2022 Open Source Modelling and Simulation of Energy Systems (OSMSES) | 978-1-6654-1008-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/OSMSES54027.2022.9769084

Recent Developments in Open Source Simulation Software pandapower and pandapipes

Roman Bolgaryn*, Gourab Banerjee*§, Dennis Cronbach*, Simon Drauz*,
Zheng Liu§, Maryam Majidi§, Hendrik Maschke*, Zhenqi Wang*§, Leon Thurner*
*Division of Grid Planning and Operation, Fraunhofer IEE, Kassel, Germany
§Department of Energy Management and Power System Operation, University of Kassel, Germany
roman.bolgaryn@iee.fraunhofer.de

***Abstract*—we introduce recent developments in the open source Python libraries pandapower and pandapipes. The purpose of pandapower is to provide an easy to use tool for power system analysis and enable a high degree of workflow automation. We describe the recent developments that extend the functionality of pandapower, such as new elements and controllers, as well as the calculation methods for state estimation, distributed slack, short circuit, asymmetric power flow, as well as advanced optimization functions via a new interface to the PowerModels.jl library. Furthermore, we describe the recent progress in the library pandapipes, which provides functionality to simulate pipe networks, such as district heating or gas networks. We demonstrate the new features of pandapipes in an example for transient temperature calculation and showcase the combined application of pandapower and pandapipes to model sector coupling.**

***Index Terms*—power system modeling, fluid system modeling, grid calculation, analysis, optimization, state estimation, short circuit calculation, asymmetric power flow, sector coupling, transient calculation, open source, simulation, automation, pandapower, pandapipes, PandaModels**

I. Introduction

The energy system is undergoing a transformation towards more renewable generation and electrification of mobility and heating. The associated increase in complexity of the power systems requires reliable simulation software that enables automation of workflows related to power system analysis and planning. Open source software, in particular pandapower and pandapipes, can be expanded or modified to match the requirements of a particular use case. Especially in distribution systems, where the majority of distributed energy resources are connected, open source tools provide a free, user-friendly, transparent and community-based alternative to commercial software in educational, scientific, and industry applications.

The development of pandapower and pandapipes has progressed since the last publications [1], [2]. The forthcoming sections describe the most important changes.

II. Open Source Software pandapower

Open source software pandapower was developed for power system analysis and is released under the 3-clause BSD license. The core functionalities of pandapower are based on PYPOWER [3], and have gradually been modified and expanded. pandapower is implemented in the programming language Python, which allows combining it with a vast set of Python-based libraries to automate workflows for power system analysis, starting with reading and processing input data in different formats, proceeding with static and quasi-static calculations and optimizations, and finally processing, visualizing and saving the results. [4] Its functionalities include (optimal) power flow calculation, state estimation, topological graph searches and short circuit calculations according to IEC 60909. The calculation results have been validated with the help of commonly used proprietary software [1]. pandapower has been successfully applied in several grid studies as well as for educational purposes [5]–[7].

A. Changes of grid elements and controllers

A new element motor makes it possible to model asynchronous machines. This element has additional input parameters that are necessary to describe an asynchronous motor. The nominal mechanical power is a required input parameter that replaces the active power. Efficiency of the motor, current in a locked rotor condition and $\cos\phi$, as well as the $\mathrm{R/X}$ ratio and the nominal voltage, allow a more detailed configuration for the power flow and short-circuit calculation.

A possibility to customize the particular, often grid operator-specific, behavior of grid elements is provided by the controller elements. The approach of the controller elements is to add a loop outside of the load flow calculation (fig. 1) to enable an automated adjustment of specified grid elements in reaction to the grid state. A typical use case for a controller element is modeling of On-Load Tap Changers (OLTC) of transformers.

The base *Controller* class provides the basic structure for all other controllers. The control loop consists of a sequence of methods that are triggered for every active controller in the grid at a time. First, if the control loop is executed as part of a time series simulation, the method *time_step* allows a controller to modify the grid state according to the current time step. The controller initialization step *initialize_control* is executed before running an initial power flow in order to enable sanity checks or setting up the grid for the calculation. Afterwards, the method *control_step* and power flow calculation are repeated sequentially until all of the controllers are converged, according to the *is_converged* method. The finalization step allows additional wrapping-up procedures, e.g. post-processing of results, if needed. The control loop is designed to provide a clear framework for

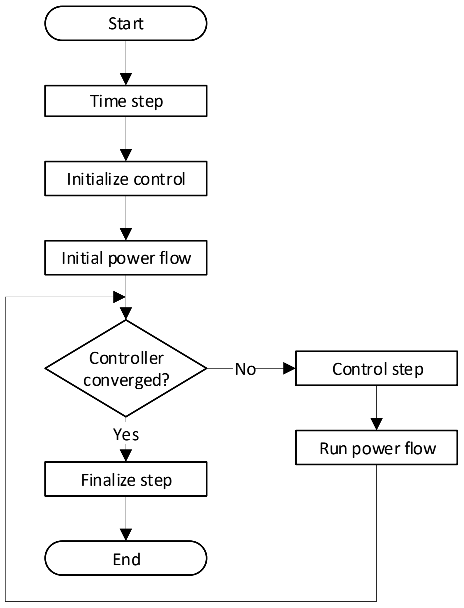

Fig. 1. Control loop implementation in pandapower

controller implementation and maximize the flexibility of controller implementation.

A new object table characteristic, which contains objects of a new class *Characteristic*, is added to pandapower. This new element table is similar to the table of *Controller* objects, in the way that it contains the object column. The *Characteristic* class, similarly to the *Controller* class, is derived from the *JSONSerializableObject* class, and therefore it integrates seamlessly in the existing JSON I/O routines.

The *Characteristic* class provides the capability to define a relationship between an input value and a desired output value. The inheriting class *SplineCharacteristic* implements this relationship by using a quadratic spline characteristic, as defined in the package SciPy [8]. We added a new kind of controller, *CharacteristicControl*, that relies on using the *Characteristic* object. This controller enables using any variable from any table as an input, passing it through a user-defined characteristic object, and writing the output in any other variable of any element. This provides a template to easily implement additional, use-case-specific, controllers. The flexibility of this controller was useful for implementing *VmSetTapControl* and *TapDependentImpedance*.

VmSetTapControl enables adjusting the transformer tap position depending on the transformer loading. The user can specify which variable from the transformer result table should be used as the input (e.g., p_hv_mw, loading_percent, i_lv_ka, etc.) and the *Characteristic* object. The output of the characteristic defines the voltage set-point and must be passed over to another controller, either *ContinuousTapControl* or *DiscreteTapControl*.

TapDependentImpedance provides the possibility to adjust the impedance of a transformer depending on its tap position. The tap position serves as the input of the characteristic, and the controller sets the new value of the relevant parameters in the trafo or trafo3w table.

The controller *DiscreteTapControl* was modified so that it can be used with the *VmSetTapControl* controller. It has received an alternative mode of operation, so that it can be initialized with a single voltage set-point rather than a voltage band (vm_lower_pu and vm_upper_pu for the lower and upper voltage limits respectively). Internally, the allowed voltage band is calculated based on the impact of the tap position on the voltage, as defined by the parameter tap_step_percent of the transformer. The controller must be initialized with the constructor *from_tap_step_percent* to use this functionality.

B. Tap dependent impedance of transformers

The consideration of the tap dependent impedance of 2-winding and 3-winding transformers is possible with a controller, as described in the previous section. As the next step, we improved the performance of the power flow calculation with tap dependent impedance of transformers by omitting the controller and implementing this functionality internally in pandapower. The impedance of tranformers is adjusted during the calculation of the branch impedance values for the internal data structure ppc. To activate this feature, the optional Boolean column tap_dependent_impedance must be added, which defines whether the tap dependent impedance must be calculated for a given transformer. For the 2-winding transformers, the columns vk_percent_characteristic and vkr_percent_characteristic must be added to the transformer table. In the case of the 3-winding transformers, the additional columns are $\mathrm{vk\_} < \mathrm{side} > \mathrm{\_percent\_characteristic}$ and $\mathrm{vkr\_} < \mathrm{side} > \mathrm{\_percent\_characteristic}$, where $< \mathrm{side} >$ stands for hv, mv and lv. These additional columns must reference the index of a tap dependent impedance characteristic from the net.characteristic table. We defined a convenience function create_trafo_characteristics in the pandapower control module to create and add the characteristic objects to net and assign them to the transformers.

C. State estimation

State estimation is a method to estimate the state of a power system by eliminating inaccuracies and errors from the grid measurement data. Measurements are not perfect, which leads to an inherent inaccuracy in the measurement value. Faulty devices can even return completely wrong measurement values. To account for these errors, the state estimation processes all available measurements to identify the likely real state of the electric grid. The output of the state estimation is therefore a set of complex voltage values for all buses in the grid, which can be used to calculate all the bus injections and branch flows. The inputs are the grid model and the measurement data.

The classical approach for state estimation is the Weight Least Square (WLS) method, which is widely used in well observed transmission grids. This method solves an optimization problem with the Newton's method, which minimizes the quadratic sum of the deviation between the estimated state to the measurement value weighed by the measurements'

TABLE I
IMPLEMENTED OPTIMIZATION METHODS AND ESTIMATORS

State Estimation/Optimization methods	Available estimators
wls	quadratic
wls with zero injection constraints	quadratic
lp	LAV
irwls	quadratic, SHGM
SciPy Optimization Tool	quadratic, LAV, QL, QC

accuracy. The drawback of the method is that the result is sensitive to bad data, especially those with high measurement errors, because of the underlying quadratic estimator. In reality, this could be the case with faulty measurement devices. In addition, for distribution grids, which have comparably lower measurement density, pseudo measurement values with low fidelity are added to ensure visibility for the state estimation method. Major improvements of the implementation of state estimation in pandapower address these issues.

To reduce the influence of the bad data, robust estimators are required. This is realized by e.g. using a linear estimator instead of the quadratic estimator, such as in Least Absolute Value (LAV). Similarly for the Quadratic Linear (QL) and Quadratic Constant (QC) estimator, the piece-wise function replaces only the section with large deviation, e.g., the section with greater than 3σ deviation, as a linear function or constant value. Multiple robust estimators such as LAV, QC, QL, and Schweppe-type Huber Generalized Maximum-likelihood (SHGM) are implemented in pandapower, as described in literature [9], [10]. For the SHGM estimator, the measurement weights are dynamically adapted during the Iteratively Reweighted Least-Square (IRWLS) iterations.

Different optimization methods are required to incorporate the estimators. Besides the Newton's method, e.g. Linear Programming (LP) is used to solve the LAV state estimation problem. The universal SciPy Optimization Tool is utilized for the aforementioned estimators [8]. An overview of the implemented state estimation methods is provided in table I.

Another major improvement is to model the state estimation problem with the unified Extended ppci (eppci) class. This implementation not only simplifies the interface to the optimization library but also enables the state estimation to be executed by combining multiple different estimators in a chain, which further improves the convergence.

D. PandaModels: pandapower-PowerModels Interface

To solve the classic AC and DC optimal power flow in power systems, pandapower has combined its element-based data structure with the power flow optimization in PYPOWER [3], [11]. This allows solving the cost-related optimization problems such as dispatch optimization, load shedding and loss minimization [1].

Alternatively, pandapower provides a direct interface to PowerModels [12]. PowerModels is a Julia [13] package based on JuMP [14] environment steady-state power system optimization. It enables computational evaluation of power system models with focus on decoupling problem terms from the different formulations. The interface to PowerModels was implemented as a pandapower-PowerModels converter that enables the transformation of a pandapower grid into the PowerModels format and converts the PowerModels optimization results back into the original pandapower grid. However, this interface has some limitations as only a few models related to cost optimization in PowerModels can be called. Additionally, more diverse and customized parameters for optimization cannot be used. The supported cost functions are limited to piece-wise linear and n-polynomial formulations.

Besides all the aforementioned issues, the technical obstacles related to the build-in interface are required to be developed. Although the pandapower users are python users, they ought to install Julia and the required packages manually to set up the basic interface. Moreover, as the basic interface was not provided the automatic synchronization between pandapower and PowerModels and its dependencies, after while, the users may receive warnings or errors related to the old version of the packages, therefore they need to manually manage the Julia packages.

To improve the pandapower-PowerModels interface and deal with the variety of optimization problems, we developed a new open-source Julia package PandaModels [15]. The structure of PandaModels is introduced in fig. 2.

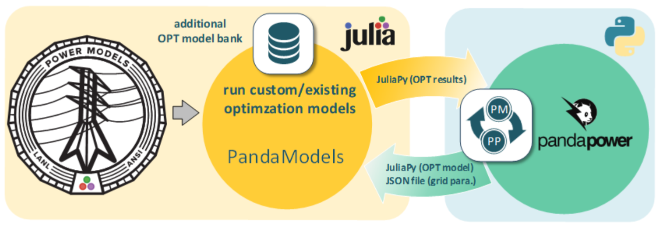

Fig. 2. General scheme of PandaModels

PandaModels is a Julia package under development that contains supplementary data and code to prepare pandapower grid models in a compatible format for Julia packages which are based on InfrastructureModels [16], particularly PowerModels. It allows calling all of the existing models and also provides an additional database based on the PowerModels framework for users to solve custom and application-oriented optimization problems.

Presently, we have implemented two application-oriented models for the reactive power optimization, i.e., bus voltage Deviation Minimization (VD) and reactive power Injection Minimization (QI). The improved pandapower-PowerModels converter enables the simple setting of customized parameters based on the pandapower data structure.

To better understand the functionalities of the new developments, in the following, the steps for implementing the VD are briefly described.

- ***motivation:*** Distribution System Operator (DSO) requires minimizing the deviation of the voltage amplitude at buses located at the DSO-TSO interface from its pre-defined set-point by optimizing the reactive power of Distributed Energy Resources (DER).

- ***data configuration:*** Based on the original pandapower data structure, specific optimization-parameters can be passed through the converter, by adding additional columns to the pandapower elements named *“pm-param/<PARAMETER_NAME>”*. All these parameters are collected into the ”user_defined_params” PandaModels internal dictionary and, translated to PowerModels exclusive dictionary for external parameter. In this case, the column “pm-param/setpoint_v” is added to **pandapower.net.bus** data frame, then, we need to define the set-points for the buses located at the interfaces between DSO and the Transmission System Operator (TSO), e.g. *setpoint_v* $= 1.03$.
- ***optimization model:*** The general form of VD problem while set *setpoint_v* $= 1.03$ is defined as follows:

$$\begin{aligned} \underset{\mathcal{X}=[q,\ldots]}{\text{minimize}} \quad & \sum_{i \in \mathcal{BI}} [v_i - 1.03]^2 \\ \text{subject to} \quad & g(\mathcal{X}) = 0 \\ & h(\mathcal{X}) \leq 0 \end{aligned} \tag{1}$$

  where v_i is the voltage variable of bus i in $\mathcal{BI}$ which denotes the set of buses located at the DSO-TSO interfaces. The $g(\mathcal{X})$ and $h(\mathcal{X})$, denote equality and inequality constraints, respectively. The $\mathcal{X}$ denotes the set of variables decisions, such as reactive power, q.
- ***execution:*** After preparing the grid model and all essential information, by calling the function **pp.runpm_vd**, the pandapower-PowerModels interface converts the grid to PowerModels format and saves it as a JSON [17] file. Afterwards, PandaModels reads the JSON file, checks the data, and prepares the extra information, before calling the VD model from the developed database of PandaModels. After executing the optimization problem, the result dictionary containing reactive power set points (PowerModels format) is converted back to the pandapower format, with the help of the pandapower-PowerModels converter.

The addressed limitations with the built-in pandapower-PowerModels interface are solved using PandaModels and introducing automated synchronization in the developed version of pandapower. This automated synchronization introduced in *_call_pandamodels* function called when the user runs optimization models from *runpm.py*.

Firstly, the automated synchronization calls Julia (as python package), however, the error will be raised if python could not connect to Julia. Afterward, it checks whether the pandamodels package has been already installed, otherwise it automatically adds the registered version of the package. Moreover, when the PandaModels developers run optimization models from *runpm.py*, they need to set the *dev_mode* option to *True*, this option activates the develop mode of the PandaModels package.

Furthermore, by updating the pandapower version or running its tests, the users will automatically access the most updated version of the PandaModels, which contains PowerModels as its dependencies. However, the users are still required to manually install Julia and PyCall [18] by the existing version of the automated synchronization.

E. Distributed slack

In power flow analysis, a slack bus is chosen which compensates the mismatch of active power in the grid as related to the balance of consumption, generation and losses. This mismatch results from grid losses or an unbalanced dispatch. In real transmission and distribution systems, multiple generators can play a role in compensating the mismatch of active power. The approach of distributed slack allows to model the allocation of the mismatch compensation to a number of buses according to pre-defined weights, which more closely resembles real-world scenarios. To describe how much active power a single element contributes to the compensation of the mismatch of active power, the parameter $\mathrm{slack\_weight}$ is used.

In pandapower, the power flow calculation with Newton-Raphson algorithm supports the functionality of distributed slack. To this end, the user can set the argument $\mathrm{distributed\_slack}$ to True when calling the function *runpp*. The slack weights for $\mathrm{ext\_grid}$, gen and ward elements should be provided in the $\mathrm{slack\_weight}$ column.

We used the implementation of distributed slack in PyPSA [19] as reference. We extended the formulation of the Jacobian matrix to include an additional column that represents the slack weights, and an additional row that represents the slack bus. The implementation is valid for one interconnected zone. In further work, the implementation can be expanded to multiple zones, if necessary.

The resulting active power of an element i after the power flow calculation with distributed slack is represented in eq. (2) (ng: number of generation plants, nc: number of consumptions, nb: number of branches).

$$\begin{aligned} P_{result,i} = P_{set,i} - (\textstyle\sum_{elm=1}^{ng} P_{generation,elm} - \\ \textstyle\sum_{elm=1}^{nc} P_{consumption,elm} - \sum_{branch=1}^{nb} P_{losses,branch}) \cdot \\ \cdot\, slack\_weight_i \end{aligned} \tag{2}$$

F. Short circuit calculations

The short circuit calculation in pandapower was extended to modeling of unsymmetrical fault. This enables the calculation of single-phase-to-ground, phase-to-phase, and three-phase short-circuit currents with pandapower. To calculate the single phase short-circuit current $I_{k,1}$, it is necessary to consider the zero-sequence and positive-sequence admittance of the grid elements. The input parameters for zero-sequence impedance were added to the branch elements in pandapower. The new parameters are presented in table II. The supported transformer vector groups are Dyn, Yyn, YNyn, Yzn. These parameters are

TABLE II
ADDITIONAL PARAMETERS FOR CALCULATION OF UNSYMMETRICAL FAULT

Line parameters	Transformer parameters
r0_ohm_per_km	vk0_percent
x0_ohm_per_km	vkr0_percent
c0_ohm_per_km	mag0_percent
g0_ohm_per_km	mag0_rx
	si0_hv_partial
	vector_group

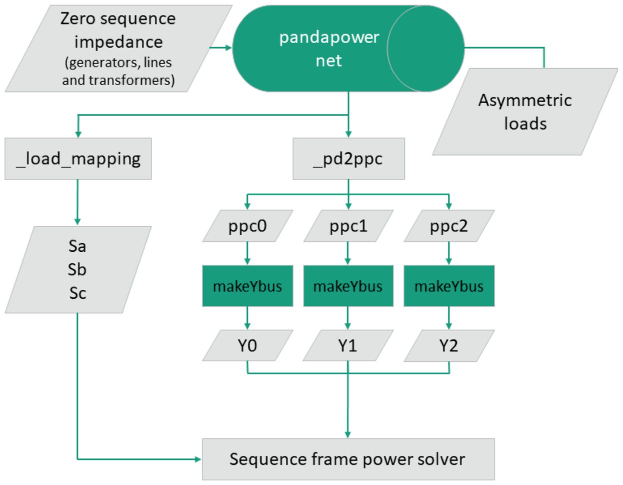

Fig. 3. Flow chart of the three-phase power flow algorithm

used internally to formulate the zero sequence admittance matrix in addition to the positive sequence and negative sequence matrices. Using these matrices, the short-circuit currents are calculated according to the standard IEC 60909-0:2016 [20].

We implemented further aspects included in IEC 60909-0:2016, namely the correct consideration of power station units and wind power station units. Furthermore, we extended the ground fault short-circuit calculation for three-winding transformers by implementing their relevant vector groups: YNyd, YNdy, YYnd, YNynd, YNdyn, YNdd, YNyy, Ydyn, Yyd, Yyy, Ydd, Ydy, Yyd, Ddd.

G. Asymmetric power flow calculations

Based on the newly included zero and negative sequence modeling, we extended pandapower with an unbalanced power flow calculation algorithm. The implementation is especially useful for studies of Low Voltage (LV) grids, it enables calculating unbalanced power flows through symmetric, three phase lines. Unsymmetrical feeders (like e.g. in North America), as well as single- or two-phase lines and transformers, are planned to be added in the future. For unbalanced power flow input, the new elements asymmetric_load (unbalanced loads) and asymmetric_sgen (unbalanced static generators) are added to allow the phase-wise definition of active and reactive power values.

The current state of development allows for the use of relevant elements for the LV distribution grid, i.e. asymmetric loads and asymmetric static generators, as well as lines/cables, buses, and 2-winding transformers. Currently, the transformers support the following vector groups: Dyn, YNyn, and Yzn, which are the most commonly used transformer vector groups for low voltage distribution grids in Europe.

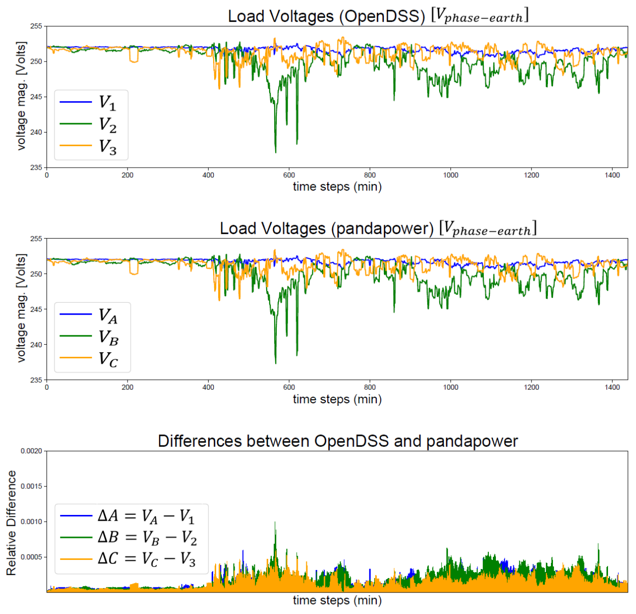

Fig. 4. Result comparison between OpenDSS and pandapower

The calculation algorithm follows the method described in [21] and utilizes the method of the symmetric components to convert the asymmetric grid parameters into a set of positive-, negative- and zero-sequence values as shown in fig. 3. The algorithm utilizes the Newton-Raphson method, similarly to the balanced power flow calculation, for the positive sequence parameters and the current injection method for negative- and zero-sequence parameters. This algorithm works for both radial and meshed grids and is integrated in the power flow function of pandapower.

The results of pandapower unbalanced power flow are validated by comparison to the commercial software DIgSILENT PowerFactory [22] and the open-source simulation software OpenDSS [23]. "IEEE European Low voltage grid" is modeled and the unbalanced power flow is calculated both in pandapower as well as in OpenDSS. The absolute value of the voltage magnitude difference is compared, and a negligible difference is achieved as a result, which is shown in fig. 4.

III. OPEN SOURCE SOFTWARE PANDAPIPES

In 2020 we extended our power system analysis library pandapower by an open source pipe grid simulation library pandapipes. pandapipes is not only able to simulate fluid systems containing incompressible and compressible media, but also introduces a module *multinet* that connects pandapower and pandapipes grid models in a common calculation and enables sector coupling simulations [2].

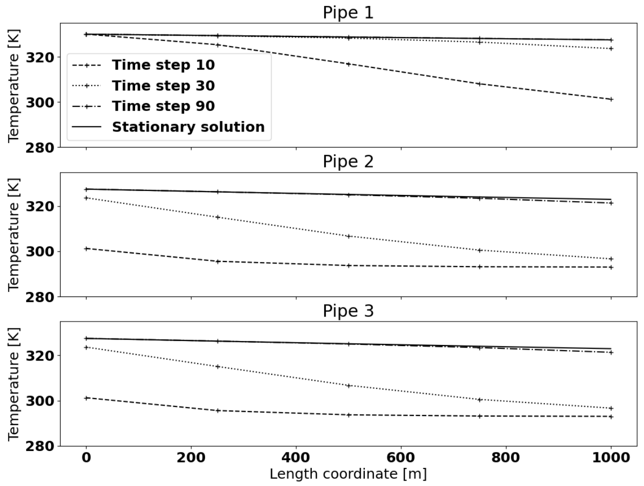

Fig. 5. Temperatures for different time steps in minutes along the three pipes in the example grid shown in fig. 6.

The *multinet* module uses the controller concept in pandapower to couple a pandapower and a pandapipes grid, allowing a simultaneous manipulation of the components in both grids. This approach has been successfully implemented for a power and gas grid coupled by an electrolyser, as well as for a power and a heating grid coupled by a heat pump [2]. So far, however, it was not possible to consider mixtures within a gas grid. First implementations for simple gas grids have been successfully tested and will be released soon.

A. Transient calculations

For typical applications of pandapipes, it is not necessary to take into account the transient behavior of hydraulic properties. However, heating processes are subject to thermal inertia that must be considered. Observing the time-dependent behavior of temperature inside a district heating grid provides further insights into a district heating system. For example, a time delay until temperature losses exceed a prescribed value can be evaluated. The transient behavior of a water storage tank was observed in [2]. The model was implemented as a controller describing a behavior of an electric heater installed in the storage tank. A Backward Differentiation Formula solver from the SciPy package was applied to the corresponding differential equation. The data structure of pandapipes made it necessary to reinitialize this solver in every time step, which in turn led to performance issues.

To accelerate the calculation, we implemented the implicit Euler method directly in pandapipes. Because this method was integrated seamlessly into the current data structure of pandapipes, only a few changes to the underlying equations were necessary. To include the time-dependent behavior of temperature of the fluid inside a pipe, an additional term needs to be included in the corresponding equation, which represents the time derivative of the solution variable. The equation for a single pipe is shown in eq. (3).

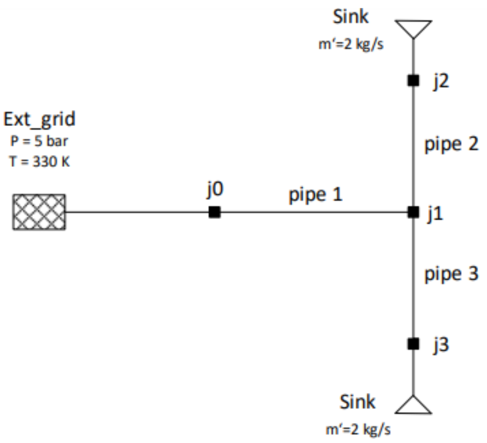

Fig. 6. Exemplary grid for transient temperature calculation in pandapipes.

$$\rho A C_p \frac{\partial T}{\partial t} + \rho A C_p v \frac{\partial T}{\partial x} = \alpha d \pi \cdot (T_{amb} - T) \tag{3}$$

In equation eq. (3), ρ is the fluid density, A the cross-sectional area of the pipe, C_p the heat capacity, T the fluid temperature, t the time, v the fluid velocity, α the heat transfer coefficient, d the pipe diameter, T_{amb} the ambient temperature, and x the spatial coordinate.

The equation is based on [24]. The balance equation implemented at junctions is not changed, as it is assumed that mixing occurs instantaneously. With these changes, we simulated small test grids carrying incompressible media, similar to the grid displayed in fig. 6. This grid consists of three pipes, each having a diameter of 75 mm and a length of 1 km. An external grid at the junction j0 sets the pressure to 5 bar and the temperature to 330 K. A mass flow of 2 kg/s is drawn via sinks connected at the junctions j2 and j3. Initially, all nodes have a temperature of 293 K. To be able to represent the temperature behavior properly, each pipe contains 5 auxiliary nodes. The thermal model includes heat transfer between the pipes and their environment. The external temperature is set to 293 K, while the heat transfer coefficient is set to 5 $\frac{W}{m^2 \cdot K}$. The time resolution is set to 1 min.

New fluid enters the system at the junction j0 with the temperature of 330 K, leading to a temperature increase according to the specified time step. fig. 5 shows the temperature along the pipe for different time steps. Additionally, the steady-state solution is plotted. After a certain amount of time steps, the calculated temperature reaches the steady-state temperature.

Although no new solution variables are added in eq. (3), a higher spatial resolution is required to accurately calculate the temperature of the medium along the pipe. This leads to a larger equation system and increases the computation time.

The introduced approach is applicable to all types of components in pandapipes. Similarly, a water storage with an arbitrary number of layers was modeled based on [25]. A general formulation of the model is presented in eq. (4).

$$
\begin{aligned}
m_i c_p \frac{\partial T_i}{\partial t} = \delta_i^s \dot{m}_s c_p (T_{f,s} - T_i) + \delta_i^l \dot{m}_l c_p (T_{r,l} - T_i) + \\
\alpha A_{ext,i} (T_{amb} - T_i) + \\
\delta_i^+ \dot{m}_i c_p (T_{i+1} - T_i) + \delta_i^- \dot{m}_i c_p (T_i - T_{i-1}) + \\
\frac{A_i \lambda_{eff}}{z_i} (T_{i+1} - 2T_i + T_{i-1})
\end{aligned} \tag{4}
$$

In equation eq. (4), m_i is the mass of layer i,c_p is the heat capacity, T_i is the temperature of layer i, T_{i+1} the temperature of layer (i+1), T_{i-1} the temperature of layer (i-1), t the time, $A_{ext,i}$ the external surface of layer i, $\dot{m}_s$ the mass flow of the source circuit, $\dot{m}_l$ the mass flow of the load circuit, z_i the height of layer i, λ_{eff} the verticalheat conductivity, A_i the cross-sectional area of layer i.

The model has been successfully tested independently of pandapipes with the implicit Euler method combined with a Newton Raphson method and will be included into pandapipes in the near future. Both examples prove that the approach used in pandapipes for solving pipe grids can be adjusted to model transient behavior.

Conclusion

In this paper we introduce recent developments of pandapower and pandapipes. Energy system transformation requires reliable solutions for power system analysis and planning. The presented tools provide a free, user-friendly and transparent simulation software, which is continuously developing. The improvements include calculation methods for state estimation, distributed slack, short circuit and asymmetric power flow for pandapower. Additionally, we describe transient temperature calculation with pandapipes and showcase the application of sector coupling with both tools.

Acknowledgment

This work presents selected results of the project "Robust-Plan" and "Hybridbot", which are funded by the German Federal Ministry for Economic Affairs and Energy. The authors also thank Michael Jurasovic with Vippy for his support with the implementation of the distributed slack functionality. The development of state estimation was supported by the European Union's Horizon 2020 research and innovation program within the project EU-SysFlex. Further development of short-circuit calculation and the implementation of tap dependent impedance of transformers was supported by 50Hertz Transmission GmbH.

References

[1] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, and M. Braun, "pandapower - an open source python tool for convenient modeling, analysis and optimization of electric power systems," *IEEE Transactions on Power Systems*, vol. 33, no. 6, pp. 6510–6521, November 2018.

[2] D. Lohmeier, D. Cronbach, S. R. Drauz, M. Braun, and T. M. Kneiske, "Pandapipes: An open-source piping grid calculation package for multi-energy grid simulations," *Sustainability*, vol. 12, no. 23, 2020. [Online]. Available: https://www.mdpi.com/2071-1050/12/23/9899

[3] R. Lincoln. (2021) PYPOWER: Port of MATPOWER to Python. [Online]. Available: https://github.com/rwl/PYPOWER

[4] Department of Energy Management and Power System Operation (University of Kassel) and Institute for Energy Economics and Energy System Technology IEE (Business Unit Grid Planning and Operation). (2016) pandapower documentation. [Online]. Available: https://pandapower.readthedocs.io/en/develop/

[5] M. Braun, I. Krybus, H. Becker, R. Bolgaryn, J. Dasenbrock, P. Gauglitz, D. Horst, C. Pape, A. Scheidler, and J. Ulffers, "DER integration study for the German state of Hesse – methodology and key results," in *CIRED 2019 (25th International Conference on Electricity Distribution)*, Madrid, June 2019.

[6] R. Bolgaryn, A. Scheidler, J. Dasenbrock, and M. Braun, "Automated planning of high voltage grids for DER integration studies–results of a study for the German state of Hesse," in *CIRED 2019 (25th International Conference on Electricity Distribution).* Madrid: AIM, June 2019.

[7] C. Ma, S. Drauz, R. Bolgaryn, J. Menke, F. Schäfer, J. Dasenbrock, M. Braun, L. Hamann, M. Zink, K. Schmid *et al.*, "A comprehensive evaluation of the energy losses in distribution systems with high penetration of distributed generators," in *25th International Conference and Exhibition on Electricity Distribution (CIRED 2019)*, 2019.

[8] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[9] A. Abur and A. G. Exposito, *Power system state estimation: theory and implementation*. CRC press, 2004.

[10] L. Mili, M. Cheniae, N. Vichare, and P. J. Rousseeuw, "Robust state estimation based on projection statistics of power systems," *IEEE Transactions on Power Systems*, vol. 11, no. 2, pp. 1118–1127, 1996.

[11] 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 Transactions on Power Systems*, vol. 26, no. 1, pp. 12–19, Feb 2011.

[12] C. Coffrin, R. Bent, K. Sundar, Y. Ng, and M. Lubin, "Powermodels. jl: An open-source framework for exploring power flow formulations," in *2018 Power Systems Computation Conference (PSCC)*. IEEE, 2018, pp. 1–8.

[13] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, "Julia: A fresh approach to numerical computing," *SIAM review*, vol. 59, no. 1, pp. 65–98, 2017.

[14] I. Dunning, J. Huchette, and M. Lubin, "Jump: A modeling language for mathematical optimization," *SIAM review*, vol. 59, no. 2, pp. 295–320, 2017.

[15] e2nIEE. Pandamodels.jl. [Online]. Available: https://e2niee.github.io/PandaModels.jl/dev/

[16] C. Coffrin. Infrastructuremodels.jl. [Online]. Available: https://lanl-ansi.github.io/InfrastructureModels.jl

[17] F. Pezoa, J. L. Reutter, F. Suarez, M. Ugarte, and D. Vrgoč, "Foundations of json schema," in *Proceedings of the 25th International Conference on World Wide Web*, 2016, pp. 263–273.

[18] S. G. Johnson. Pycall.jl. [Online]. Available: https://github.com/JuliaPy/PyCall.jl

[19] T. Brown, J. Hörsch, and D. Schlachtberger, "PyPSA: Python for Power System Analysis," *Journal of Open Research Software*, vol. 6, no. 4, 2018. [Online]. Available: https://doi.org/10.5334/jors.188

[20] I. Kasikci, *Short circuits in power systems: a practical guide to IEC 60909-0*. John Wiley & Sons, 2018.

[21] M. Abdel-Akher, K. Mohamed Nor, and A.-H. Abdul-Rashid, "Development of unbalanced three-phase distribution power flow analysis using sequence and phase components," in *2008 12th International Middle-East Power System Conference*, 2008, pp. 406–411.

[22] "DIgSILENT GmbH, DIgSILENT PowerFactory, 2020."

[23] Electric Power Research Institute (EPRI). (2021) OpenDSS. [Online]. Available: https://www.epri.com/pages/sa/opendss

[24] K. Baher, Hans-Dieter; Stephan, *Wärme- und Stoffübertragung*. Springer, 2010.

[25] P. Sawant, "A contribution to optimal scheduling of real-world tri-generation systems using economic model predictive control," Ph.D. dissertation, Technischen Universität Dresden, 2020.