STEPS: A Portable Dynamic Simulation Toolkit for Electrical Power System Studies

Changgang Li, *Member, IEEE*, Yue Wu, Hengxu Zhang, *Member, IEEE*, Hua Ye, *Member, IEEE*, Yutian Liu, *Senior Member, IEEE*, and Yilu Liu, *Fellow, IEEE*

***Abstract*—Numerical simulation is the key technique for large-scale power system analysis. Redistribution of global renewable power via international interconnections requires new simulation tools to study the interconnected systems with different nominal frequencies as a whole. This paper introduces an open-source simulation toolkit for electrical power systems (STEPS) hosted at Github. Its kernel is coded in C++ with major functions of power flow and electro-mechanical dynamic simulation. Flexible options are provided and configurable to improve power flow solution and dynamic simulation. Common devices and models are supported in STEPS for AC/DC hybrid system studies. The study of interconnected systems with different nominal frequencies is supported in STEPS for research of international interconnection. Application program interfaces are provided and wrapped with Python to enable high-level interfaces for general applications. STEPS is thread-safe, and parallel computation is supported in both kernel-level and task-level parallelization to accelerate simulation. It is portable and works on Windows and GNU/Linux platforms. Cases from small to large-scale systems are thoroughly tested to validate the toolkit with commercial packages as benchmarks.**

***Index Terms*—Power system, open-source, simulation package, power flow, electro-mechanical dynamic.**

I. Introduction

POWER system is the most sophisticated and complex artificial system supporting the operation of modern society. It is critical to keep the power system operating in a secure, stable, and economical mode [1]. Small power grids have been interconnected to form large-scale power grids via high voltage AC tie lines or DC links to achieve better efficiency [2]. The interconnection is further enhanced to redistribute clean but uneven renewable power such as wind and solar. The uncertainty of increasing renewable generation gives rise to more complex operation of modern power systems [3]. The stability margin of power systems is significantly depressed, and many blackouts occur in the last two decades [4], [5]. It is essential to study the dynamics of the modern power system and improve its stability with sophisticated techniques.

Currently, electro-mechanical dynamic simulation is the major technique for studying power system stability. There are many commercial packages available for power system electro-mechanical studies, such as PSS/E, PSASP, BPA, DigSILENT/Power Factory, Eurostag, and PSLF. Those packages are sophisticated and well tested for industry applications. The advantage of those packages lies in reliability, function maturity, and model completeness. However, their pricy license and maintenance prevent them from broader usage. Besides, the protection of their intellectual property also prevents public availability of their source codes. The development of new functions for a specific application and bug fix is usually time-consuming.

Aside from commercial software, some free packages have been developed, e.g., PST [6], PSAT [7], MATPOWER [8], MatDyn [9], PYPOWER [10], pypower-dynamics [11], PandaPower [12], PyPSA [13], PSST [14], and interPSS [15]. The PST, PSAT, MATPOWER, and MatDyn depend on commercial software Matlab. PYPOWER is a port of MATPOWER to Python with functions of power flow and optimal power flow (OPF). pypower-dynamics extends PYPOWER with dynamic simulation capability but provided very limited dynamic models. PandaPower is developed based on PYPOWER and focuses on static analysis. PyPSA provides power flow, OPF, security-constrained optimal power flow (SCOPF), and investment optimization. PSST is hosted on Github and focuses on Unit Commitment and Economic Dispatch. InterPSS is built with Java and XML to realize the online power flow and short circuit calculation. As a special case, the DOME project, which works only on the Fedora GNU/Linux platform, is a commercial package but provides a free version with a limited size up to 14 buses [16].

For large-scale power system studies, an easy-to-use simulation package should provide abundant models with robust expandability. It should also be portable to different platforms with high computation efficiency. Referring to the available open-source packages, we have started the STEPS project to build an open-source simulation package dating back to 2010. STEPS, the acronym of Simulation Toolkit for

Manuscript received May 12, 2020; revised November 17, 2020; accepted December 5, 2020. Date of publication December 15, 2020; date of current version June 18, 2021. This work was supported in part by National Key R&D Program of China under Grant 2017YFB0902600, in part by State Grid Corporation of China Project under Grant SGJS0000DKJS1700840, and in part by the Young Scholars Program of Shandong University under Grant 2018WLJH31. Paper no. TPWRS-00778-2020. *(Corresponding author: Yutian Liu.)*

Changgang Li, Yue Wu, Hengxu Zhang, Hua Ye, and Yutian Liu are with the Key Laboratory of Power System Intelligent Dispatch and Control of the Ministry of Education (Shandong University), Jinan 250061, China (e-mail: lichgang@sdu.edu.cn; sduyuewu2018@163.com; zhanghx@sdu.edu.cn; yehua@email.sdu.edu.cn; liuyt@sdu.edu.cn).

Yilu Liu is with the School of Electrical Engineering and Computer Science, the University of Tennessee, Knoxville, TN 37996-2250 USA (e-mail: liu@utk.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TPWRS.2020.3045102.

Digital Object Identifier 10.1109/TPWRS.2020.3045102

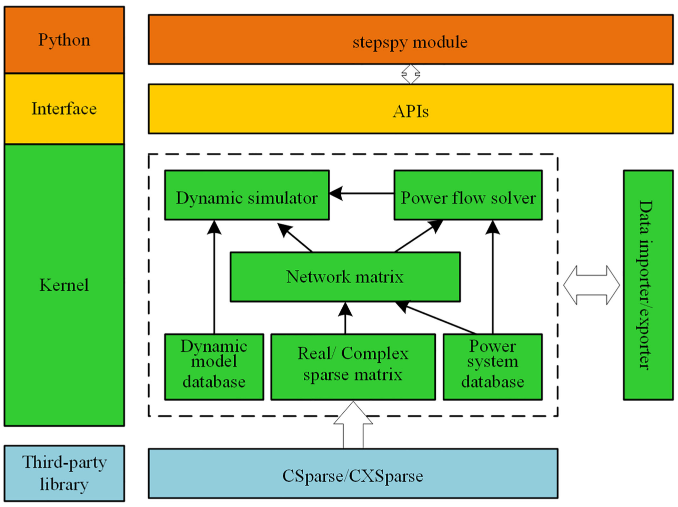


Fig. 1. Structure of STEPS.

Electrical Power Systems, is open-source and now hosted at https://github.com/changgang/steps under MIT license.

STEPS is focusing on power flow and dynamic simulation. Its major features can be summarized as follows. 1) Abundant models are supported for studying systems with conventional synchronous generators, renewable generation, and HVDC without capacity limitation. Joint simulation of interconnected systems with different nominal frequencies is supported to promote research of international interconnection. 2) STEPS kernel is independent and coded with C++ to achieve high computation efficiency. A Python module named stepspy is developed to wrap the application program interfaces (API) for making STEPS easy to use. The stepspy module can be installed in Python from https://pypi.org/project/stepspy with standard pip utility. 3) STEPS is open-source and portable to different platforms. It has been tested to work in Windows and Linux operating systems on X64 and Loongson platforms. 4) STEPS is thread-safe with parallel computation capability based on OpenMP. Both kernel-level and task-level parallel computation are supported to accelerate simulation.

The rest of this paper is organized as follows. Section II introduces the structure and major classes of STEPS. The two main functions, power flow solution and dynamic simulation, are discussed in Sections III and IV. Section V introduces additional functions, including user-defined modeling and parallel simulation. In Section VI, tests are carried out to verify the accuracy and efficiency of STEPS. Summary and plans are given in Section VII.

II. Structure and Database of STEPS

A. Structure of STEPS

The general structure of STEPS is summarized and shown in Fig. 1. STEPS mainly includes three parts: STEPS kernel, APIs, and Python module stepspy. Third-party packages, including CSparse and CXSparse, are adopted to implement sparse matrix classes for storing real and complex matrix separately [17]. More than 1,300 unit tests have been passed to ensure the quality of the kernel. On the top layer of the STEPS kernel are APIs, which are linked to most functions defined in STEPS and can be called by external programs. A Python module named stepspy is provided to further encapsulate APIs' functions for simplifying the users' operation.

The kernel of STEPS is coded in C++ following object-oriented programming with all classes and functions named in a readable way. A class named *STEPS* is the major class for creating a simulation toolkit object. The *STEPS* class holds several major private objects:

1) A power system database for storing all device data. It is an object of class *POWER_SYSTEM_DATABASE*.
2) A dynamic model database for storing dynamic model data. It is an object of class *DYNAMIC_MODEL_DATABASE*.
3) A network matrix for building network admittance matrix. It is an object of class *NETWORK_MATRIX*.
4) A power flow solver for solving power flow. It is an object of class *POWERFLOW_SOLVER*.
5) A dynamic simulator for running dynamic simulation. It is an object of class *DYNAMIC_SIMULATOR*.

Besides the above classes in the *STEPS* class, there are some other major classes defined in STEPS, such as:

1) A real sparse matrix is an object of class *STEPS_SPARSE_MATRIX,* and a complex sparse matrix is an object of class *STEPS_COMPLEX_SPARSE_MATRIX*. A Jacobian matrix is also an object of *STEPS_SPARSE_MATRIX* for solving power flow and running dynamic simulation.
2) The Jacobian matrix for solving power flow is built by an object of class *JACOBIAN_BUILDER*.
3) Data importer and exporter for loading and exporting power flow and dynamic data are implemented as objects of class *DATA_IMEXPORTER*. Currently, STEPS supports power flow data in PSS/E *raw* format and BPA *dat* format. For dynamic simulation, STEPS supports PSS/E *dyr* format.

Details about the above classes and modules are discussed in the following subsections. Since power flow and dynamic simulation are the two major functions of STEPS, they are discussed in Sections II and III in detail.

B. Structure of Class POWER_SYSTEM_DATABASE

The *POWER_SYSTEM_DATABASE* class holds all device data of the power system and provides interfaces for setting and retrieving data. The following variables are included in class *POWER_SYSTEM_DATABASE*:

1) system base power S_{base} in MVA;
2) vectors for storing all device models;
3) indexing map of all devices.

Many types of devices are supported for building the power system model. Table I shows the supported devices of STEPS and some other open-source packages.

Supported devices are discussed as follows:

TABLE I
COMPARISON OF SUPPORTED DEVICE MODELS IN OPEN-SOURCE TOOLS

Device models	STEPS	MATPOWER	PYPOER	PSAT	PyPSA	pandapower
bus	•	•	•	•	•	•
synchronous generator	•	•	•	•	•	•
WT generator	•	○	○	•	○	○
PV unit	•	○	○	○	○	○
energy storage	•	○	○	○	○	○
load	•	•	•	•	•	•
fixed shunt	•	•	•	•	•	•
transmission line	•	•	•	•	•	•
2-winding transformer	•	•	•	•	•	•
3-winding transformer	•	○	○	•	○	•
HVDC link	•	•	○	•	•	•
equivalent device	•	○	○	○	○	•

Note: • for supported, ○ for not supported.

1) Bus. Bus comprises the nodes for other devices to connect to. Its properties include bus number, bus name, bus type, base voltage, base frequency, voltage magnitude, and voltage angle. In STEPS, all buses are indexed with bus numbers. The bus type can be one of the four basic types: out of service, P-Q, P-V, and slack. It is allowed in STEPS to set up multiple slack buses in a synchronized island. The initial voltage angle of slack buses can be non-zero to tune power flow.
2) Power source. Synchronous generator, wind turbine (WT) generator, photo-voltaic (PV) unit, and energy storage are supported power sources modeled in STEPS. They have similar properties, including bus number, source identifier, status flag, base capacity, active and reactive power generation, and active and reactive power limits.

Synchronous generator, WT generator, PV unit, and energy storage are modeled in the *raw* file of the same type. An additional column is added to the end of each generator data in the *raw* file to distinguish them: 0 for synchronous generator, 1 for WT generator, 2 for PV unit, and 3 for energy storage.

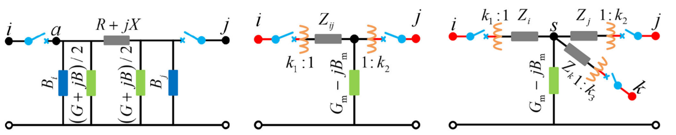
3) Load. In STEPS, voltage-dependent ZIP load is modeled with constant power, constant current, and constant impedance parts. The constant power part is transformed into constant current when load bus voltage V drops beyond threshold V_{th} to improve power flow convergence. Two types of transformation are supported in STEPS: elliptical (1) and linear (2):

$$\frac{(V-V_{th})^2}{V_{th}^2}+\frac{I^2}{I_{\max}^2}=1 \tag{1}$$

$$I=I_{\max}V/V_{\mathrm{th}} \tag{2}$$

where I is the current of constant power load, $I_{\max}=S_{\mathrm{P0}}/V_{\mathrm{th}}$, and S_{P0} is the nominal constant power load. The typical value of V_{th} is 0.7 p.u. and can be configured in *LOAD* class.

4) Fixed shunt. In STEPS, the fixed shunt is used to model capacitor and inductor. Different from the constant impedance load, fixed shunts are included in the network matrix, while constant impedance loads are excluded.
5) Transmission line. The AC transmission line is modeled in the Π form, as shown in Fig. 2(a). In the AC line model, R, X, G, and B are the resistance, reactance, conductance, and susceptance of the line. B_i and B_j are the susceptances of reactive compensation on i and j side of the line.



(a) Transmission Line (b) 2-winding Transformer (c) 3-winding Transformer

Fig. 2. Models of transmission line and transformer in STEPS.

Line breakers are also modeled. For a line with one breaker open, say breaker at side i is open, it is possible to create an artificial bus to represent the point a in Fig. 2(a). However, this would lead to an increase of system bus number. To overcome the problem, STEPS uses an equivalent admittance Y_{e} to represent the line looking from bus j into the line as in (3).

$$Y_{\mathrm{e}}=jB_j+\frac{G+jB}{2}+\frac{1}{R+jX+\frac{1}{jB_i+\frac{G+jB}{2}}} \tag{3}$$

6) Transformer. In STEPS, both two-winding and three-winding transformers are modeled with the class *TRANSFORMER*. The two-winding transformer in STEPS is shown in Fig. 2(b), where Z_{ij} is the leakage impedance, $G_{\mathrm{m}}-jB_{\mathrm{m}}$ is magnetizing admittance, and k_1 and k_2 are the off-nominal turn ratio of the primary and secondary winding, respectively. The three-winding transformer is represented with magnetizing impedance at the artificial star bus s to model the excitation voltage, as shown in Fig. 2(c), where Z_i, Z_j, and Z_k are leakage impedance of the primary, secondary, and tertiary windings, and k_3 is the off-nominal turn ratio of the tertiary winding.

Similar to line breakers, transformer winding breakers are also modeled to enable simulation of tripping or closing winding breakers. Equivalent circuits are also built to model the openness of winding breakers.

7) HVDC. STEPS adopts the quasi-static HVDC model from PSS/E [18] with the difference of determination of converter transformer tap. The converter transformer tap in STEPS is always optimized to minimize the firing angle to reduce reactive power consumption. However, for PSS/E, the converter transformer tap can be fixed even though excessive reactive power may be absorbed from the grid. In normal operation modes, the rectifier holds DC current or power command, and the inverter holds DC voltage or gamma angle. When the rectifier's AC voltage drops, the rectifier may fail to hold DC power or current command. In this case, the rectifier firing angle is held at its minimum value, and the inverter tries to hold DC current with a reduced current order.

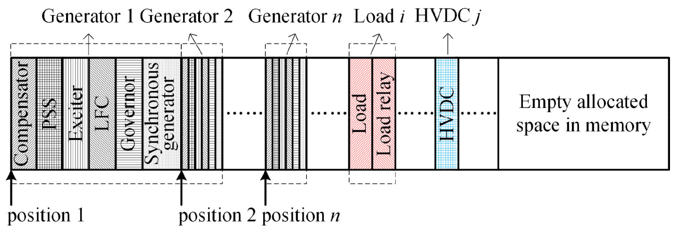


Fig. 3. Storage of dynamic models.

8) Equivalent Device. The equivalent device is used to improve simulation flexibility by representing special devices with equivalent active and reactive power. It is modeled as a combination of equivalent ZIP load and voltage source.

It should be emphasized that STEPS has no limit on the number of devices as long as the operating system and RAM support. The supported maximum number of devices can be configured via *steps_config.json* file or corresponding APIs.

C. Structure of Class DYNAMIC_MODEL_DATABASE

Dynamic models are vital to system's dynamic behavior. The class *DYNAMIC_MODEL_DATABASE* is used to store all dynamic models except for the bus frequency model. Interfaces are provided for setting and retrieving data. Supported dynamic model categories in STEPS are listed as follows:

1) Synchronous generator-related models: voltage compensator, stabilizer, exciter, turbine load controller, turbine governor, and synchronous generator.
2) WT generator-related models: wind speed, relay, pitch, aerodynamic, electrical, wind turbine, and WT generator.
3) PV unit related models: converter. PV panel, electrical, and irradiation models are under development.
4) Energy storage-related models: converter. Models of battery and its management systems are under development.
5) Load-related models: static and dynamic load characteristics, under-frequency and under-voltage load shedding.
6) HVDC-related models: quasi-static HVDC.

For the above dynamic model categories, some widely used models are implemented in STEPS. Some of them are listed as follows: Synchronous generator: GENCLS, GENROU, and GENSAL of PSS/E; Exciter: IEEET1 of PSS/E, type 1 to 14 exciter models of PSASP; Turbine governor: IEEEG1, IEEEG2, IEEEG3, IEESGO, and TGOV1 of PSS/E; Stabilizer: IEE2ST of PSS/E, type 1 to 6 PSS models of PSASP; WT generator: WT3G1 and WT3G2 of PSS/E, WT3G0; PV converter: PVGU1 of PSS/E; Load: IEEL and CIM6 of PSS/E; Load relay: UFLS, UVLS; HVDC: CDC6T of PSS/E [18], [19].

To load dynamic data, *DYNAMIC_MODEL_DATABASE* allocates a contiguous memory space with a size defined by users. As shown in Fig. 3, dynamic models are continuously stored in this space to improve data locality for alleviating the cache miss problem [20], [21]. Users can easily acquire the actual memory space occupied by dynamic models from the detailed log and adjust the memory space with API.

The bus frequency model is used to calculate bus frequency for load characteristics and UFLS models. In STEPS, each bus is automatically equipped with a bus frequency model. It is implemented as a differential block with the input of bus voltage angle and the output of frequency deviation. The filter time constant is set as quadruple system simulation time step.

To extend STEPS, more dynamic models should be added. To simplify the modeling procedure, commonly used blocks are implemented in STEPS, including proportional, integral, differential, PID, PI, PD, first-order, second-order, and lead-lag block. Besides, saturation block and continuous buffer are also implemented to simulate saturation characteristics and time latency of specific control.

D. Structure of Class NETWORK_MATRIX

The class *NETWORK_MATRIX* serves to build a network matrix based on the devices in *POWER_SYSTEM_DATABASE*. It provides interfaces to build the following sparse matrix:

1) full admittance matrix Y for power flow and dynamic solution, which is stored as an object of class *STEPS_COMPLEX_SPARSE_MATRIX*;
2) susceptance matrix B' and B" for active power-angle and reactive power-voltage solution of fast decoupled power flow solution, which is stored as an object of class *STEPS_SPARSE_MATRIX*.

The *NETWORK_MATRIX* class also provides interfaces to optimize bus ordering and store physical-internal bus pairs' mapping index. In STEPS, the initial Y matrix is first built by assigning an internal bus number to each bus according to the data importing sequence. The Y matrix is then optimized with the approximate minimum degree (AMD) algorithm to permute the internal bus numbers. It helps to reduce non-zero fill-in when solving linear algebraic equations [22].

E. APIs of STEPS and Stepspy Module

APIs for the most commonly used functions are implemented in STEPS to facilitate users to use high-level languages to call specific functions. Furthermore, multiple functions are combined and encapsulated into advanced APIs to reduce the coding burden significantly for using STEPS.

Though APIs are easy to use, they require package compilation for building new applications. To avoid repeated compile and link of C++ codes, the STEPS package is first compiled into a dynamic link library (DLL) on the Windows platform or a shared object (SO) on Unix-like systems. Further applications can call the APIs in the DLL or SO file.

Since Python language is popular and easy to use, a module named stepspy is developed to wrap the DLL or SO. Some high-level APIs are also developed to simplify the simulation. The stepspy module has been uploaded to the Python Package Index (PyPI) and can be installed via pip tool. Though stepspy can be used in Python 2, it is strongly suggested to use stepspy in Python 3 since Python 2 is no longer officially maintained.

III. Power Flow Solution

A. Basic Solutions

Class *POWERFLOW_SOLVER* provides functions to solve power flow. Before solving power flow, it reads data from the power system database and network matrix. When solving power flow of AC system, Newton-Raphson (NR) and fast decoupled (PQ) methods can be called. For a hybrid AC/DC system, an alternate method is adopted in STEPS. In each iteration, each converter's AC power is calculated with the input of the solved AC voltage. Then, the AC power of each converter is added to AC bus power mismatch, and the AC voltage is updated with the NR or PQ method.

B. Features of Power Flow Solution

To balance efficiency and convergence, STEPS provides several configurations for the power flow solution. Besides basic options such as convergence tolerance and maximum iteration count, additional options listed as follows:

1) flat start logic. In some open-source tools, only flat start is supported, and power flow may diverge for large-scale systems. In STEPS, both flat start and non-flat start are supported to deal with this problem.
2) var limit check logic. Power flow is difficult to converge if reactive power is not locally balanced. When var limit check logic is enabled, reactive power on each P-V bus is compared with the limitation before each iteration. The bus type may be converted to P-Q type if reactive power exceeds the limit. By disabling the var limit check logic, power flow is easy to converge. Reactive imbalance can then be identified with the converged result. Therefore, for systems hard to converge, it is suggested to disable var limit check logic first and then enable it to get the true result by tuning system configuration.
3) iteration accelerator α. The accelerator is used to update the bus voltage and angle as follows:

$$\begin{cases}\theta_{k+1}=\theta_k+\alpha\Delta\theta\\ V_{k+1}=V_k+\alpha\Delta V\end{cases}\tag{4}$$

where θ_k and V_k are the angle and voltage of the *k*-th iteration, and $\Delta\theta$ and ΔV are angle and voltage update calculated with NR or PQ iteration. α is usually configured within (0.2, 2.0). For systems hard to converge, α can be tuned to less than 1.0 to reduce the update of voltage and angle in each iteration. It can effectively improve power flow convergence.

4) non-divergent solution logic. Enabling non-divergent solution logic is another efficient way to improve the convergence for systems hard to converge. The logic is realized by changing the iteration accelerator α automatically. It first updates bus voltage and angle to get initial voltage magnitude $V^0{}_{k+1}$ and angle $\theta^0{}_{k+1}$ with:

$$\begin{cases}V_{k+1}^0=V_k+\Delta V_{k+1}\\ \theta_{k+1}^0=\theta_k+\Delta\theta_{k+1}\end{cases}\tag{5}$$

Then the bus power mismatch is checked. If the maximum bus power mismatch exceeds that of the previous iteration, α is halved to update bus voltage angle and magnitude with:

$$\begin{cases}V_{k+1}^i=V_{k+1}^{i-1}-\alpha_i\Delta V\\ \theta_{k+1}^i=\theta_{k+1}^{i-1}-\alpha_i\Delta\theta\\ \alpha_i=0.5^i\end{cases}\tag{6}$$

where $i\geq1$.

For the worst case, if α is reduced to 0, bus voltage angle and magnitude are no longer updated, and the bus power mismatch is identical to the previous iteration. Therefore, with the non-divergent logic enabled, the power flow solver can be guaranteed to get a solution no worse than the initial solution. In STEPS, α is set to be halved by ten times at most. If the power flow solution exits with $i=10$, no further solution is attempted. The result of the last iteration can provide abundant information to find the bottleneck of system convergence.

5) maximum increment of bus voltage angle and magnitude. Sometimes, each step's updating values are too great. It may lead to power flow blown-up even though α is reduced or non-divergent solution logic is enabled. In this case, it is suggested to set the upper limit of ΔV and $\Delta\theta$ to further improve the convergence for solving large-scale power systems.
6) export Jacobian logic. When solving power flow with NR, the Jacobian matrix of each step can be exported as a *csv* file. It is helpful for studies such as static analysis, voltage and active power control, and sensitivity analysis.
7) solving power flow cases with combined methods successively. NR solution converges fast if proper initial values are given, and PQ solution is less sensitive to the initial values. Therefore, it is recommended to combine the two methods for a better power flow solution. First, PQ solution is performed for several iterations with flat start logic enabled. If the power flow is hard to converge, non-divergent solution logic should also be enabled. After several iterations, *V* and θ are probably close to the final solution, and convergence difficulty is reduced. The NR solution is then performed with flat start logic disabled to solve power flow with the bus voltage angle and magnitude solved by the PQ solution as initial values. Non-divergent solution logic can be disabled to improve solution speed.

IV. Dynamic Simulation

With the initial steady-state provided by converged power flow, the *DYNAMIC_SIMULATOR* class provides functions to run dynamic simulations. It reads device data from the power system database, dynamic model data from the dynamic model database, and dynamic admittance matrix from the network matrix. Details about the dynamics simulation function are discussed in the following sections.

A. Basics of the Dynamic Simulation Method

For dynamic simulation, the core is to solve differential-algebraic equations (DAE) as follows:

$$\begin{cases}x=\dot{f}\left(x,y\right)\\ 0=g\left(x,y\right)\end{cases}\tag{7}$$

where *x* is the state variable, and *y* is the operational variable.

For differential equations, there are two kinds of integration methods: explicit and implicit. In some software, explicit integration is used in dynamic simulation, such as Euler, Modified Euler, and Runge-Kutta. For implicit integration, the most common one is trapezoidal integration. The truncation error of trapezoidal integration is the same as that of Modified Euler and is easy to implement. Therefore, trapezoidal integration is selected as the default method to solve differential equations in STEPS.

There are generally two kinds of methods to solve differential and algebraic equations: simultaneous and alternate. The simultaneous method usually accompanies the implicit integration rule. It first discretizes differential equations into difference equations and then combines them with the algebraic equation. With the trapezoidal integration rule, the simultaneous method can be modeled as:

$$\begin{cases} 0 = -x_{n+1} + x_n + \frac{h}{2}\left[\dot{f}\left(x_n, y_n\right) + \dot{f}\left(x_{n+1}, y_{n+1}\right)\right] \\ 0 = g\left(x_{n+1}, y_{n+1}\right) \end{cases} \tag{8}$$

where *h* is the simulation time step, x_n and y_n are variables of the *n*-th step, and $x_{n\,+\,1}$ and y_{n+1} are variables of the *n*+1-th step.

Then, the DAE solution problem becomes a nonlinear algebraic equation solution problem. The idea of NR solution can be adopted to solve it.

Although differential equations can be transformed into difference equations, there are still some problems. First, the number of equations in (8) will change when limiters are considered. Second, saturation in the generator or AC exciter models would lead to the change of coefficients in (8). To overcome these problems, the alternate method with implicit trapezoidal integration rule is adopted in STEPS. It can be expressed as follows:

$$\begin{cases} x_{n+1}^{k} = x_n + \frac{h}{2}\left[\dot{f}\left(x_n, y_n\right) + \dot{f}\left(x_{n+1}^{k}, y_{n+1}^{k}\right)\right] \\ 0 = g\left(x_{n+1}^{k}, y_{n+1}^{k+1}\right) \end{cases} \tag{9}$$

In (9), *k* is the count of DAE iterations in the *n*+1-th step, and the initial value is $x_{n+1}^0 = x_n, y_{n+1}^0 = y_n$. Since alternate error always exists in the alternate method, it is important to reduce alternate error by increasing *k*. Theoretically, if the maximum value of *k* is infinite, the alternate error can be reduced to 0.

B. Dynamic Simulation Features

In STEPS, some key parameters can be configured to run the dynamic simulation:

1) allowed maximum bus power imbalance in MVA. It is used to control whether the solution to the algebraic equations is converged or not. It can be represented by:

$$\Delta S_{\max} < \varepsilon_{\text{th}} \tag{10}$$

where $\Delta S_{\max}$ is the maximum bus power imbalance, ε_{th} is the threshold. Typically, ε_{th} can be set as 0.1 MVA. The criterion of maximum current imbalance $\Delta I_{\max}$ can also be used as (11) to reduce the calculation of nodal power imbalance:

$$\Delta I_{\max} < \varepsilon_{\text{th}}/S_{\text{base}} \tag{11}$$

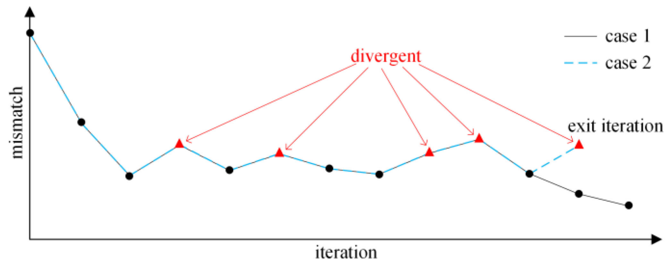


Fig. 4. Demonstration of the early exit of network solution.

2) minimum and maximum DAE iteration count. It is recommended to set the minimum value of *k* as 3 to guarantee simulation accuracy. DAE iteration should stop when (10) or (11) is satisfied. However, the mismatch can be huge when a severe disturbance is applied. It may lead to a large number of DAE iteration and slow down the simulation. To guarantee simulation speed, the maximum DAE iteration count can be limited. DAE iteration stops automatically when the iteration count exceeds the maximum value. As a special case, the DAE iteration count is fixed if the minimum and maximum values are equal.
3) maximum network iteration count. It is similar to the maximum iteration count of power flow and used to terminate the divergent iteration when solving algebraic equations.
4) iteration accelerator and non-divergent logic. They perform similar to the iteration accelerator and non-divergent logic used in the power flow solution. It is especially useful when events are applied and simulation is hard to converge.
5) early exit of network solution. Although maximum network iteration count can be set to exit network solution automatically, long time consumption is still observable if the network solution is always divergent. To solve this problem, STEPS enables users to set a divergent count threshold. Suppose the error of the current network iteration is greater than that of the previous one. In this case, the current iteration is treated as divergent, and the divergence count increases by one. Fig. 4 shows an example with the divergent count threshold of 4. If the network solution is divergent, the divergent count increases quickly and reaches the threshold. Then STEPS exits the network solution, as shown in case 2 of Fig. 4.
6) rotor angle stability surveillance logic and its threshold. The maximum rotor angle difference of each synchronized electrical island can be monitored when the logic is enabled. Once the value exceeds the threshold, the system is judged as unstable. Then dynamic simulator quits, and no more simulation is conducted. The typical threshold is 360°.
7) parallel computation. In STEPS, OpenMP is adopted for parallel computation when running dynamic simulation [23]. The parallel thread number can be user-configured according to the number of CPU cores. Details of the kernel-level parallelization are discussed in Section V-B.

C. Events

The primary function of dynamic simulation is to analyze the system response in case of disturbances or operations. Those disturbances and operations are collectively referred to as events in STEPS. Currently, supported events are:

1) Bus related events: bus fault, bus fault clearance, trip bus;
2) Line related events: line fault, line fault clearance, trip line, close line, trip line breaker, close line breaker;
3) Transformer related events: trip transformer, close transformer, trip transformer breaker, close transformer breaker;
4) Generator related events: trip generator, shed generator, manually change exciter reference, manually change turbine governor reference;
5) Load related events: trip load, close load, scale load;
6) Fixed shunt related events: trip fixed shunt, close fixed shunt;
7) HVDC related events: manually bypass HVDC, manually unbypass HVDC, manually block HVDC, manually unblock HVDC, manually change HVDC power order.

In STEPS, events can be set at any time during the dynamic simulation. It is also allowed to set multiple events in a simulation, and different events can be set simultaneously.

D. Meters

Meters can be defined to monitor system dynamics when running dynamic simulations. The supported meters can be found in the implementation of class *METER*. The most commonly used meters are listed as follows:

1) bus: voltage magnitude, voltage angle, frequency;
2) synchronous generator: terminal active and reactive power, terminal current, rotor angle, rotor speed, excitation voltage, mechanical power;
3) WT generator: active and reactive power, terminal current, active power or current command, voltage or reactive current command, rotor and turbine speed, pitch angle;
4) PV unit: terminal active and reactive power, terminal current, active power or current command, reactive voltage or current command;
5) Line and transformer: active and reactive power, current, apparent impedance;
6) HVDC: firing angle, extinction angle, converter active and reactive power, converter DC voltage, converter current;
7) Load: active and reactive power, current, shed scale.

Theoretically, all variables in STEPS can be monitored. The DAE iteration count, network iteration count, maximum power mismatch, and the bus with the maximum power mismatch of each step are automatically monitored as basic meters to check the DAE solution's convergence. They can help to validate dynamic simulation and tune dynamic simulation parameters for improving validity. All the monitored meters can be stored in a binary file or text files in *csv* and *json* format.

V. Additional Functions

In addition to the two basic functions of power flow solution and dynamic simulation, STEPS also provides additional functions to improve scalability and efficiency. In this section, the implementation and usage of two additional functions, a.k.a. user-defined modeling and parallelization, are introduced.

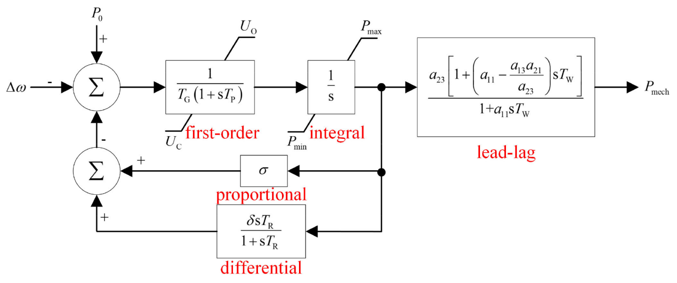


Fig. 5. Transfer function block diagram of IEEEG3.

A. User-Defined Modeling

As mentioned in Section II-C, basic blocks are provided in STEPS for simplifying the modeling process. They can also be used for user-defined modeling. Procedures of user-defined modeling are as follows:

1) Select the needed basic blocks according to the transfer function block diagram of the user-defined model.
2) Declare the definition of the user-defined model in a header file. The header file should be put in the folder of the corresponding dynamic model category under header/model in the STEPS project.
3) Implement the user-defined model. The implementation file should be created in the folder of the corresponding dynamic model category under source/model in the STEPS project. All the functions defined in the header file must be implemented here. Of all the functions, the two most important ones are initialize() and run(). The former is used to acquire each block's initial state based on the initial outputs obtained from the power flow solution. The latter is used to integrate states and update outputs based on inputs in the time-domain simulation process.

Based on the above extensions, the importing interface of the user-defined model can be updated in the class *STEPS_IMEXPORTER*. Codes for acquiring the user-defined model's memory consumption should be added in the function get_model_size() of class *DYNAMIC_MODEL_DATABASE*.

To show the modeling process more specifically, IEEEG3 is taken as an example here. Fig. 5 shows that the needed basic blocks for building IEEEG3 include first-order, integral, lead-lag, proportional, and differential blocks. Then, the header and implementation files for IEEEG3 are created in folder header/model/sg_models/turbine_governor_model/ and source/model/sg_models/turbine_governor_model/, respectively. Users can check the specific implementation of IEEEG3 in the source code of STEPS. Finally, interface add_IEEEG3_model() is defined in class *STEPS_IMEXPORTER*, and codes for acquiring the size of IEEEG3 is implemented in *DYNAMIC_MODEL_DATABASE*.

Besides adding the above compulsory codes, it is also recommended to carry out unit tests for the user-defined model. Please refer to built-in unit tests in STEPS.

Though the process of user-defined modeling is not complicated, users still need basic knowledge of C++. To further simplify user-defined modeling, compilation-free scripts for building dynamic models are under development.

B. Kernel-Level Parallelization

As mentioned in Section IV-B, kernel-level parallelization is implemented with OpenMP in STEPS. It is supported in dynamic simulation but not in powerflow solution because the time consumption of power flow solution is much less than that of dynamic simulation. The working procedures of kernel-level parallelization in STEPS are as follows:

1) Set the parallel thread number n_k according to the number of CPU cores N and the size of the test system. The selection of thread number is further discussed in Section VI-D. Kernel-level parallelization is enabled when the thread number is greater than 1. When using the stepspy module, kernel parallelization can be enabled with the following codes:

```
from stepspy import STEPS
simulator = STEPS(is_default = True)
simulator.set_parallel_thread_number(nk) #nk > 1
```

2) Acquire the number of dynamic devices, N_m, which can be easily obtained in *POWER_SYSTEM_DATABASE*.
3) Allocate dynamic devices to parallel threads. In STEPS, static scheduling is used to realize kernel-level parallelization. Each thread is statically allocated with about $N_\mathrm{m}/n_\mathrm{k}$ devices.

It should be noted that the kernel-level parallelization is disabled automatically if multiple devices of the same category are connected to the same bus. If kernel-level parallelization is enabled, two or more threads with devices connecting to the same bus may write to the same current injection vector location at the same time. It is prone to cause write conflicts. Therefore, users should guarantee that no more than one device of the same category is connected to the same bus before enabling kernel-level parallelization. One possible solution is to add a zero impedance switch between multiple devices of the same category at the same bus.

At present, STEPS only supports the parallelization on CPU. GPU-based parallelization is deserved to be developed in the future to achieve higher performance [24].

C. Task-Level Parallelization

Task-level parallelization is implemented in STEPS by creating multiple toolkits to deal with tasks at the same time. Therefore, it saves time only when multiple tasks are performed. When using the stepspy module, task-level parallelization can be realized with multiple processes and create STEPS object for each process with *is_default* option disabled. Procedures of the task-level parallelization are as follows:

1) Set the number of parallel processes n_s according to the number of CPU cores and task number. When task-level parallelization is enabled, the kernel-level parallelization is usually disabled to maintain computation efficiency. If they are both enabled, n_s should be no more than N/n_k.
2) Assign tasks to parallel processes. When the task in a parallel process is finished, a new task will be automatically added to the parallel process until all tasks are completed.

An example of performing some task for all buses using task-level parallelization is given as follows:

```
from multiprocessing import Pool
from stepspy import STEPS
def run_task(bus):
    sim2 = STEPS(is_default=False)
    # here goes details of the task

if __name__=='__main__':
    sim = STEPS(is_default=True, log_file="demo.log")
    sim.load_powerflow_data('demo.raw','PSS/E')
    buses = sim.get_all_buses()
    pool =Pool(processes=20) #create 20 processes
    for bus in buses:
        pool.apply_async(run_task, (bus,))
    pool.close()
    pool.join()
```

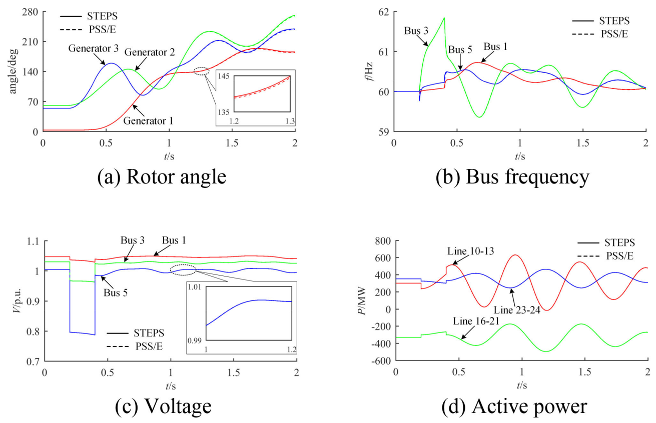


Fig. 6. Dynamic simulation results in STEPS and PSS/E.

VI. Case Studies

In this section, several cases are provided to show the accuracy and application scenarios of dynamic simulation in STEPS. All tests are performed with the stepspy module on a server with Intel Exon Gold 6148 CPU with 20 cores.

A. Accuracy Verification

To verify STEPS's accuracy, the IEEE 9-bus model and New England 39-bus model are tested with a three-phase fault applied on bus 6 at 0.2s and cleared at 0.4s. Results of STEPS and PSS/E are shown in solid and dotted lines separately. Rotor angle and frequency of the 9-bus model are shown in Fig. 6(a) and (b). Voltage and active power on transmission lines of the 39-bus model are shown in Fig. 6(c) and (d). It can be found that STEPS can produce almost the same result as PSS/E, and the accuracy of STEPS can be guaranteed.

Since the alternate solution is used to solve DAE, the alternate error is an important factor affecting dynamic simulation

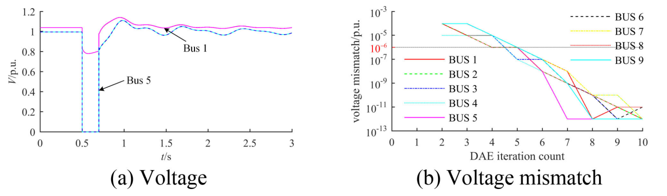

(a) Voltage (b) Voltage mismatch

Fig. 7. Voltage and its mismatch with different $k_{\max}$.

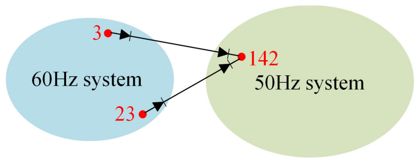


Fig. 8. Diagram of the tested interconnected system.

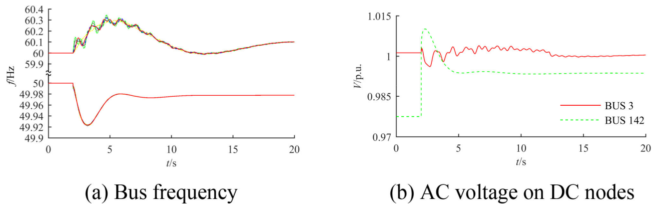

(a) Bus frequency (b) AC voltage on DC nodes

Fig. 9. Simulation results of a system with different base frequencies.

accuracy. To quantify the effect, 2, 3, 4, 5, 10 are set as the maximum DAE iteration number $k_{\max}$ on the IEEE 9-bus model. A three-phase fault is applied on bus 5 at 0.5 s and cleared at 0.7 s. The voltage of bus 1 and 5 with different $k_{\max}$ are shown in Fig. 7(a). It is shown that the difference between voltage curves with different $k_{\max}$ is negligible. In other words, the alternate error is acceptable.

To find the proper value of $k_{\max}$, the voltage of all buses with $k_{\max} = 20$ is taken as reference, and the voltage mismatch with different $k_{\max}$ is shown in Fig. 7(b). It can be found that the mismatch is generally reduced by an order of magnitude with one more DAE iteration. For $k_{\max} = 3$, the mismatch is less than 10^{-3} p.u. and is reduced to less than 10^{-6} p.u. when $k_{\max} \geq 5$.

B. Simulation of Interconnected System With Different Base Frequencies

As mentioned in Section II-C, each bus in STEPS is equipped with a nominal frequency and bus frequency model. Therefore, it is possible to check the interaction between physical systems with different nominal frequencies with STEPS. A simulation is carried out in a system with 169 buses to show an interconnected system's dynamics with two nominal frequencies. It has a 50Hz part with 130 buses and a 60 Hz part with 39 buses, and two HVDC links connect the two parts. Fig. 8 shows that rectifiers are located in the 60Hz part at bus 3 and 23, and inverters are located in the 50Hz part at bus 142.

In the test system, the active power order of each HVDC is 500MW. The HVDC linking bus 23 and 142 is blocked at 2s. The frequency of the two parts is shown in Fig. 9(a), and the voltage of bus 3 and 142 are shown in Fig. 9(b). It can be seen that the dynamics of both 50Hz and 60Hz parts can be obtained as a whole directly with STEPS.

Simulation results show STEPS's capability of simulating systems with different base frequencies. Therefore, STEPS is promising to be used in the uniform study of international interconnection with multiple nominal frequencies. The uniform simulation can better illustrate interconnected systems' interactive response than tools with a single base frequency.

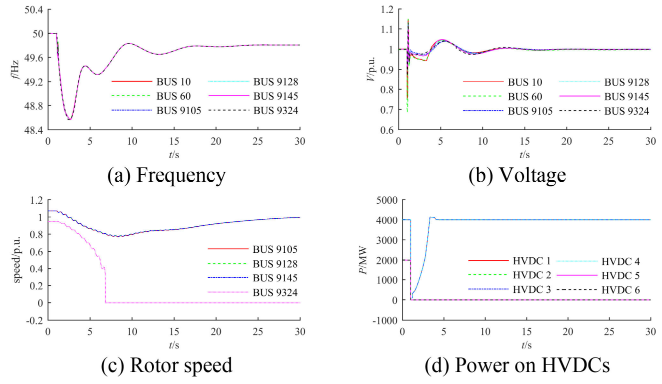

(a) Frequency (b) Voltage
(c) Rotor speed (d) Power on HVDCs

Fig. 10. Dynamic simulation results in a system with renewable generation.

C. Simulation of System With WT Generators and HVDC

Currently, power electronic-based power source and transmission are developing rapidly. A provincial power grid of 146 buses with 14 equivalent WT generators and 6 HVDC links is examined to demonstrate STEPS in complex systems with DFIG and HVDC. A three-phase fault is applied on bus 80 at 1s and cleared after five cycles. Parameters of WT generators on bus 9324 are set to abnormal values to show the activation of under-speed relay logic. The frequency, voltage, and rotor speed of WT generators are shown in Fig. 10(a) to (c) separately. The power of HVDC links is shown in Fig. 10(d).

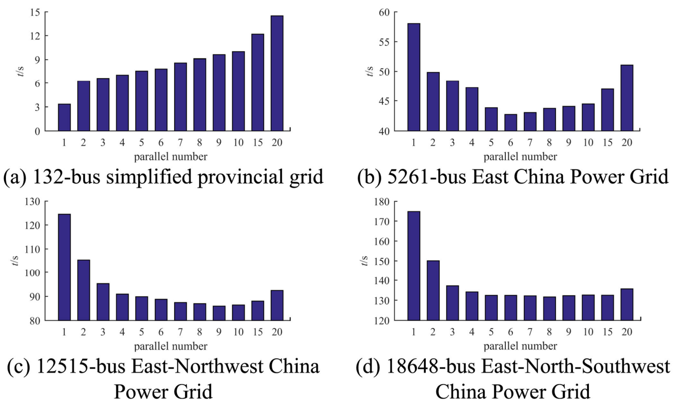
Figures show that 4 HVDC links are bypassed, and 2 HVDC links are blocked, causing 4GW power loss and significant frequency drop. After the fault is cleared, 4 HVDC links recover, and the frequency begins to rise. During the dynamics, the rotor speed of WT generators is decreasing to increase power generation for supporting system frequency. However, the rotor speed of WT generators on bus 9324 drops beyond 0.3 p.u. at about 6s and is then tripped due to the under-speed relay logic in WTRLY0 model.

D. Simulation Speed

The same dynamic simulations are carried out in STEPS and PSS/E separately to test the simulation speed. IEEE 9-bus model, New England 39-bus model, and a provincial grid model of 132 nodes are tested. The simulation time step is 2ms, and the simulation duration is 10s. Events and time consumption are shown in Table II. It can be found that the simulation speed of STEPS is about 1/4 to 1/2 of PSS/E.

TABLE II
COMPARISON OF SIMULATION TIME BETWEEN STEPS AND PSS/E

Model	event	$t_{PSS/E}$/s	t_{STEPS}/s
9-bus	Line fault	1.214	2.105
39-bus	Line fault	1.512	2.935
132-bus	DC block	4.260	17.113



(a) 132-bus simplified provincial grid (b) 5261-bus East China Power Grid

(c) 12515-bus East-Northwest China Power Grid (d) 18648-bus East-North-Southwest China Power Grid

Fig. 11. Simulation time with different parallel thread numbers.

Tests are also carried out to show the efficiency of OpenMP parallel computation. Test models are the 132-bus simplified provincial grid model, the 5261-bus East China Power Grid model, the 12515-bus East-Northwest China Power Grid model, and the 18648-bus East-North-Southwest China Power Grid model, having 39, 533, 1394, and 2186 generators, respectively. At 1s, two HVDC links are manually blocked. With a simulation duration of 10s, the simulation time with different parallel numbers of different test systems is shown in Fig. 11.

It is shown in Fig. 11(a) that serial simulation is faster than parallel simulation when the system size is small. The parallelization efficiency increases with the increase of system size in general. Therefore, it is recommended to enable kernel-level parallelization for large-scale systems. Fig. 11(b) to (d) show that the maximum simulation speed when parallelization is enabled can reach about 1.5 times the serial simulation. It can also be seen from Fig. 11(b) to (d) that the simulation speed is slowed down if the thread number is greater than about 8. The main reason may lie in the limited cache of the CPU. With the increase of thread number, more data can be processed in a fixed time. However, if the thread number is too high, the cache size will be the bottleneck, and more time is required for the CPU to fetch missing data from RAM. Using CPUs with greater cache size and improving STEPS data structure will help to improve the parallelization efficiency.

VII. CONCLUSION

The proposed power system dynamic simulation package STEPS is open-source software with an independent kernel. It provides functions of power flow and dynamic simulation. Plenty of options are provided in the power flow solver and dynamic simulator to guarantee accuracy and computing speed. To improve computational efficiency, OpenMP is adopted to enable parallel simulation. The APIs of STEPS and wrapped Python module stepspy provides the flexibility of running simulation with STEPS. With tests from small to large-scale power systems, the STEPS is proved to be a reliable package for research and education purposes.

STEPS currently stores devices in an array of structure (AoS) mode, which is friendly to coding but inefficient for heavy computation. The efficiency of kernel-level parallelization needs further improvement, and we are working on it. STEPS is actively updated and maintained on Github for higher performance, more models, and new functions, such as short circuit analysis, eigenanalysis, and graphical user interface.

REFERENCES

[1] *IEEE Recommended Practice for the Design of Reliable Industrial and Commercial Power Systems*, IEEE Standard pp. 493–2007, 2007.
[2] R. Billinton, S. Aboreshaid, and M. Fotuhi-Firuzabad, "Well-being analysis for HVDC transmission systems," *IEEE Trans. Power Syst.*, vol. 12, no. 2, pp. 913–918, May 1997.
[3] Z. Wang and Z. Guo, "Uncertain models of renewable energy sources," *J. Eng.*, vol. 2017, no. 13, pp. 849–853, 2017.
[4] Black system South Australia 28 September 2016, *Australian Energy Market Operator Ltd.*, Melbourne, AU. Mar. 2017. [Online]. Available: https://apo.org.au/node/74886
[5] Technical Report on the Events of 9 August 2019. National Grid ESO, Warwick, UK. [Online]. Available: https://www.nationalgrideso.com/document/152346/download
[6] J. H. Chow and K. W. Cheung, "A toolbox for power system dynamics and control engineering education and research," *IEEE Trans. Power Syst.*, vol. 7, no. 4, pp. 1559–1564, Nov. 1992.
[7] F. Milano, L. Vanfretti, and J. C. Morataya, "An open source power system virtual laboratory: The PSAT case and experience," *IEEE Trans. Educ.*, vol. 51, no. 1, pp. 17–23, Feb. 2008.
[8] R. D. Zimmerman, C. E. Murillo-Sánchez, and R. J. Thomas, "MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education," *IEEE Trans. Power Syst.*, vol. 26, no. 1, pp. 12–19, Feb. 2011.
[9] S. Cole and R. Belmans, "MatDyn, a new matlab based toolbox for power system dynamic simulation," *IEEE Trans. Power Syst.*, vol. 26, no. 3, pp. 1129–1136, Aug. 2011.
[10] PYPOWER 5.1.2. Accessed: Dec. 24, 2020. [Online]. Available: https://github.com/rwl/PYPOWER
[11] pypower-dynamics 1.1, Accessed: Dec. 24, 2020, [Online]. Available: https://pypi.org/project/pypower-dynamics/1.1/
[12] L. Thurner *et al.*, "Pandapower—An open-source python tool for convenient modeling, analysis, and optimization of electric power systems," *IEEE Trans. Power Syst.*, vol. 33, no. 6, pp. 6510–6521, Nov. 2018.
[13] T. Brown, J. Hörsch, and D. Schlachtberger, "PyPSA: Python for power system analysis," *J. Open Res. Softw.*, vol. 6, no. 4, 2018.
[14] D. Krishnamurthy, "Psst: An open-source power system simulation toolbox in python," in *Proc. North Amer. Power Symp.*, Denver, USA, Sep. 2016, pp. 1–6.
[15] M. Zhou and S. Zhou, "Internet, open-source and power system simulation," in *Proc. IEEE Power Eng. Soc. Gen. Meeting*, Tampa, USA, Jul. 2007, pp. 1–5.
[16] F. Milano, "A python-based software tool for power system analysis," in *Proc. IEEE Power Energy Soc. Gen. Meeting*, Vancouver, Canada, Jul. 2013, pp. 1–5.
[17] SuiteSparse 5.6.0, Accessed: Dec. 24, 2020. [Online]. Available: https://people.engr.tamu.edu/davis/suitesparse.html
[18] *Model Library of PSS/E 33.4*, 4th ed., Schenectady, New York, NY, USA, USA, Siemens Power Technologies International, 2013.
[19] *Dynamic Element Model Library User.s Manual of PSASP 7.1*, 1st ed., Beijing, China, China Electric Power Research Institute, 2010.
[20] Q. G. Samdani and M. A. Thornton, "Cache resident data locality analysis," in *Proc. 8th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, San Francisco, USA, Aug. 2000, pp. 539–546.
[21] H. Tomiyama and H. Yasuura, "Size-constrained code placement for cache miss rate reduction," in *Proc. 9th Int. Symp. Syst. Synth.*, La Jolla, USA, Nov. 1996, pp. 96–101.

[22] T. A. Davis, *Fill-Reducing Orderings, in Direct Methods for Sparse Linear Systems*, 1st ed. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2006, ch.7, pp. 99–134.

[23] OpenMP. Accessed: Dec. 24, 2020, [Online]. Available: https://www.openmp.org/.

[24] V. Jalili-Marandi, Z. Zhou, and V. Dinavahi, "Large-scale transient stability simulation of electrical power systems on parallel GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 7, pp. 1255–1266, Jul. 2012.

Changgang Li (Member, IEEE) received the B.E. and Ph.D. degrees in electrical engineering from Shandong University, Jinan, China, in 2006 and 2012, respectively. From 2012 to 2014, he was a Research Scholar with the School of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, Knoxville, TN, USA. He is currently an Associate Research Fellow with the School of Electrical Engineering, Shandong University. His research focuses on the power system operation and control.

Yue Wu received the B.E. degree in electrical engineering from Zhengzhou University, Zhengzhou, China, in 2018. He is currently a Graduate Student with the School of Electrical Engineering, Shandong University, Jinan, China. His research focuses on power system frequency control.

Hengxu Zhang (Member, IEEE) received the B.E. degree in electrical engineering from the Shandong University of Technology, Zibo, China, in 1998, and the M.S. and Ph.D. degrees in electrical engineering from Shandong University, Jinan, China, in 2000 and 2003, respectively. He is currently a Professor with the School of Electrical Engineering, Shandong University. His main research interests include power system security and stability assessment, power system monitoring, and numerical simulation.

Hua Ye (Member, IEEE) received the B.Eng. and Ph.D. degrees in electrical engineering from Shandong University, Jinan, China, in 2003 and 2009, respectively. He is currently a Professor with the Key Laboratory of Power System Intelligent Dispatch and Control, Ministry of Education, Shandong University. His research interests include power system dynamic stability analysis and control, and cyberphysical systems.

Yutian Liu (Senior Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from the Shandong University of Technology, Jinan, China, in 1984 and 1990, respectively, and the Ph.D. degree in electrical engineering from Xi'an Jiaotong University, Xi'an, China, in 1994. He is currently a Chair Professor with the School of Electrical Engineering, Shandong University, Jinan, China. His research interests include power system analysis and control, renewable energy integration, and artificial intelligence application to power system.

Yilu Liu (Fellow, IEEE) received the B.S. degree from Xian Jiaotong University, Xi'an, China, and the M.S. and Ph.D. degrees from Ohio State University, Columbus, OH, USA, in 1986 and 1989, respectively. She is currently the Governor's Chair with the University of Tennessee, Knoxville (UTK), Knoxville, TN, USA, and Oak Ridge National Laboratory (ORNL), Oak Ridge, TN, USA. In 2016, she was elected as a Member of the National Academy of Engineering. She is also the Deputy Director of the DOE/NSF cofunded engineering research center CURENT. Before joining UTK/ORNL, she was a Professor with Virginia Tech, Blacksburg, VA, USA. She led the effort to create the North American power grid frequency-monitoring network (FNET) at Virginia Tech, which is now operated at UTK and ORNL as GridEye. Her current research interests include power system wide-area monitoring and control, large interconnection-level dynamic simulations, electromagnetic transient analysis, and power transformer modeling and diagnosis.