
powerSAS.m

Release 0.0.0

Apr 25, 2023

POWERSAS.M MANUAL

1	PowerSAS.m	1
1.1	Key features	1
1.1.1	About	1
1.1.2	Using PowerSAS.m: The Basics	2
1.1.3	Advanced Usage	6
1.1.4	Data and Models	15
1.1.5	Installation	15
1.1.6	SAS and PowerSAS.m: The Story	16
2	Indices and tables	21

POWERSAS.M

PowerSAS.m is a robust, efficient and scalable power grid analysis framework based on semi-analytical solutions (SAS) technology. The **PowerSAS.m** is the version for MATLAB/Octave users. It currently provides the following functionalities (more coming soon!):

- **Steady-state analysis**, including power flow (PF), continuation power flow (CPF), contingency analysis.
- **Dynamic security analysis**, including voltage stability analysis, transient stability analysis, and flexible user-defined simulation.
- **Hybrid extended-term simulation** provides adaptive QSS-dynamic hybrid simulation in extended term with high accuracy and efficiency.

1.1 Key features

- **High numerical robustness.** Backed by the SAS approach, the PowerSAS tool provides much better convergence than the tools using traditional Newton-type algebraic equation solvers when solving algebraic equations (AE)/ordinary differential equations (ODE)/differential-algebraic equations (DAE).
- **Enhanced computational performance.** Due to the analytical nature, PowerSAS provides model-adaptive high-accuracy approximation, which brings significantly extended effective range and much larger steps for steady-state/dynamic analysis. PowerSAS has been used to solve large-scale system cases with 200,000+ buses.
- **Customizable and extensible.** PowerSAS supports flexible customization of grid analysis scenarios, including complex event sequences in extended simulation term.

1.1.1 About

Contributors

- Rui Yao, Argonne National Laboratory <ryao@anl.gov>
- Feng Qiu, Argonne National Laboratory <fqiu@anl.gov>
- Kai Sun, University of Tennessee, Knoxville <kaisun@utk.edu>
- Jianzhe Liu, Argonne National Laboratory <jianzhe.liu@ieee.org>

Acknowledgement

This work is supported by the Laboratory Directed Research and Development (LDRD) program of Argonne National Laboratory, provided by the U.S. Department of Energy Office of Science under Contract No. DE-AC02-06CH11357, and the U.S. Department of Energy Office of Electricity, Advanced Grid Modeling program under Grant DE-OE0000875.

Publications

- Rui Yao, Kai Sun, Feng Qiu, “Vectorized Efficient Computation of Padé Approximation for Semi-Analytical Simulation of Large-Scale Power Systems,” IEEE Transactions on Power Systems, 34 (5), 3957-3959, 2019.
- Rui Yao, Yang Liu, Kai Sun, Feng Qiu, Jianhui Wang, “Efficient and Robust Dynamic Simulation of Power Systems with Holomorphic Embedding”, IEEE Transactions on Power Systems, 35 (2), 938 - 949, 2020.
- Rui Yao, Feng Qiu, “Novel AC Distribution Factor for Efficient Outage Analysis”, IEEE Transactions on Power Systems, 35 (6), 4960-4963, 2020.
- Xin Xu, Rui Yao, Kai Sun, Feng Qiu, “A Semi-Analytical Solution Approach for Solving Constant-Coefficient First-Order Partial Differential Equations”, IEEE Control Systems Letters, 6, 704-709, 2021.
- Rui Yao, Feng Qiu, Kai Sun, “Contingency Analysis Based on Partitioned and Parallel Holomorphic Embedding”, IEEE Transactions on Power Systems, in press.

License

This software is under BSD license. Please refer to LICENSE.md for details.

1.1.2 Using PowerSAS.m: The Basics

1 Initialization before use

To initialize PowerSAS.m, add the main directory of PowerSAS to Matlab or GNU Octave path, and execute command `initpowersas`. This will ensure all the functions of PowerSAS be added to the path and thus callable.

2 Call high-level API – runPowerSAS

Most grid analysis functionalities can be invoked through the high-level function `runPowerSAS`. `runPowerSAS` is defined as follows:

```
function res=runPowerSAS(simType,data,options,varargin)
```

Input arguments

- `simType` is a string specifying the type of analysis, which can be one of the following values:
 - `'pf'`: Conventional power flow or extended power flow (for finding steady state of dynamic model).
 - `'cpf'`: Continuation power flow.
 - `'ctg'`: Contingency analysis (line outages).
 - `'n-1'`: N-1 line outage screening.
 - `'tsa'`: Transient stability analysis.
 - `'dyn'`: General dynamic simulation.
- `data` is the system data to be analyzed. It can be either a string specifying the data file name, or a `SysData` struct. For more information about data format and `SysData` struct, please refer to the “Data format and models” chapter.
- `options` specifies the options for analysis. If you do not provide `options` argument, or if you simply set the field to empty with `[]`, the corresponding routines will provide default options that will fit most cases. See Advanced Use chapter for more details.
- `varargin` are the additional input variables depending on the type of analysis. Section 3 Basic analysis functionalities will explain more details.

Output

Output `res` is a struct containing simulation result, system states, system data, etc. * `res.flag`: Flag information returned by the analysis task. * `res.msg`: More information as supplemental to the flag information. * `res.caseName`: The name of the analyzed case. * `res.timestamp`: A string showing the timestamp the analysis started, can be viewed as an unique identifier of the analysis task. * `res.stateCurve`: A matrix storing the evolution of system states, where the number of rows equals the number of state variables, and the number of columns equals the number of time points. * `res.t`: A vector storing time points corresponding to states in `res.stateCurve`. * `res.simSettings`: A struct specifying the simulation settings, including simulation parameters and defined events. * `res.eventList`: A matrix showing as the list of events in the system in the analysis task. * `res.SysDataBase`: A struct of system data at base state. * `res.snapshot`: The snapshot of the system states at the end of analysis, which can be used to initialize other analysis tasks.

To access the system states, we need to further access each kind of state variable in `res.stateCurve`. For example, the commands to extract the voltage from `res.stateCurve` are shown below:

```
[~,idxs]=getIndexDyn(res.SysDataBase); % Get the indexes of each kind of state_  
↳variables  
vCurve=res.StateCurve(idx.vIdx,:); % idx.vIdx is the row indexes of voltage_  
↳variables
```

3 Basic analysis functionalities

When `simType='pf'`, the `runPowerSAS` function runs power flow analysis. In addition to the conventional power flow model, `runPowerSAS` also integrates an extended power flow to solve the steady state of dynamic models. For example, it will calculate the rotor angles of synchronous generators and slips of induction motors in addition to the network equations.

To perform power flow analysis, call the `runPowerSAS` function as follows:

```
res=runPowerSAS('pf',data,options)
```

where the argument `data` can either be a string of file name or a `SysData` struct.

Below are some examples:

```
% Use file name string to specify data  
res1=runPowerSAS('pf','d:/codes/d_003.m'); % Filename can use absolute path  
res2=runPowerSAS('pf','d_003.m'); % If data file is already in the Matlab/Octave_  
↳path,  
                                     % then can directly use file name  
res3=runPowerSAS('pf','d_003'); % Filename can be without '.m'  
res4=runPowerSAS('pf','d_003',setOptions('dataPath','d:/codes')); % Another way_  
↳to specify data path  
  
% Use SysData struct to specify data  
SysData=readDataFile('d_003.m','d:/codes'); % Generate SysData struct from data_  
↳file  
res5=runPowerSAS('pf',SysData); % Run power flow using SysData struct
```

Continuation power flow (CPF) analysis in PowerSAS.m features enhanced efficiency and convergence. To perform continuation power flow analysis, call `runPowerSAS` function as follows:

```
res=runPowerSAS('cpf',data,options,varargin)
```

where `options` (optional) specifies the options of CPF analysis, and `varargin` are the input arguments: * `varargin{1}` (optional) is the ramping direction of load, which is an $N \times 12$ matrix, the first column is the index of the bus, and the columns 5-10 are the ZIP load increase directions. * `varargin{2}` (optional) is the ramping direction of generation power, which is an $N \times 2$ matrix, the first column is the index of the bus, and the 2nd column is the generation increase directions. * `varargin{3}` (optional) is the snapshot of the starting state, with which the computation of starting steady state is skipped.

Some examples can be found in `example/ex_cpf.m`.

Contingency analysis computes the system states immediately after removing a line/lines. To perform contingency analysis, call `runPowerSAS` as follows:

```
res=runPowerSAS('ctg',data,options,varargin)
```

where `options` (optional) specifies the options of contingency analysis. When not using customized options, set `options=[]`. And `varargin` are the input arguments: * `varargin{1}` (mandatory) is a vector specifying the indexes of lines to be removed simultaneously. * `varargin{2}` (optional) is the snapshot of the starting state. With this option, computing the starting steady state is skipped.

Some examples can be found in `example/ex_ctg.m`.

N-1 screening is essentially performing a series of contingency analysis, each removing a line from the base state. To perform N-1 screening, call `runPowerSAS` as follows:

```
res=runPowerSAS('n-1',data,options)
```

The return value `res` is a cell containing each contingency analysis results.

Some examples can be found in `example/ex_n_minus_1.m`.

Transient stability analysis (TSA) assesses the system dynamic behavior and stability after given disturbance(s). 3-phase balanced fault(s) are the most common disturbances in the TSA. In PowerSAS, the TSA supports the analysis of the combinations of multiple faults. To perform transient Stability Analysis, call `runPowerSAS` in the following way:

```
res=runPowerSAS('tsa',data,options,varargin)
```

where `options` (optional) specifies the options of TSA. When not using customized options, set `options=[]`. And `varargin` are the input arguments: * `varargin{1}` (mandatory) is a $N \times 6$ matrix specifying the faults: * The 1st column is the index of line where the fault happens. * The 2nd column is the relative position of the fault, 0.0 stands for the starting terminal and 1.0 stands for the ending terminal. For example, 0.5 means the fault happens in the middle point of the line. * The 3rd and 4th columns are the resistance and reactance of the fault. * The 5th and 6th columns specify the fault occurrence and clearing times. * `varargin{2}` (optional) is the snapshot of the starting state, with which the computation of starting steady state is skipped.

By default, the TSA is run for 10 seconds. To change the simulation length, specify in the `options` argument, e.g. `options=setOptions('simlen',30)`.

Example can be found in `example/ex_tsa.m`.

4. Plot dynamic analysis results

PowerSAS provides an integrated and formatted way of plotting the system behavior in the time domain. The function for plotting curves is `plotCurves`. The function is defined as follows:

```
function plotCurves(figId,t,stateCurve,SysDataBase,variable,...
                    subIdx,lw,dot,fontSize,width,height,sizeRatio)
```

The argument list is explained as follows: * **figId**: A positive integer or [] specifying the index of figure. * **t**: A vector of time instants. If got **res** from **runPowreSAS** function, then input this argument as **res.t**. * **stateCurve**: A matrix of system states in time domain, the number of columns should equal to the length of **t**. If got **res** from **runPowreSAS** function, then input this argument as **res.stateCurve**. * **SysDataBase**: A SysData struct specifying the base system. If got **res** from **runPowreSAS** function, then input this argument as **res.SysDataBase**. * **variable**: A string of variable name to be plotted. Here is a nonexhaustive list: * **'v'**: voltage magnitude (pu); * **'a'**: voltage angle (rad); * **'delta'**: rotor angle of synchronous generators; * **'omega'**: deviation of synchronous generator rotor speed; * **'s'**: induction motor slips; * **'f'**: frequency; * **subIdx**: Allows you to pick a portion of the variables to plot e.g., the voltage of some selected buses. Default value is [], which means that all the selected type of variables are plotted. * **lw**: Line width. Default value is 1. * **dot**: Allows you to choose whether to show data points. 1 means curves mark data dots, and 0 means no data dots are shown on curves. The default value is 0. * **fontSize**: Font size of labels. Default value is 12. * **width**: Width of figure window in pixels. * **height**: Height of figure window in pixels. * **sizeRatio**: If **width** or **height** is not specified, the size of the figure is determined by the **sizeRatio** of the screen size. The default value of **sizeRatio** is 0.7.

1.1.3 Advanced Usage

1. Customize analysis with settings file/struct

PowerSAS.m lets you customize your simulation by providing a simulation settings interface to specify the events and scenarios to be analyzed. To use the customized simulation, call the **runPowerSAS** function as follows:

```
res=runPowerSAS('dyn',data,options,settings,snapshot)
```

Details are explained as follows: ##### 1.1 Settings file The input argument **settings** can be a string specifying the settings file name, or a struct containing all the simulation settings.

When **settings** is a string, it should be a valid file name of an .m script file containing the settings. Some examples of the settings files can be found in the directory /data. The settings file should have the following variables:

- **eventList**: A gross list of events. ([more details](#))
- **bsBus**: (for black start simulation only) Black start bus information. ([more details](#))
- **bsSyn**: (for black start simulation only) Generator information on black start bus. ([more details](#))
- **bsInd**: (for black start simulation only) Inductin motor on black start bus. ([more details](#))
- **Efstd**: Excitation potential of synchronous generators. ([more details](#))
- **evtLine**: List of line addition/outage events. ([more details](#))
- **evtLineSpec**: Specifications of line addition/outage events. ([more details](#))
- **evtZip**: List of static load addition/shedding events. ([more details](#))
- **evtZipSpec**: Specifications of static load addition/shedding events. ([more details](#))
- **evtZipSpec2**: Alternative specifications of static load addition/shedding events. ([more details](#))

- `evtInd`: List of induction motor addition/outage events. ([more details](#))
- `evtIndSpec`: Specifications of induction motor addition/outage events. ([more details](#))
- `evtSyn`: List of synchronous generator addition/outage events. ([more details](#))
- `evtSynSpec`: Specifications of synchronous generator addition/outage events. ([more details](#))
- `evtFault`: List of fault occurrence/clearing events. ([more details](#))
- `evtFaultSpec`: Specifications of fault occurrence/clearing events. ([more details](#))
- `evtDyn`: List of dynamic ramping events. ([more details](#))
- `evtDynPQ`: Specifications of PQ bus ramping. ([more details](#))
- `evtDynPV`: Specifications of PV bus ramping. ([more details](#))
- `evtDynInd`: Specifications of induction motor mechanical load ramping. ([more details](#))
- `evtDynZip`: Specifications of ZIP load ramping. ([more details](#))
- `evtDynSh`: Specifications of shunt compensator ramping. ([more details](#))
- `evtDynZipRamp`: Alternative specifications of ZIP load ramping. ([more details](#))
- `evtDynTmech`: Specifications of generator mechanical torque ramping. ([more details](#))
- `evtDynPm`: Specifications of generator input active power ramping. ([more details](#))
- `evtDynEf`: Specifications of generator excitation potential ramping. ([more details](#))
- `evtDynVref`: Specifications of exciter reference voltage ramping. ([more details](#))
- `evtDynEq1`: Specifications of generator transient excitation potential ramping. ([more details](#))

1.2 Settings struct

Alternatively, the `settings` can be a struct containing all the previous variables as its fields.

Example 1: Transient stability analysis (TSA) using settings file

In this example, we want to perform transient stability analysis on 2383-bus system. Here is the scenario of the TSA: * The total simulation period is 10 seconds. * At 0.5 s, apply faults on the starting terminals of lines 74 and 114, the fault resistance is 0.0 and the reactance is 0.02. At 0.75 s, clear the faults. * At 1.5 s, apply fault on the starting terminal of line 1674, the fault resistance is 0.0 and the reactance is 0.1. At 1.95 s, clear the faults.

The settings file for this simulation is shown below. Other variables irrelevant to the fault events are omitted here for the sake of clarity.

```
eventList=[...
    1      0.0000  0.0000    0    1 0.0  0.0000
    1.1    0.5000  0.0000    6    1 0.0  0.0000
```

(continues on next page)

(continued from previous page)

```

1.2  0.7500  0.0000    6    2 0.0  0.0000
1.3  1.5000  0.0000    6    3 0.0  0.0000
1.4  1.9500  0.0000    6    4 0.0  0.0000
3    10.00   0.0000   99    0 0.0  0.0000
];

% Fault event data
evtFault=[...
    1    1    2
    2    3    4
    3    5    5
    4    6    6
];

evtFaultSpec=[...
    114,  0.00, 0, 0.02,  0;
    74,  0.00, 0, 0.02,  0;
    114,  0.00, 0, 0.02,  1;
    74,  0.00, 0, 0.02,  1;
   1674,  0.00, 0, 0.1,   0;
   1674,  0.00, 0, 0.1,   1;
];

```

Assume the settings file is `settings_polilsh_tsa.m` and the system data file is `d_dcase2383wp_mod2_ind_zip_syn.m`. We can run the TSA as follows:

```

res_2383_st=runPowerSAS('pf','d_dcase2383wp_mod2_ind_zip_syn.m'); % Run steady-
↳state
res_2383_tsa=runPowerSAS('dyn','d_dcase2383wp_mod2_ind_zip_syn.m',setOptions(
↳'hotStart',1),'settings_polilsh_tsa',res_2383_st.snapshot); % Hot start from
↳existing steady-state

plotCurves(1,res_2383_tsa.t,res_2383_tsa.stateCurve,res_2383_tsa.SysDataBase,'v
↳'); % plot the voltage magnitude curves

```

2. Extended-term simulation using hybrid QSS and dynamic engines

To accelerate computation — especially for extended-term simulation — PowerSAS.m provides an adaptive way to switch between QSS and dynamic engines in the course of a simulation. With this feature enabled, PowerSAS.m can switch to QSS simulation for better speed on detecting the fade-away of transients and switch back to dynamic simulation upon detecting transient events.

For more details on the technical approach, please refer to our paper: * Hybrid QSS and Dynamic Extended-Term Simulation Based on Holomorphic Embedding, arXiv:2104.02877

Example 2 illustrates the use of PowerSAS.m to perform extended-term simulation.

Example 2: Extended-term simulation

We want to study the response of a 4-bus system under periodic disturbances. The entire simulated process is 500 seconds. Starting at 60 s and continuing until 270 s, the system undergoes events of adding/shedding loads every 30 s.

The key settings of the simulation are:

```
% settings_d_004_2a_agc.m

eventList=[...
    1    0.0000    0.0000    0    1 0.0    0.0100
    6   60.0000    0.0000    2    1 0.0    0.0100
    7   90.0000    0.0000    2    2 0.0    0.0100
    8  120.0000    0.0000    2    3 0.0    0.0100
    9  150.0000    0.0000    2    4 0.0    0.0100
   10  180.0000    0.0000    2    1 0.0    0.0100
   11  210.0000    0.0000    2    2 0.0    0.0100
   12  240.0000    0.0000    2    3 0.0    0.0100
   13  270.0000    0.0000    2    4 0.0    0.0100
   18  500.0000    0.0000   99    0 0.0    0.0100
];

% Static load event data
evtZip=[...
    1    1    1    1
    2    1    2    2
    3    1    3    3
    4    1    4    4
];

evtZipSpec2=[...
    3 100.0000 100.0000 60.0000  0.0648  0.0648  0.0648  0.0359  0.0359  0.0359  ↵
↵ 0    1
    2 100.0000 100.0000 60.0000  0.0648  0.0648  0.0648  0.0359  0.0359  0.0359  ↵
↵ 0    1
    3 100.0000 100.0000 60.0000 -0.0648 -0.0648 -0.0648 -0.0359 -0.0359 -0.0359  ↵
↵ 0    1
    2 100.0000 100.0000 60.0000 -0.0648 -0.0648 -0.0648 -0.0359 -0.0359 -0.0359  ↵
↵ 0    1
];
```

First we run the simulation in full-dynamic mode and record time:

```
% Full dynamic simulation
tagFullDynStart=tic;
res_004_fulldyn=runPowerSAS('dyn','d_004_2a_bs_agc.m',[],'settings_d_004_2a_agc
```

(continues on next page)

(continued from previous page)

```
↪');
timeFullDyn=toc(tagFullDynStart);
```

Then we run the simulation in hybrid QSS & dynamic mode and record time:

```
% Hybrid simulation with dynamic-QSS switching
tagHybridStart=tic;
res_004=runPowerSAS('dyn','d_004_2a_bs_agc.m',setOptions('allowSteadyDynSwitch',
↪1),'settings_d_004_2a_agc');
timeHybrid=toc(tagHybridStart);
```

Compare the results:

```
plotCurves(1,res_004_fulldyn.t,res_004_fulldyn.stateCurve,res_004_fulldyn.
↪SysDataBase,'v');
plotCurves(2,res_004.t,res_004.stateCurve,res_004.SysDataBase,'v');
```

And compare the computation time:

```
disp(['Full dynamic simulation computation time:', num2str(timeFullDyn),' s.']);
disp(['Hybrid simulation computation time:', num2str(timeHybrid),' s.']);
```

The complete example can be found in /example/ex_extended_term_dyn.m. And the results can also be found in our paper: * Hybrid QSS and Dynamic Extended-Term Simulation Based on Holomorphic Embedding, arXiv:2104.02877

Appendix: Variables in settings

variable eventList

(back to top) ##### Table 1. Definition of eventList Column | Content ———| ————— 1 | Event index (can be an integer or a real number) 2 | Event start time 3 | Event end time (no effect for instant event) 4 | Type of event (see [Table 2](#)) 5 | Index of event under its type 6 | Simulation method (default 0.0) (see below [Simulation methods](#)) 7 | Timestep (default 0.01)

Table 2. Event types

(back to top) Value | Event type ———| ————— 0 | Calculate steady-state at start 1 | Add line 2 | Add static load 3 | Add induction motor load 4 | Add synchronous generator 6 | Applying/clearing faults 7 | Cut line 8 | Cut static load 9 | Cut motor load 10 | Cut synchronous generator 50 | Dynamic process 99 | End of simulation

Simulation methods

Simulation methods can be specified for each event on the 6th column of `eventList`. It is encoded as a number `x.yz`, where: * `x` is the method for solving differential equation, where 0 - SAS, 1 - Modified Euler, 2 - R-K 4, 3 - Trapezoidal rule. * `y` is the method for solving algebraic equation, where 0 - SAS, 1 - Newton-Raphson. * `z` is whether to use variable time step scheme for numerical integration (`x` is 1, 2 or 3). 0 - Fixed step, 1 - Variable step.

Note that when `x=0`, `y` and `z` are not effective, it automatically uses SAS and variable time steps.

variable `bsBus`

(*back to top*) Current version only support one black start bus and therefore only the first line will be recognized. Will expand in the future versions. Column | Content ———|————— 1 | Bus index 2 | Active power of Z component load 3 | Active power of I component load 4 | Active power of P component load 5 | Reactive power of Z component load 6 | Reactive power of I component load 7 | Reactive power of P component load

variable `bsSyn`

(*back to top*) Column | Content ———|————— 1 | Index of synchronous generator 2 | Excitation potential 3 | Active power 4 | Participation factor for power balancing

variable `bsInd`

(*back to top*) Column | Content ———|————— 1 | Index of induction motor 2 | Mechanical load torque

variable `Efstd`

(*back to top*) When there are synchronous generators in the system model, `Efstd` is needed to compute steady state. The `Efstd` is a column vector specifying the excitation potential of every synchronous generator, or it can also be a scalar assigning the excitation potentials of all the generator as the same value.

variables `evtLine` and `evtLineSpec`

(*back to top*) In `eventList`, when the 4th column (event type) equals 1 or 7 (add or cut line, respectively), the index of the line events in `evtLine` corresponds to the 5th column of `eventList`. ##### variable `evtLine`
(*back to top*) Column | Content ———|————— 1 | Index of line events (from 5th column of `eventList`) 2 | Start index in `evtLineSpec` 3 | End index in `evtLineSpec`

variable evtLineSpec

(back to top) Column | Content ———|————— 1 | Index of line 2 | Add/cut mark, 0 - add line, 1 - cut line
3 | Reserved 4 | Reserved 5 | Reserved

variables evtZip, evtZipSpec and evtZipSpec2

(back to top) In eventList, when the 4th column (event type) equals 2 or 8 (add/cut static load), the index of the load events in evtZip corresponds to the 5th column of eventList. ##### variable evtLine (back to top) Column | Content ———|————— 1 | Index of load events (from 5th column of eventList) 2 | Choose evtZipSpec (0) or evtZipSpec2 (1) 3 | Start index in evtZipSpec or evtZipSpec2 4 | End index in evtZipSpec or evtZipSpec2

variable evtZipSpec

(back to top) Column | Content ———|————— 1 | Index of zip loads in system base state 2 | Add/cut mark, 0 - add load, 1 - cut load

variable evtZipSpec2 (recommended)

(back to top) evtZipSpec2 has the same format as PSAT ZIP load format, which represents the change of ZIP load. Whether the event is specified as add/cut load does not make difference.

variables evtInd and evtIndSpec

(back to top) In eventList, when the 4th column (event type) equals 3 or 9 (add or cut induction motors, respectively), the index of the induction motor events in evtInd corresponds to the 5th column of eventList. ##### variable evtInd (back to top) Column | Content ———|————— 1 | Index of induction motor events (from 5th column of eventList) 2 | Start index in evtIndSpec 3 | End index in evtIndSpec

variable evtIndSpec

(back to top) Column | Content ———|————— 1 | Index of induction motor 2 | Event type, 0 - add motor, 1 - change state, 2 - cut motor 3 | Designated mechanical torque 4 | Designated slip

variables `evtSyn` and `evtSynSpec`

(back to top) In `eventList`, when the 4th column (event type) equals 4 or 10 (add or cut synchronous generators, respectively), the index of the synchronous generator events in `evtSyn` corresponds to the 5th column of `eventList`. ##### variable `evtSyn` Column | Content ———| ————— 1 | Index of synchronous generator events (from 5th column of `eventList`) 2 | Start index in `evtSynSpec` 3 | End index in `evtSynSpec`

variable `evtSynSpec`

(back to top) Column | Content ———| ————— 1 | Index of synchronous generator 2 | Event type, 0 - add generator, 1 - cut generator 3 | Designated rotor angle (only effective when adding generator, NaN means the rotor angle is the same with voltage angle). 4 | Designated mechanical power (only effective when adding generator). 5 | Designated excitation potential (only effective when adding generator).

variables `evtFault` and `evtFaultSpec`

(back to top) In `eventList`, when the 4th column (event type) equals 6 (apply or clear faults, respectively), the index of the fault events in `evtFault` corresponds to the 5th column of `eventList`. In the current version, we only consider three-phase grounding faults. ##### variable `evtFault` Column | Content ———| ————— 1 | Index of fault events (from 5th column of `eventList`) 2 | Start index in `evtFaultSpec` 3 | End index in `evtFaultSpec`

variable `evtFaultSpec`

(back to top) Column | Content ———| ————— 1 | Index of fault line 2 | Position of fault, 0.0 stands for starting terminal and 1.0 stands for ending terminal. 3 | Resistance of fault. 4 | Reactance of fault. 5 | Event type: 0 - add fault; 1 - clear fault.

variable `evtDyn`

(back to top) The `evtDyn` variable specifies the indexes of ramping events involving various types of components. Column | Content ———| ————— 1 | Index of event 2 | Start index in `evtDynPQ` 3 | End index in `evtDynPQ` 4 | Start index in `evtDynPV` 5 | End index in `evtDynPV` 6 | Start index in `evtDynInd` 7 | End index in `evtDynInd` 8 | Start index in `evtDynZip` 9 | End index in `evtDynZip` 10 | Start index in `evtDynSh` 11 | End index in `evtDynSh` 12 | Start index in `evtDynZipRamp` 13 | End index in `evtDynZipRamp` 14 | Start index in `evtDynTmech` 15 | End index in `evtDynTmech` 16 | Start index in `evtDynPm` 17 | End index in `evtDynPm` 18 | Start index in `evtDynEf` 19 | End index in `evtDynEf` 20 | Start index in `evtDynVref` 21 | End index in `evtDynVref` 22 | Start index in `evtDynEq1` 23 | End index in `evtDynEq1`

variable evtDynPQ

(back to top) Column | Content ———| ————— 1 | Index of bus 2 | Active power ramping rate 3 | Reactive power ramping rate

variable evtDynPV

(back to top) Column | Content ———| ————— 1 | Index of bus 2 | Active power ramping rate

variable evtDynInd

(back to top) Column | Content ———| ————— 1 | Index of induction motor 2 | Mechanical load torque ramping rate

variable evtDynZip

(back to top) Column | Content ———| ————— 1 | Index of bus 2 | ZIP load ramping rate

variable evtDynSh

(back to top) Column | Content ———| ————— 1 | Index of bus 2 | Shunt admittance ramping rate

variable evtDynZipRamp

(back to top) evtDynZipRamp has the same format as PSAT ZIP load format, which represents the ramping direction of ZIP load.

variable evtDynTmech

(back to top) Column | Content ———| ————— 1 | Index of synchronous generator 2 | Ramping rate of mechanical power reference value (TG required)

variable evtDynEf

(back to top) Column | Content ———| ————— 1 | Index of synchronous generator 2 | Ramping rate of excitation potential

variable evtDynVref

(back to top) Column | Content ———| ————— 1 | Index of synchronous generator 2 | Ramping rate of exciter reference voltage

variable evtDynEq1

(back to top) Column | Content ———| ————— 1 | Index of synchronous generator 2 | Ramping rate of transient excitation potential

1.1.4 Data and Models**1. Supported data formats**

Currently PowerSAS.m supports extended PSAT (Matlab) data format.

In addition, PSS/E data format can be easily converted to the PSAT data format using the `psse2mpc` function by MATPOWER and the `matpower2psat` function by PSAT. A sample code to convert a PSS/E **raw* file to the PSAT format is below:

```
casename = 'ieee14'
[mpc, warnings] = psse2mpc(horzcat(casename, '.raw'), horzcat(casename, '.m'));
check = matpower2psat(horzcat(casename, '.m'), pwd);
```

Support for other formats and data format conversion features will be added in future versions.

2. Extension of PSAT (Matlab) format**2.1 Automatic generation control (AGC) model**

Here PowerSAS provides a simple AGC model. It is named as `Agc.con` in data files and it is a $N \times 4$ matrix. Each column is defined as below: Column | Content ———| ————— 1 | Bus index 2 | Reciprocal of turbine governor gain on bus 3 | Effective damping ratio on bus 4 | Reciprocal of AGC control time constant

1.1.5 Installation**1. System requirements**

Matlab (7.1 or later) or GNU Octave (4.0.0 or later).

2. Installation

- Extract source code to a directory.
- Enter the directory in Matlab or GNU Octave.
- Execute command `setup`. You will see the following sub-directories:
 - `/data`: Stores test system data, simulation settings data, etc.
 - `/example`: Some examples of using PowerSAS.m.
 - `/output`: Stores test result data.
 - `/internal`: Internal functions of PowerSAS.m computation core.
 - `/util`: Auxiliary functions including data I/O, plotting, data conversion, etc.
 - `/logging`: Built-in logging system.
 - `/doc`: Documentation.

3. Test

- Execute command `initpowersas` to initialize the environment, then execute `test_powersas` to run tests. You should expect all tests to pass.

4. Initialization

To initialize PowerSAS.m, add the main directory of PowerSAS.m to your Matlab or GNU Octave path and run the command `initpowersas`. This will ensure that all the functions of PowerSAS.m are added to the path and thus callable.

1.1.6 SAS and PowerSAS.m: The Story

1. What are Semi-Analytical Solutions (SAS)?

Semi-analytical solutions (SAS) is a family of computational methods that uses certain analytical formulations (e.g., power series, fraction of power series, continued fractions) to approximate the solutions of mathematical problems. In terms of formulation, they are quite different from the commonly used numerical approaches e.g., Newton-Raphson method for solving algebraic equations, Runge-Kutta and Trapezoidal methods for solving differential equations. The parameters of SAS still need to be determined through some (easier and more robustness-guaranteed) numerical computation, and thus these methods are called semi-analytical.

2. What are the advantages of SAS?

In power system modeling and analysis, SAS has been proven to have the following features:

- **High numerical robustness.** Steady-state analysis usually requires solving nonlinear algebraic equations. Traditional tools usually use Newton-Raphson method or its variants, whose results can be highly dependent on the selection of starting point and they suffer from non-convergence problem. In contrast, SAS provides much better convergence thanks to the high-level analytical nature.
- **Enhanced computational performance.** In dynamic analysis, the traditional numerical integration approaches are essentially lower-order methods, which are confined to small time steps to avoid too-rapid error accumulation. These tiny time steps severely restrict the computation speed. In contrast, SAS provides high-order approximation, enabling much larger effective time steps and faster computation speed.
- **More accurate event-driven simulation.** For complex system simulation, it is common to simulate discrete events. Traditional numerical integration methods only provide solution values on discrete time steps and thus may incur substantial errors predicting events. In contrast, SAS provides an analytical form of solution as a continuous function, and thus can significantly reduce event prediction errors.

3. How is the performance of PowerSAS.m?

3.1 Benchmarking with traditional methods on Matlab

PowerSAS.m shows advantages in both computational robustness and efficiency over the traditional approaches.

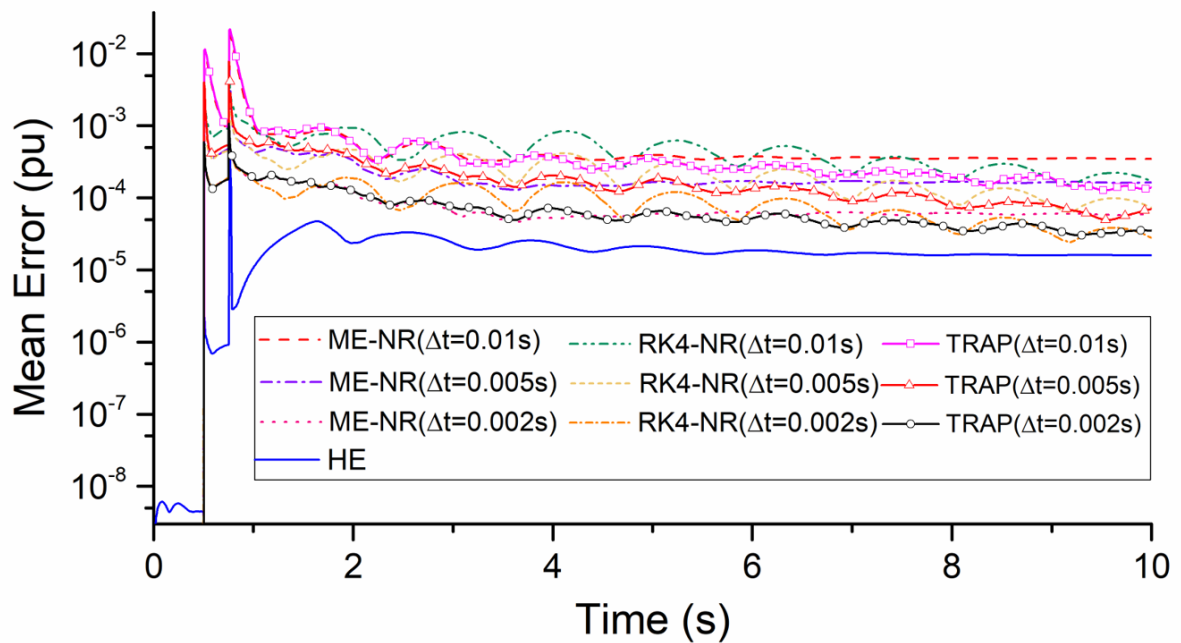
On **steady-state analysis**, we have done several benchmarking with traditional methods. For example, we test the steady-state contingency analysis on PowerSAS.m and Newton-Raphson (NR) method and its variants on Matlab. The test is performed on a reduced Eastern-Interconnection (EI) system and we tested on 30,000 contingency scenarios. The results suggest that the traditional methods have about 1% chance of failing to deliver correct results, while SAS has delivered all the correct results.

For more details, please refer to our recent paper:

- Rui Yao, Feng Qiu, Kai Sun, “Contingency Analysis Based on Partitioned and Parallel Holomorphic Embedding”, IEEE Transactions on Power Systems, in press.

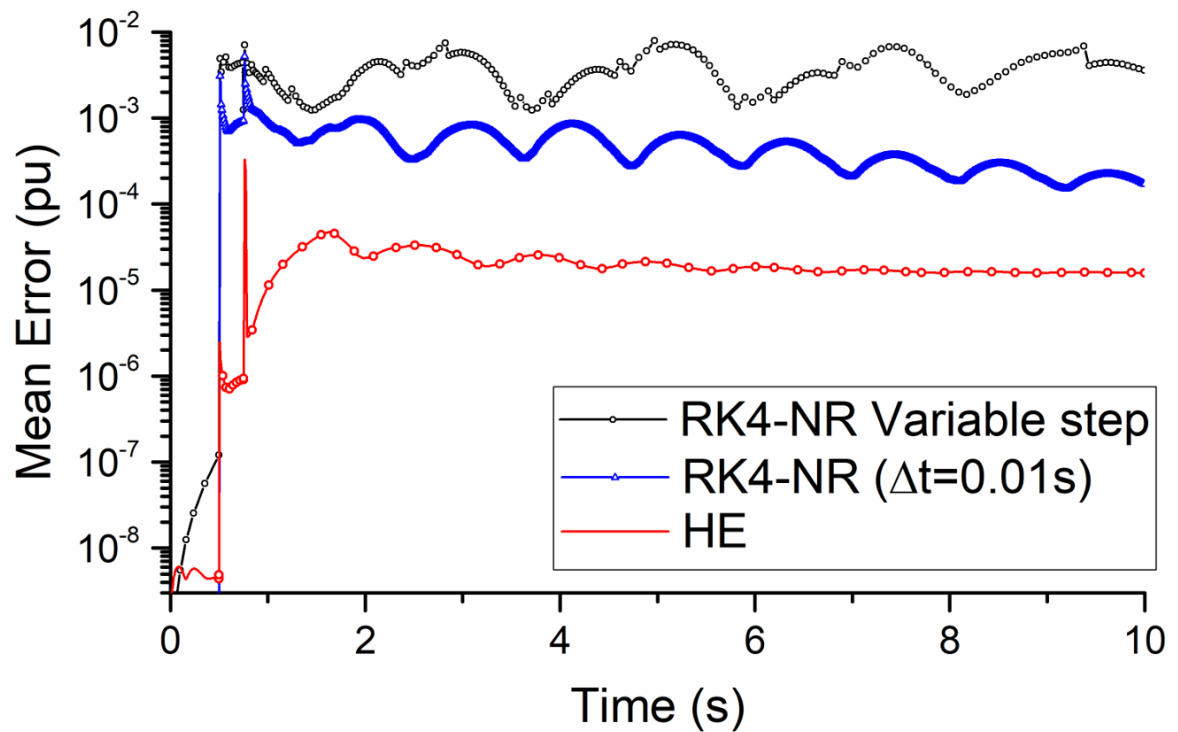
On **dynamic analysis**, we have compared with several most commonly used traditional numerical approaches for solving ODE/DAEs, including modified Euler, Runge-Kutta, and trapezoidal methods. Tests of transient-stability analysis on IEEE 39-bus system model and large-scale modified Polish 2383-bus system model have verified that SAS has significant advantages over the traditional methods in both accuracy and efficiency.

Accuracy comparison on IEEE 39-bus system (1) – Comparison with fixed-time-step traditional meth-



ods

Accuracy comparison on IEEE 39-bus system (2) – Comparison with variable-time-step traditional



method

Computation time comparison on IEEE 39-bus system

For more details, please refer to our recent paper:

- Rui Yao, Yang Liu, Kai Sun, Feng Qiu, Jianhui Wang, "Efficient and Robust Dynamic Simulation of

COMPUTATION TIME OF DIFFERENT METHODS IN IEEE 39-BUS SYSTEM

Method	Computation time (s)	Steps
ME-NR ($\Delta t = 0.01$ s)	21.58	1000
ME-NR ($\Delta t = 0.005$ s)	34.44	2000
ME-NR ($\Delta t = 0.002$ s)	84.46	5000
RK4-NR ($\Delta t = 0.01$ s)	34.64	1000
RK4-NR ($\Delta t = 0.005$ s)	64.78	2000
RK4-NR ($\Delta t = 0.002$ s)	156.37	5000
TRAP ($\Delta t = 0.01$ s)	24.34	1000
TRAP ($\Delta t = 0.005$ s)	36.02	2000
TRAP ($\Delta t = 0.002$ s)	95.51	5000
RK4-NR (variable step)	21.25	201
HE ($N_L = 30$)	4.49	54

Fig. 1: comp_time_039

Power Systems with Holomorphic Embedding”, IEEE Transactions on Power Systems, 35 (2), 938 - 949, 2020.

3.2 Benchmarking with PSS/E

3.2.1 Static Security Region (SSR)

Static Security Region (SSR) is an important decision-support tool showing region of stable operating points. However, there are often challenges on convergence when computing SSRs, especially near the boundaries. So SSR can be used for benchmarking the numerical robustness of computational methods.

We test SSR on IEEE 39-bus system by varying active power of buses 3&4. The active power of buses 3&4 are sampled uniformly over the interval of $[-4000, 4000]$ MW. The figure below shows the SSR derived by PSS/E and PowerSAS.m. It shows that PSS/E result have some irregular outliers (about 0.1% of the samples) outside of the SSR and actually are not correct solutions of power flow equations. In contrast, PowerSAS.m correctly identifies the SSR.

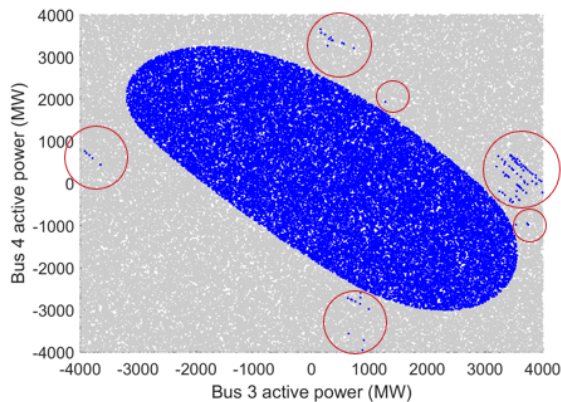


Fig. 1. PSS/E solvable (blue) and unsolvable (grey) samples. Note the circled outliers ($\sim 0.1\%$).

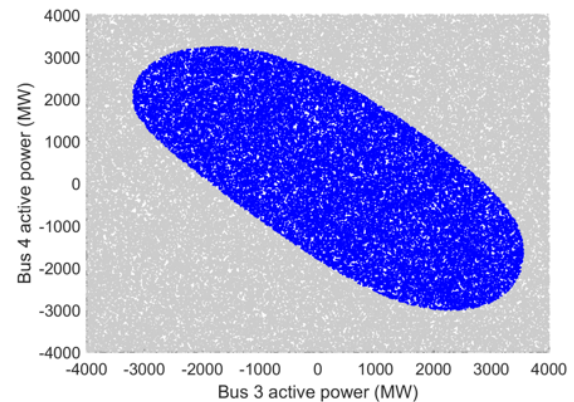


Fig. 2. SAS solvable (blue) and unsolvable (grey) samples.

Fig. 2: ssa_benchmarking

3.2.2 N-k Contingency analysis

Contingency ananlysis also has convergence challenges due to large disturbances. Here we perform bench-
marking between PSS/E (with and without non-divergence options) and PowerSAS.m on the N-25 contin-
gency analysis on a reduced eastern-interconnection (EI) system with 458 buses. We increase the load &
generation level by 15%, 20%, and 20.7%, respectively, as 3 different loading scenarios (loading margin is
20.791%). In each scenario, we randomly choose 5000 N-25 contingency samples.

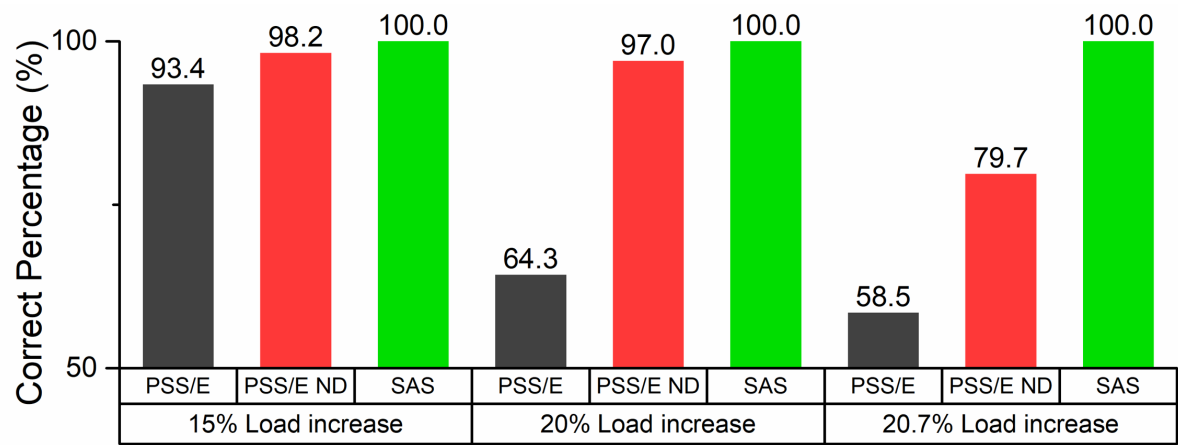


Fig. 3: contingency_458

The figure shows the percentage of correct results using different tools. It can be seen that PSS/E has some chance to deliver incorrect results, and the chance increases with loading level. In contrast, PowerSAS.m still returns results all correctly.

We also compared the computation speeds of PowerSAS.m and PSS/E. The figure below shows the average contingency analysis computation time of on the 458-bus system. The results show that SAS's speed is comparable to and even faster than PSS/E's.

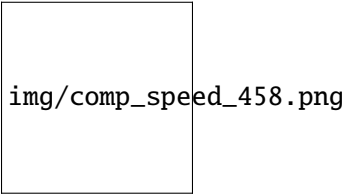


Fig. 4: x

INDICES AND TABLES

- genindex
- modindex
- search