\ c_N \end{array}\right] + \left[\begin{array}{cccc} 0 & 0 & \cdots & 0 \\ c_0 & 0 & \cdots & 0 \\ c_1 & c_0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ c_{m-1} & c_{m-2} & \cdots & \cdots \\ \hline c_m & c_{m-1} & \cdots & \cdots \\ \vdots & \vdots & \ddots & \vdots \\ c_{N-2} & c_{N-3} & \cdots & c_{N-n-1} \\ c_{N-1} & c_{N-2} & \cdots & c_{N-n} \end{array}\right] \left[\begin{array}{c} b_1 \\ \vdots \\ b_n \end{array}\right], \tag{6}$$

or the following compact form:

$$\left[\begin{array}{c} \mathbf{a} \\ \hline \mathbf{0} \end{array}\right] = \left[\begin{array}{c} \mathbf{c_1} \\ \hline \mathbf{c_2} \end{array}\right] + \left[\begin{array}{c} \mathbf{T_{c1}} \\ \hline \mathbf{T_{c2}} \end{array}\right] \mathbf{b}. \tag{7}$$

The solution of (7) can be obtained:

$$\mathbf{b} = -\mathbf{T_{c2}}^{-1}\mathbf{c_2} \tag{8a}$$

$$\mathbf{a} = \mathbf{c_1} - \mathbf{T_{c1}}\mathbf{T_{c2}}^{-1}\mathbf{c_2}. \tag{8b}$$

To achieve best approximation, $|m - n|$ should be as small as possible [9]. Eq. (8a) solves a linear equation with size of about $N/2$. For commonly used CPUs, a floating-point multiplication/addition operation can be finished in 1 CPU cycle, and division needs k_D cycles (for the 2017 Intel® Skylake architecture, $k_D = 4$ or 5 [10]). The step (8a) needs LU-decomposition, forward and backward substitutions of triangular matrices. For $n = N/2$, the LU decomposition is estimated to use $N^3/12 - N^2/8 - N/12 + k_D N/2$ cycles, and the forward and backward substitutions in total take $N^2/2 + N/2$ cycles. Step (8b) has a matrix-vector multiplication and an addition of two vectors, which costs $N^2/4 - N/2$ cycles. Take $k_D = 4$, the estimated total CPU cycles for solving (8) by using LU-decomposition (named as LU method in this paper) is approximately

$$n_{op}^{LU} = \frac{N^3}{12} + \frac{5N^2}{8} + \frac{23N}{12}. \tag{9}$$

III. Enhance efficiency with Levinson Algorithm

A. Levinson algorithm

In (8), solving the linear equations is the most time-consuming part. Generally, solving the linear equations with size $n = N/2$ has temporal complexity of $O(N^3)$ and spatial complexity of $O(N^2)$. However, noticing that $\mathbf{T_{c2}}$ is a Toeplitz matrix, the computational burden can be reduced by making use of its special structure. The Levinson Algorithm solves Toeplitz systems in $O(N^2)$ time and $O(N)$ space.

Toeplitz matrix $\mathbf{T_{c2}}$ can be described with an vector $\mathbf{t_{c2}}$:

$$\mathbf{t_{c2}} = [c_{N-2n+1}, c_{N-2n+2}, \cdots, c_m, \cdots, c_{N-1}]^{\mathrm{T}}, \tag{10}$$

where $c_k = 0$ for all $k < 0$. The Levinson Algorithm is described in Algorithm 1.

Algorithm 1. Solving $\mathbf{T_{c2}}^{-1}\mathbf{c_2}$ with Levinson Algorithm

Input: Toeplitz matrix elements $\mathbf{t_{c2}}$ ($(2n-1) \times 1$), vector $\mathbf{c_2}$ ($n \times 1$).
Variables: $n \times 1$ vectors $\mathbf{u}$, $\mathbf{v}$, $\mathbf{w}$. Scalars ε_u, ε_v, ε_x, α_u, α_v, β_u, β_v.
Output: $n \times 1$ vector $\mathbf{x}$ ($\mathbf{x}[i]$ is the ith element in $\mathbf{x}$, and $\mathbf{x}[i:j]$ is the block from the ith to jth elements).

```
u[1] = 1/t_c2[n], v[n] = u[1], x[1] = c_2[1] * u[1].
for $k = 2 \to n$ do
  $\varepsilon_u \leftarrow 0$, $\varepsilon_v \leftarrow 0$, $\varepsilon_x \leftarrow 0$.
  for $i = 1 \to k-1$ do
    $\varepsilon_u \leftarrow \varepsilon_u + \mathbf{u}[i] * \mathbf{t_{c2}}[n+k-i]$.
    $\varepsilon_v \leftarrow \varepsilon_v + \mathbf{v}[n-i+1] * \mathbf{t_{c2}}[n-k-i+2]$.
    $\varepsilon_x \leftarrow \varepsilon_x + \mathbf{x}[i] * \mathbf{t_{c2}}[n+k-i]$.
  end for
  $\alpha_u \leftarrow \frac{1}{1-\varepsilon_u * \varepsilon_v}$, $\alpha_v \leftarrow -\varepsilon_u * \alpha_u$, $\beta_u \leftarrow -\varepsilon_v * \alpha_u$, $\beta_v \leftarrow \alpha_u$.
  $\mathbf{w}[1:k-1] \leftarrow \mathbf{u}[1:k-1]$.
  for $i = 1 \to k$ do
    $\mathbf{u}[i] \leftarrow \alpha_u * \mathbf{w}[i] + \beta_u * \mathbf{v}[N-k+i]$.
    $\mathbf{v}[N-k+i] \leftarrow \alpha_v * \mathbf{w}[i] + \beta_v * \mathbf{v}[N-k+i]$.
    $\mathbf{x}[i] \leftarrow \mathbf{x}[i] + (\mathbf{c_2}[k] - \varepsilon_x) * \mathbf{v}[N-k+i]$.
  end for
end for
```

By counting the operations, the CPU time needed for computing (8) with Levinson algorithm (named as Levinson method in this paper) is approximately $5n^2 + (k_D + 8)n - k_D - 13 + 2n^2$ cycles, and with $k_D = 4$ and $n = N/2$, it is

$$n_{op}^{Lev} = \frac{7N^2}{4} + 6N - 17. \tag{11}$$

Eqs. (9) and (11) indicate that the Levinson method has lower order of complexity as compared with the LU method. Fig. 1 compares the CPU times of the two methods with N set as 5 to 30. Levinson method is estimated to cost more CPU time than LU method when $N < 16$, and it costs less CPU time when $N \geq 16$.

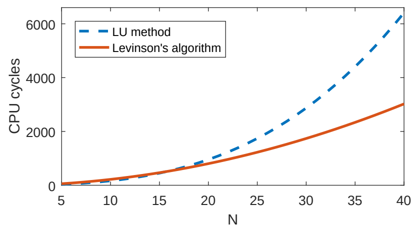

Fig. 1. Compare CPU cycles of LU and Levinson methods.

B. Vectorization of Levinson method

When N is not big, the Levinson method on each variable does not significantly improve the performance. However, note the homogeneity in the SAS: for a given N, the approximate solutions (2) and (3) of all the variables are in the same form. The procedures of calculating PAs are also identical for all variables. Such a homogeneity enables vectorization of PAs. The homogeneity is important for vectorization: there are superfast algorithms for solving Toeplitz matrix in $O(N \log N)$ time e.g. the algorithm in [11], but it uses approximation and has inhomogeneous loops controlled by error thresholds, which is hard to be vectorized across variables. Moreover, as N is not big in SAS, the superfast methods are not significantly faster than Levinson method and thus are not necessary.

There are various platforms supporting vectorization, such as C/C++ compilers that support SIMD instructions, and vectorization-friendly language Matlab/Octave, etc. Specifically, the Algorithm 1 can be easily modified as a vectorized version by vectorizing the atomic operations. The vectorization and optimization of the code are language-specific. For example, the inner loops in Algorithm 1 can be replaced by the built-in function `sum()` in Matlab, while in C/C++, the inner loops can be vectorized by the compiler.

IV. Test case

The proposed method for deriving PAs is tested on voltage security analysis with HE [4]. The test is on the Polish system model with 2383 buses (including 326 PV buses and 2056 PQ buses) and 1562 induction motors. The simulation is developed on Matlab™ and is tested on a computer with Intel® Core™ i7-6600U CPU (Skylake architecture, 4 logical processors) and 8GB DDR4 2133MHz RAM. Assume that the load increase on each bus is 50% of its base state value per second, and simulation is run until voltage collapse occurs. With HE, the approximate solution in the form of power series is first derived, and then the Padé approximants are computed. In this case, there are 6652 variables in total.

Four methods for deriving PAs are compared: the LU method on a single variable, the Levinson method on a single variable, the vectorized LU method, and the vectorized version of Levinson method. These methods are implemented in Matlab codes, respectively, and the LU methods do not use pivoting. The performances of the four methods with $N = 12$ and $N = 40$ are listed in Table I. Although the Levinson method on a single variable is faster than the LU method, the computation of PAs is still the bottleneck of SAS. For both LU and Levinson methods, the vectorization significantly accelerates computation, and the advantage of Levinson method becomes more significant with a larger N. The vectorized LU and Levinson methods are compared with even more N values. As Fig. 2 shows, the average time consumption per HE step of LU method is higher than that of Levinson method, and the difference becomes more significant as N increases, which generally matches the trends shown in Fig. 1. However, the complexity analysis cannot comprehensively consider all the factors that may influence the performance. For example, the computational overhead (e.g. time for managing memory) is an important factor. Since the LU method has higher spatial complexity than the Levinson method, we may infer that it also has higher overhead. As N increases, the overhead of LU method may also increase more significantly than that of the Levinson method. This indicates that the lower spatial complexity also helps accelerate computation in SAS.

TABLE I
COMPUTATION TIME OF POLISH SYSTEM CASE

Method	Time for Padé (s)		Total time (s)	
	$N = 12$	$N = 40$	$N = 12$	$N = 40$
LU-Single	32.42	100.84	45.15	130.90
Levinson-Single	14.90	49.42	30.09	82.62
LU-Vectorized	1.27	15.13	16.93	51.87
Levinson-Vectorized	0.26	2.34	15.91	39.74

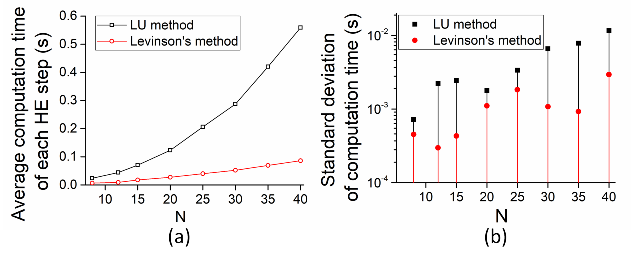

Fig. 2. Comparison of vectorized LU and Levinson methods under different N values. Each setting is run 10 times. (a) average computation time per HE step. (b) standard deviation of computation time per HE step. The low deviations in (b) indicate that the average computation time in (a) can well reflect the actual performance of the methods.

For compiled languages like C/C++, it is also recommended to organize the codes for vectorization. The vectorization has more stable performance than task-level parallelism because it is not affected by the task scheduling among processors. And as the vectorization is taken care of by the compilers, it does not need extra efforts to design multi-processor tasking in codes. It gives insights that besides concurrent tasking on multiple processors, the potentials inside each processor can also be conveniently exploited to achieve faster computation.

It should be noted that many factors may influence the actual performance of a computer program, such as programming language, algorithm and code efficiency, CPU instruction sets and caches, memory access, background processes, etc. And those factors should be carefully considered in the performance analysis. Besides, a more general insight can also be derived: for the enhancement of a computational task, both theoretical algorithm improvement (e.g. using Levinson algorithm) and appropriate engineering measures (e.g. code vectorization) should be adopted.

V. CONCLUSION

Since the calculation of Padé approximation (PA) is a performance bottleneck in the steady-state or dynamic analysis based on semi-analytical simulation (SAS) in power systems, a new method of calculating PAs based on Levinson algorithm is proposed. The Levinson algorithm effectively reduces both temporal and spatial complexity as compared with the conventional LU decomposition method. Moreover, by making use of the homogeneity in the structure of computing PAs of large number of variables, a vectorized method for calculating PAs is proposed in this paper. The new approach is tested with the voltage security analysis based on SAS in a Polish 2383-bus system. The results demonstrate that the proposed approach significantly reduces the computation time of PAs, and thus also substantially accelerate the SAS computation. The results verify the advantage of Levinson algorithm in both spatial and temporal complexities, and also show the importance of utilizing the instruction-level parallelism of CPUs.

REFERENCES

[1] P. Aristidou, D. Fabozzi, and T. Van Cutsem, "Dynamic simulation of large-scale power systems using a parallel schur-complement-based decomposition method," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 10, pp. 2561–2570, 2014.

[2] Y. Liu and Q. Jiang, "Two-stage parallel waveform relaxation method for large-scale power system transient stability simulation," *IEEE Trans. Power Syst.*, vol. 31, no. 1, pp. 153–162, 2016.

[3] M. A. Tomim, J. R. Martí, and J. A. Passos Filho, "Parallel transient stability simulation based on multi-area thévenin equivalents," *IEEE Trans. Smart Grid*, vol. 8, no. 3, pp. 1366–1377, 2017.

[4] R. Yao and K. Sun, "Voltage stability analysis of power systems with induction motors based on holomorphic embedding," *IEEE Trans. Power Syst.*, vol. 34, no. 2, pp. 1278–1288, 2019.

[5] N. Duan and K. Sun, "Power system simulation using the multistage adomian decomposition method," *IEEE Trans. Power Syst.*, vol. 32, no. 1, pp. 430–441, 2017.

[6] B. Wang, N. Duan, and K. Sun, "A time-power series based semi-analytical approach for power system simulation," *IEEE Trans. Power Syst.*, vol. 34, no. 2, pp. 841–851, 2019.

[7] I. A.-H. Hassan, "Application to differential transformation method for solving systems of differential equations," *Applied Mathematical Modelling*, vol. 32, no. 12, pp. 2552–2559, 2008.

[8] C. Liu, B. Wang, and K. Sun, "Fast power system simulation using semi-analytical solutions based on pade approximants," in *Power & Energy Society General Meeting*. IEEE, 2017, pp. 1–5.

[9] S. Rao, Y. Feng, D. J. Tylavsky, and M. K. Subramanian, "The holomorphic embedding method applied to the power-flow problem," *IEEE Trans. Power Syst.*, vol. 31, no. 5, pp. 3816–3828, 2016.

[10] A. Fog. (2018) Instruction tables: Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs. [Online]. Available: www.agner.org/optimize

[11] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, "A superfast algorithm for toeplitz systems of linear equations," *SIAM Journal on Matrix Analysis and Applications*, vol. 29, no. 4, pp. 1247–1266, 2007.