A SPICE Circuit Emulator-based Transient Simulation Tool with Automatic Script-Generation

Zilin Li, Yongli Zhu
Sun Yat-sen University, Guangzhou, China (lizlin28@mail2.sysu.edu.cn, yzhu16@alum.utk.edu)

***Abstract*—This paper introduces a circuit emulator-based power system transient simulation tool with automatic circuit script-generation functionality. The tool automatically generates the SPICE netlist with reusable subcircuits for both network components and synchronous machines, and initializes dynamic states using the power-flow solution. The developed tool features generator models (from second- to sixth-order), four load models, and four types of disturbances. A case study on the nine-bus system demonstrates the workflow's reliability and efficiency, achieving approximately a $12\times$ speed-up while maintaining global generator-speed and bus-voltage mismatches below 1.3×10^{-3} and 9.7×10^{-2} p.u. across all buses and generators.**

***Index Terms*—automatic netlist generation, education tool, power system transient simulation, SPICE circuit emulator**

I. Introduction

Transient stability studies are essential for secure power-system operation. With growing penetration of converter-interfaced generation and declining inertia, stability margins tighten and interactions between system dynamics and controls become more complex [1], increasing the need for faster and reproducible time-domain simulation.

Commercial tools like DIgSILENT and open-source packages like MatDyn [2] have been used in the teaching of power system dynamics courses, but challenges exist. For example, certain commercial tools are not only cost-unaffordable for educators, but also hard for students to learn quickly due to over-complicated GUI designs. Certain open-source tools run at a slow speed and lack support for complex scheduling of disturbance events, flexible switching of device models, easy visualization of simulation results, and future expandability.

SPICE-based time-domain emulators, e.g., Xyce [4], [5], offer an alternative to overcome the above challenges. Xyce requires a SPICE-like *netlist* to specify the network topology, device models, and disturbance events. However, public test systems for transient study are usually given in fixed formats (e.g., MATPOWER [3] or PSSE) with a separate file for parameters of dynamic components (generators, exciters, etc). Thus, the main challenges are to map those raw input files to an electronics-circuit representation that preserves topology and steady-state injections and to encode disturbances (faults, line trips, etc.) as time-dependent parameters. To make Xyce's simulation results comparable *fairly* with established tools (e.g., MatDyn), the most challenging parts are: 1) handling the vast amount of parameters and variant models for dynamic components (generator, exciter, etc.), 2) keeping the initialization process aligned with the referenced power flow solution, and 3) correctly emulating the desired disturbance events.

This paper presents a SPICE-based **education tool** that automates the transient simulation workflow from raw input files to results visualization. The merits are: 1) Reusable libraries are developed, covering synchronous machines (second- to sixth-order), exciters, governors, loads (constant-impedance, constant-power, exponential, ZIP), etc.; 2) Disturbances are modeled by introducing a smoothly changing parameter; 3) The workflow integrates with Qucs-S [6] for one-click execution and plotting. Validation on the IEEE 9-bus system demonstrates close agreement with MatDyn under identical events and tolerances, with notable reductions in the wall-clock solving time.

II. SPICE-based Circuit Emulator for Power System Transient Study

A. Xyce Introduction

Xyce is a SPICE-based circuit simulation engine for large-scale differential-algebraic equation systems. Under modified nodal analysis, power-system quantities are represented by circuit signals, and complex voltages/currents are represented by two coupled channels, real ("*R*") and imaginary ("*I*"):

$$\begin{bmatrix} I_R \\ I_I \end{bmatrix} = \begin{bmatrix} G & -B \\ B & G \end{bmatrix} \begin{bmatrix} V_R \\ V_I \end{bmatrix} \tag{1}$$

In the above $I = YV$ formulation, the solution variables are the real and imaginary parts of bus voltage and current, typically named `VR`, `VI`, `IR`, and `II` in the netlist. Network and control components are implemented as reusable `.SUBCKT` blocks that act on these (V_R, V_I) and (I_R, I_I) channels.

The bottom engine of Xyce finally tries to solve a differential–algebraic (DAE) system in the following form:

$$\mathbf{f}(\dot{\mathbf{x}}, \mathbf{x}, \mathbf{y}, t) = \begin{bmatrix} \mathbf{M}\dot{\mathbf{x}} - \mathbf{g}(\mathbf{x}, \mathbf{y}) \\ -\mathbf{h}(\mathbf{x}, \mathbf{y}) \end{bmatrix} = \mathbf{0}, \tag{2}$$

where $\mathbf{x}$ means the dynamic states and $\mathbf{y}$ means the algebraic variables (e.g., node voltages and currents).

B. Examples of Power Grid Components Modeling in SPICE

Xyce's built-in circuit elements can be utilized to *emulate* various power system components, e.g., the modeling of a slack bus and a constant impedance load is described below.

1) Slack Bus: The rectangular components of the slack bus voltage are fixed from the power-flow solution:

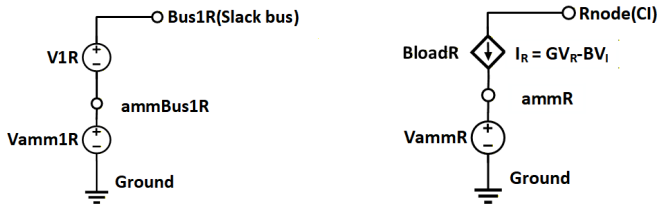
$$V_R = |V| \cos\theta, \qquad V_I = |V| \sin\theta. \tag{3}$$

979-8-3315-8138-1

Node names follow the convention `BusXXR` and `BusXXI`, where `XX` denotes the bus index. Each channel uses a controlled voltage source to impose V_R or V_I, followed in series by a zero-volt source that measures current. The zero-volt source is oriented so that a positive reading means active power injected from the slack generator into the network.

For Bus 1 with $|V| = 1.04$p.u., $\theta = 0°$, the netlist code is:

```
V1R Bus1R ammBus1R 1.04V
V1I Bus1I ammBus1I 0V
Vamm1R 0 ammBus1R 0V
Vamm1I 0 ammBus1I 0V
```



(a) Slack bus (b) CI load

Fig. 1: Example schematic (only real parts)

2) Constant-Impedance Load: The constant-impedance (CI) load is computed based on a steady-state power-flow solution and then absorbed into the network admittance. With active power P_0, reactive power Q_0 and complex bus voltage V_0 (from steady-state), the equivalent shunt admittance is

$$Y_{\text{load}} = \frac{P_0 - jQ_0}{|V_0|^2} = G + jB, \quad G = \frac{P_0}{|V_0|^2}, \quad B = -\frac{Q_0}{|V_0|^2}. \tag{4}$$

Based on the "I=YV" format of Xyce grammar, the load current in the real and imaginary channels reads:

$$I_R = G\,V_R - B\,V_I, \qquad I_I = G\,V_I + B\,V_R. \tag{5}$$

Eq. (4) is enforced by adding Y_{load} to the diagonal of the load-bus admittance, and the augmented matrix is rebuilt after each topology or parameter change using the same rule.

The SPICE netlist of a reusable *subcircuit* realizes Eq. (5), is described below (a current limiter is also added to suppress large initialization spikes):

```
.SUBCKT ConstantImpedance V_r V_i PARAMS: P0=0.5
    Q0=0.0 V0=1.0 CurrLim=1000
.PARAM G_eq={P0/(V0*V0)} B_eq={-Q0/(V0*V0)}
VammR ammR 0 0V
VammI ammI 0 0V
BloadR V_r ammR I={limit(G_eq*V(V_r) - B_eq*V(V_i),
    -CurrLim, CurrLim)}
BloadI V_i ammI I={limit(B_eq*V(V_r) + G_eq*V(V_i),
    -CurrLim, CurrLim)}
.ENDS
```

A higher-level netlist instantiates the CI load as:

```
Xload5 bus5R bus5I ConstantImpedance PARAMS: P0=0.9
    Q0=0.3 V0=1.0127
```

Our tool provides a feature called "staged parametric network model (SPM)", by which the network admittances are preserved across each simulation stage, ensuring that only the intended topology or fault-impedance changes are considered in each corresponding stage during simulation.

III. Automatic Script Generation and GUI

This section details the workflow of the proposed tool that can automatically convert external raw input data into Xyce's (SPICE format) netlists and then execute transient simulations from a Qucs−S front end (GUI). The workflow is implemented via a single MATLAB driver, which imports raw input data, solves the base power flow, initializes dynamic states, *generates* the netlist, and schedules disturbance events. Thus, the workflow *eliminates* the need for manual efforts to prepare the SPICE netlist for Xyce.

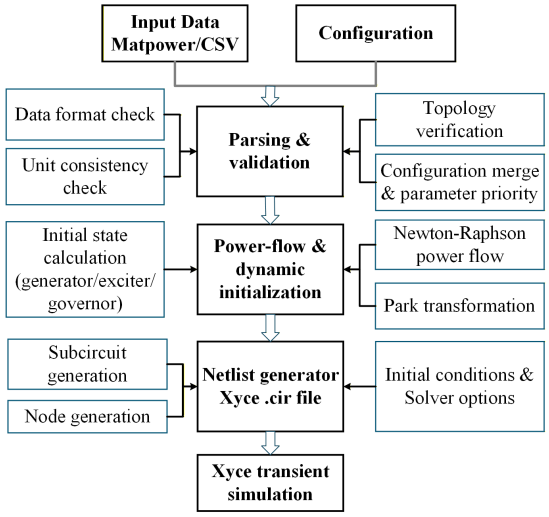


Fig. 2: Input and netlist generation workflow of the system

The input layer can read files in MATPOWER format or CSV format. The generation layer assembles a netlist `.cir` file with subcircuit instances, parameter blocks, event schedulers, and output directives. Qucs−S then loads and executes the netlist for transient simulation (by calling Xyce from backend) and plots selected signals from simulation outputs.

A. Library Organization

Following a similar modeling framework in [5], this work develops a dedicated SPICE library for transient simulation studies. The library encapsulates the following components:

- **Generators**: 2nd/3rd/4th/6th-order models with soft limiters, regression forces, low-damping optimizations
- **Exciters**: IEEE T1, AC1A with anti-windup, etc.
- **Loads**: ZIP, CPL, CI, Exponential, etc.
- **Network**: Lines, Transformers with staged switching
- **Numerical**: Limiters, deadbands, etc.

For example, the 2nd-order synchronous generator model implemented in the library reads:

$$\begin{cases} \dfrac{\mathrm{d}\delta}{\mathrm{d}t} = \omega_{\mathrm{b}}\,(\omega - \omega_{\mathrm{s}}), \\ \dfrac{2H}{\omega_{\mathrm{s}}}\,\dfrac{\mathrm{d}\omega}{\mathrm{d}t} = P_{\mathrm{m}} - P_{\mathrm{e}} - D\,(\omega - \omega_{\mathrm{s}}), \\ P_{\mathrm{e}} = \dfrac{|V|\,|E'|}{X'_d}\,\sin\!\big(\delta - \theta_V\big), \end{cases} \tag{6}$$

and the fourth-order model in the dq reference frame reads:

$$\begin{cases} \dot{\delta} = \omega_{\mathrm{b}}\,(\omega - \omega_{\mathrm{s}}), \\ \dfrac{2H}{\omega_{\mathrm{s}}}\,\dot{\omega} = P_{\mathrm{m}} - P_{\mathrm{e}} - D\,(\omega - \omega_{\mathrm{s}}), \\ \dot{E}'_q = \dfrac{E_{\mathrm{f}} - E'_q - (X_d - X'_d)\,i_d}{T'_d}, \\ \dot{E}'_d = \dfrac{-E'_d + (X_q - X'_q)\,i_q}{T'_q}. \end{cases} \tag{7}$$

The generated netlist relies on three reusable component libraries for network primitives, control blocks, and machine and load models, which are included at the top of the netlist:

```
.INCLUDE "powerGridModels.lib"
.INCLUDE "controlModels.lib"
.INCLUDE "machineModels.lib"
```

B. Configuration and Data Handling

The simulation configuration is set in a JSON file that can be processed by our tool based on a default template. The model type, inertia H, damping D, reactances can be overridden for each generator entry. A data module loads the raw input data/files and conduct unit detection (MW vs. p.u.) based on total load magnitude. The "parameter-overridden" happens based on the following priority order:

$$P_{\text{final}} = \text{merge}(P_{\text{per-gen}} > P_{\text{user-json}} > P_{\text{default}}), \tag{8}$$

where $P_{\text{per-gen}} > P_{\text{user-json}} > P_{\text{default}}$ denotes priority order, with per-generator entries overriding user JSON defaults

In command-line mode, the tool is invoked from MATLAB:

```
RawData_to_XyceGen() % Default configuration
RawData_to_XyceGen('XXXX.json') % file specified
```

Disturbances are configured via JSON entries. For example, to set a bus-fault at Bus 16 from 1s to 1.1s:

```
"disturbance": {
  "type": "bus fault",
  "location": 16,
  "start_time": 1.0,
  "duration": 0.1,
  "fault_resistance": 0.00001
}
```

Changing `"type"` to `"line_trip"` reconfigures the simulation without regenerating templates. The GUI mode will be introduced in Section E.

C. Initialization based on Power Flow Results

The power flow is solved by a Newton–Raphson scheme with PV–PQ switching, residual-based damping and adaptive steps. At iteration k:

$$\mathbf{J}(\mathbf{x}^{(k)})\,\Delta\mathbf{x}^{(k)} = -\mathbf{F}(\mathbf{x}^{(k)}), \mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \alpha^{(k)}\Delta\mathbf{x}^{(k)} \tag{9}$$

where $\alpha^{(k)} \in (0, 1]$ is a self-adjusting damping factor chosen by a short line search on the mismatch norm.

The converged power-flow solution $(P_0, Q_0, |V_0|, \theta_0)$ provides initial quantities for each generator's terminal bus. Terminal voltage and current are then computed as:

$$V = |V_0|e^{j\theta_0}, \qquad I = \frac{(P_g - jQ_g)}{V^*}, \tag{10}$$

where P_g and Q_g denote the active and reactive power injected by the generator at its terminal bus.

To interface with the machine reference frame, the Park transformation maps stator quantities onto the d–q axes:

$$\begin{bmatrix} V_d \\ V_q \end{bmatrix} = \begin{bmatrix} \sin\delta & -\cos\delta \\ \cos\delta & \sin\delta \end{bmatrix} \begin{bmatrix} V_R \\ V_I \end{bmatrix}, \tag{11}$$

where δ is the rotor angle. Then, we can compute:

$$E'_q = V_q + R_a I_q + X'_d I_d, \qquad E'_d = -V_d - R_a I_d + X'_q I_q. \tag{12}$$

Finally, the initial rotor angle and speed deviation are:

$$\delta_0 = \angle\big(V + (R_a + jX'_d)I\big), \qquad \omega_0 = 0. \tag{13}$$

All steady-state quantities $(E'_q, E'_d, \delta_0, \omega_0)$ are exported as `.PARAM` statements to initialize dynamic simulations.

D. Netlist Generation and Event Modeling

The netlist generator writes system bases, library header files, transient-control directives, and default solver options. Machine constants and initial conditions are encoded as `.PARAM` statements in the generated netlist.

Network components (branch, transformer, shunt) are defined by using several primitives: `YPGBR` (transmission lines), `YPGTR` (transformers), `YPGBS` (shunt admittances). Branch admittance is parameterized in time:

$$Y_\ell(t) = \frac{1}{R_\ell(t) + jX_\ell(t)} + j\frac{B_\ell(t)}{2}, \tag{14}$$

where branch parameters read as:

$$\begin{bmatrix} R_\ell(t) \\ X_\ell(t) \end{bmatrix} = \begin{bmatrix} R_{\text{pre}} \\ X_{\text{pre}} \end{bmatrix} + c(t)\left(\begin{bmatrix} R_{\text{open}} \\ X_{\text{open}} \end{bmatrix} - \begin{bmatrix} R_{\text{pre}} \\ X_{\text{pre}} \end{bmatrix}\right). \tag{15}$$

$$c(t) = \begin{cases} 0, & t < t_1, \\ \frac{t - t_1}{t_2 - t_1}, & t_1 \le t < t_2, \\ 1, & t \ge t_2, \end{cases} \tag{16}$$

Topology-changing events utilize the SPM, with bus faults being represented by time-varying ground resistance. Load models are integrated into the nodal admittance matrix and updated automatically. In practice, the ramp intervals for those changes are set much shorter than the global time scale, with almost no effect on the final results in the tested cases.

E. GUI Integration and Output Visualization

The tool operates in two modes: command-line and graphical (MATLAB GUI). The GUI exposes configuration via five tabs (System Parameters, Generator Config, Disturbance Settings, Simulation Parameters, Load & Network), and can *automatically* generate all needed files (JSON and SPICE netlist) with a single click (Fig. 3) in the background.

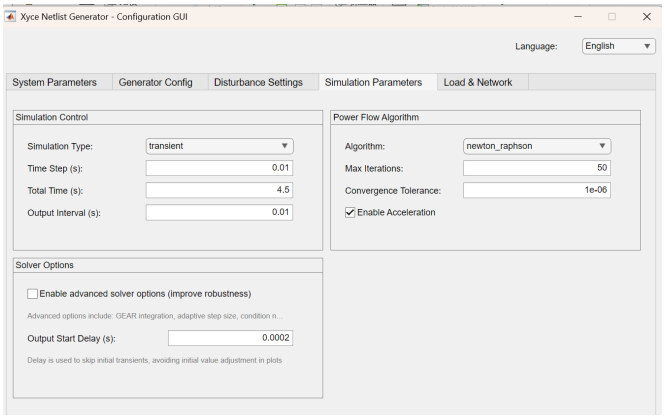


Fig. 3: Config GUI

The generated JSON and netlist files are then routed to Qucs–S, where Xyce is set as the simulation engine. After the simulation is done, Qucs–S loads the output and plots the results (e.g., bus phasors and generators' angle/speed). An example plot in Qucs–S GUI is displayed in Fig. 4.

Fig. 4: Qucs-S GUI

IV. Case Study

To validate the performance of our tool, we conduct transient simulations on the IEEE 9-bus system (cf. Fig. 5).

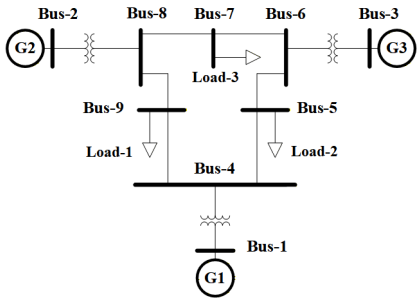


Fig. 5: IEEE 9 bus

A. Simulation Settings

Simulation settings and partial system parameters are summarized in Tables I–III.

TABLE I: Disturbance Types and Scheduling

Type	**Element**	$t_{\text{open/start}}$	$t_{\text{close/clear}}$
Bus fault	Bus 2	0.20 s	0.40 s
Line trip–reclose	Line 7–8	0.20 s	0.40 s

TABLE II: System Configuration

Item	**Setting**
Generator model	Classical 2nd-order (no exciter)
Load model	Constant-impedance
Lines	Series (R, X) only; line shunt $B = 0$
Transformers	Tap ratios included
Shunt elements	Nonzero (G_s, B_s)

TABLE III: Generator Parameters

Bus	**Role**	H (s)	D	X'_d (p.u.)	R_a (p.u.)
1	Slack	1.20	1.0	0.14	0
2	PV	2.40	1.0	0.14	0
3	PV	2.40	1.0	0.14	0

B. Simulation Results and Comparisons

Two disturbance scenarios are considered: a bus fault event and a line trip–reclosing event. The simulation results are compared with MatDyn. The mismatches mainly arise from implementation differences between Xyce and MatDyn, including the difference between the C++ and MATLAB languages, the distinct DAE formulations, and varying numerical integration techniques, as well as subtle differences in load and network admittances.

It can be observed that Xyce and MatDyn produce closely aligned responses under both bus fault (Fig. 6) and line-reclosing (Fig. 7) disturbances.

To quantify the mismatch between the two simulators, the simulation horizon is sampled at 100 uniformly spaced time points. At each time point, pointwise mismatches $|\Delta\omega|$, $|\Delta V|$, and $|\Delta\delta|$ are computed for each generator or bus. For each quantity, the maximum error over time is taken per generator or bus, which are reported in Table IV under each scenario.

TABLE IV: Global Mismatch Summary

Scenario	Bus.	$\max\|\Delta\omega\|$ [p.u.]	$\max\|\Delta\delta\|$ [rad]	$\max\|\Delta V\|$ [p.u.]
Bus fault	G2	3.0×10^{-4}	4.2×10^{-2}	–
	G3	1.0×10^{-4}	4.3×10^{-2}	–
	All	–	–	1.3×10^{-2}
Line reclose	G2	1.3×10^{-3}	9.6×10^{-2}	–
	G3	1.1×10^{-3}	9.5×10^{-2}	–
	All	–	–	9.7×10^{-2}

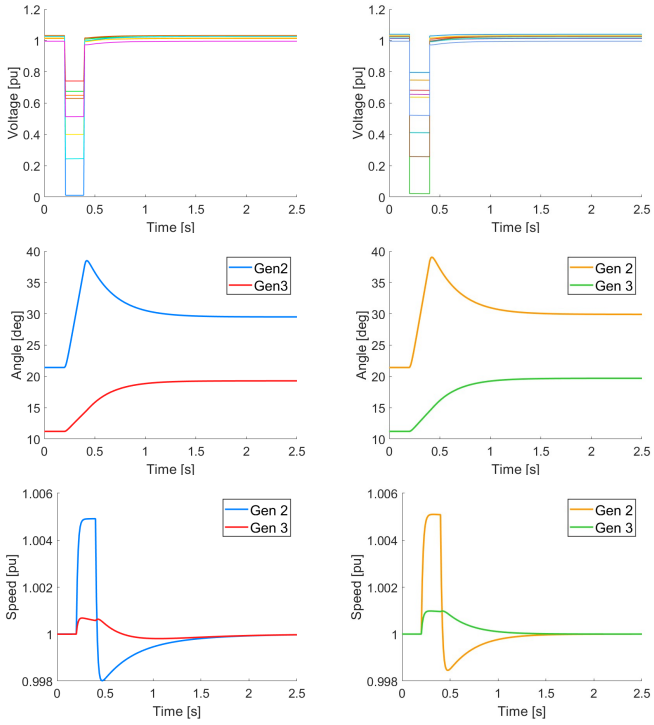


Fig. 6: Bus-fault case: Xyce vs. MatDyn

Note. Left column: Xyce; right column: MatDyn. Top row: bus-voltage magnitude; middle: rotor angle; bottom: speed.

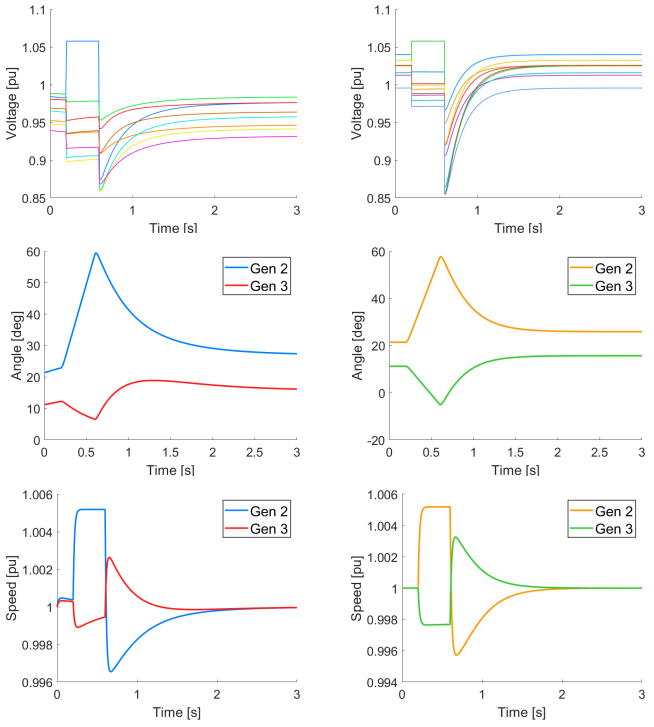


Fig. 7: Line-reclosing case: Xyce vs. MatDyn

Note. Left column: Xyce; right column: MatDyn. Top row: bus-voltage magnitude; middle: rotor angle; bottom: speed.

The standard deviation over the last 10% of each waveform further characterizes the steady-state behavior. The corresponding metrics are summarized in Table V, indicating that both simulators converge to almost the same post-fault operating point.

TABLE V: Steady–state Convergence Metric

Scenario	σ_V	σ_ω	σ_δ
Bus fault	6.39×10^{-5}	4.74×10^{-6}	4.07×10^{-5}
Line reclose	6.01×10^{-5}	3.54×10^{-6}	4.82×10^{-3}

The wall-clock time costs are compared using the median of ten cold-start executions (Table VI). All scenarios use identical event schedules and a relative tolerance of 10^{-4}.

TABLE VI: Comparison of the Median Wall-Clock Time Costs

Scenario	**MatDyn [s]**	**Our Tool [s]**	**Speedup**
Bus fault	4.41	**0.3420**	**12.89** $\times$
Line trip	5.57	**0.4605**	**12.09** $\times$
Line reclose	2.55	**0.1632**	**15.62** $\times$

Our tool achieves a 12–15$\times$ speedup while obtaining aligned simulation curves as MatDyn, under identical disturbances, load assumptions, and numerical tolerances. This confirms both the efficiency and the numerical consistency of the proposed approach.

V. Conclusion

In this paper, an end-to-end, Xyce-based tool for power system transient simulation is developed, featuring automatic netlist generation and consistent initialization from power-flow data. Benchmark tests on the IEEE 9-bus system demonstrate close agreement with MatDyn in waveforms, together with an order-of-magnitude speedup for the tested scenarios. Future work will extend the library to renewable models and apply the workflow to large-scale multi-area systems.

References

[1] P. Kundur, *et al.*, "Definition and classification of power system stability IEEE/CIGRE joint task force on stability terms and definitions," *IEEE Trans. Power Syst.*, vol. 19, no. 3, pp. 1387–1401, 2004. , doi: 10.1109/TPWRS.2004.825981.

[2] 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, doi: 10.1109/TPWRS.2010.2071888.

[3] 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. , doi: 10.1109/TPWRS.2010.2051168.

[4] Application Note: Power Grid Modeling With Xyce™.

[5] Y. Zhu, X. Zhang and R. Dai, "Power System Transient Analysis via Circuit Simulator: A Case Study of SVC," *2021 IEEE Kansas Power and Energy Conference (KPEC)*, Manhattan, KS, USA, 2021, pp. 1-6. , doi: 10.1109/KPEC51835.2021.9446258.

[6] M. Brinson and V. Kuznetsov, "Qucs-0.0.19S: A new open-source circuit simulator and its application for hardware design," *2016 International Siberian Conference on Control and Communications (SIBCON)*, Moscow, Russia, 2016, pp. 1-5, doi: 10.1109/SIBCON.2016.7491696.