Software Impacts 19 (2024) 100603

Contents lists available at ScienceDirect

Software Impacts

journal homepage: www.journals.elsevier.com/software-impacts

Original software publication

pymfm—A Python framework for microgrid flexibility management

A. Ahmadifar [a,*], E. Gümrükcü [a], A. Yavuzer [a], F. Oppermann [a], A. Monti [a,b]

[a] Institute for Automation of Complex Power Systems, E.ON Energy Research Center, RWTH Aachen University, Mathieustrasse 10, Aachen, Germany
[b] Fraunhofer Institute for Applied Information Technology FIT, Schinkelstr. 2, 52062 Aachen, Germany

ARTICLE INFO

Keywords:
Distributed energy resources
Energy management
Microgrid
Optimization

ABSTRACT

pymfm is an open-source Python framework for microgrid flexibility management. It is used for developing and testing management strategies according to the rule-based and optimization-based algorithms. This framework allows to control flexible assets in form of battery energy storage and photovoltaic units within microgrids and in both (near) real-time and scheduling operation modes. Additionally, pymfm provides simulation routines for microgrids and can be used by researchers, system operators, aggregators, microgrid owners, renewable energy communities, etc. An example is also provided and serves as an illustration of how the framework can be used to test custom energy management strategies.

Code metadata

Current code version	v0.5.5
Permanent link to code/repository used for this code version	https://github.com/SoftwareImpacts/SIMPAC-2023-437
Code Ocean compute capsule	https://codeocean.com/capsule/0245765/tree/v1
Legal Code License	Apache2.0
Code versioning system used	git
Software code languages, tools, and services used	Python 3
Compilation requirements, operating environments & dependencies	pymfm can be used independently of the hardware. The Python site-package requirements are pandas, matplotlib, scipy, pyomo, astral, xlsxwriter, and pydantic. Additionally, a mathematical programming solver compatible with the pyomo optimization modeling library is required.
If available Link to developer documentation/manual	https://pymfm.fein-aachen.org
Support email for questions	post_acs@eonerc.rwth-aachen.de

1. Introduction

A Microgrid (MG) is a distinct electrical entity comprising interconnected loads and Distributed Energy Resources (DER)s. It operates as a controllable unit within the grid, capable of connecting or disconnecting as needed to function in both grid-connected and islanded modes [1]. What sets MGs apart from mere DER-interconnected lines is their resource control, presenting them as coordinated entities to the upstream network. Moreover, MGs can operate independently, enhancing reliability and resilience in the event of upstream network faults [2]. In this context, pymfm serves as an open-source Python framework for development and testing of management concepts for MGs. It enables MG operators to manage flexibility resources in both grid-connected and islanded modes while increasing self-sufficiency of the MGs and minimizing power exchanges with the upstream networks.

Pymfm offers a range of decision algorithms, each tailored for specific UCs and adaptable to MG scenarios with various flexibilities. The library incorporates simulation routines that execute different algorithms depending on the chosen management concept. By running these simulation routines, users can observe how a particular management concept behaves in a user-defined scenario. For example, they can model scenarios involving community and household battery storage systems with specific parameters and predefined control conditions, providing insights into Near-Real-Time (NRT) or Scheduling (SCH) operation modes. The existing simulation routines also serve as valuable references for creating new management concepts. Users have the flexibility to modify these routines by incorporating their own custom algorithms into the existing algorithms. Additionally, pymfm's scenario-generation routines, which generate different UC scenarios

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail address: amir.ahmadifar@eonerc.rwth-aachen.de (A. Ahmadifar).

https://doi.org/10.1016/j.simpa.2023.100603
Received 22 October 2023; Received in revised form 10 November 2023; Accepted 22 November 2023

with respective boundary conditions, contribute to the operational research in the MG energy management domain.

2. Software architecture and functionalities

pymfm can be found on the Python Package Index (PyPI) repository [3] and is distributed under the Apache license. It is constructed using underlying site packages such as numpy [4] and scipy [5]. Pymfm's optimization problem is formulated using the pyomo modeling language [6], for which a diverse range of solvers, including Gurobi [7] and SCIP [8,9], can be utilized to solve the respective optimization problem. The architecture of pymfm comprises two Python packages: *scenario_forecast_kit* and *control* which are described below:

control package comprises Python implementations of Rule-Based (RB) and Optimization-Based (OptB) *algorithms*. These functions accept standard Python data types such as integers, floats, and dictionaries as inputs and generate outputs in the same standard Python data types. Additionally, the parameters related to decision-making are handled by the data handling components under *utils*. The simulation routines are implemented through the *mode_logic_handler.py* that links the data handling components to the *algorithms*. Each simulation routine aligns with a specific logical sequence of decisions and actions. Specifically, *mode_logic_handler.py* collects various parameters from the *BatterySpecs* objects, including initial State of Charge (SoC) and target Final SoC (FSoC), the number and types of flexibilities (which may encompass community battery energy storage *cbes* and household battery energy storage *hbes*), and, based on the selected *OperationMode*, invokes the relevant *ControlLogic* to compute the power setpoints for the flexibilities. In the case of *near_real_time* operation mode, the *rule_based* control algorithm is executed on *MeasurementsRequest* class which holds the near real time measurement and requested powers input data of the MG. However, in *scheduling* mode, both *rule_based* and *optimization_based* control algorithms can be employed on *GenerationAndLoad* class which holds the forecast input data of the MG. These executions result in the generation of a specific output DataFrame that includes information such as flexibility setpoint(s), corresponding SoCs, net power exchange within the MG after the application of control actions, etc. It is noteworthy that the *rule_based* control logic has limited capability and can handle only a single *cbes* unit while the *optimization_based* control algorithm has more features and can handle multiple flexibility units including the optional Phtovoltaic (PV) curtailment, upper and lower boundaries for net power delivery, bulk delivery/reception of energy from flexibilities, etc.

In addition to the *control* package, *scenario_forecast_kit* package provides scenario and forecast generation routines. The *forecast_generation* functions process the forecast input data to produce forecasts according to the Standard Load Profile (SLP) [10] and the corresponding user defined SLP-scaling parameters, PV forecast, and the user-defined number of households within an MG. Additionally, relevant Python functions of *scenario_generation* receive parameters such as battery specifications including initial SoC and target FSoC and storage types, bulk delivery/reception of energy, upper and lower bound profiles for the power delivery of MG. These parameters along with other UC-related input parameters and forecast profiles are processed to produce specific scenarios. These scenarios basically define the boundary conditions necessary for implementing different UCs, i.e., different control logics and operation modes with different sources of flexibility.

Individuals with a basic grasp of Python can easily apply the provided simulation routines to evaluate management concepts within their specific scenarios. To do this, users should input their scenarios in JSON format into the simulator and set the boundary conditions for their desired UCs. In such instances, users can modify the existing simulation routines to assess customized operational processes. For instance, the upper and lower boundaries used by *scenario_forecast_kit* package, which set the power boundaries for *optimization_based* control algorithm, can originate from other simulators like pandapower [11] or any other higher system level power flow manager. This enables exploring the impact of setting specific boundaries on the power delivery of the MGs and the corresponding coordination among multiple MGs via such boundaries. Additionally, the current version of pymfm offers adaptable references allowing it to integrate and run external tools using Application Programming Interface (API)s and in form of co-simulation.

3. Impact

Pymfm addresses the evolving operational landscape within MGs with new technologies, services, and participants [12]. It provides a comprehensive package, including scenario generation tools and a range of functions and algorithms for MG management. Users can tailor decision processes through flexible simulations, and existing routines offer references for potential enhancements, aiding operational research. Accessible as an open-source library, pymfm caters to researchers, engineers, and policymakers with basic Python skills. It streamlines MG energy management research, contributing to society's decarbonization goals.

pymfm has been used as a testing environment for comparing storage coordination algorithms in recent publications [13,14]. Furthermore, it has been successfully applied in some projects for MG flexibility management. For instance, in the Horizon Europe research and innovation project Platone [15] and within the German demonstrator, pymfm has been used to develop an energy management system in charge of balancing out the generation and consumption within a rural Renewable Energy Community (REC). As described through educational videos series and promoted via the Linux Foundation Energy (LFE) Project SOGNO [16], pymfm (the previous versions and sometimes referred to as "balancing" service) has been deployed in Docker containers and interfaced with external systems via well-established APIs [17,18]. Additionally and in the ongoing second and the upcoming third phases of the Kopernikus project Ensure [19], pymfm is used to develop local energy management systems for REC MGs at low voltage grid side of a field trial with high penetration of PVs.

With respect to future developments, the authors aim to fully dockerize the current pymfm version, incorporating an open-source solver for its optimization problem. Real-field tests are planned during the third phase of the Ensure project, along with deploying pymfm at the premises of the relevant MG operator. Another future plan involves integrating pymfm with the datafev Python framework for managing electric vehicle charging infrastructures [20] with the aim to optimize day-ahead and intra-day MG management, considering electric vehicles and their charging infrastructures. All these future developments and tests adhere to grid codes and ensure pymfm's reusability across diverse MG flexible assets. It is noteworthy that the results and incremental developments will be publicly accessible through the SOGNO project [21] and disseminated by LFE, aligning with the authors' commitment to an open-source approach for advancing energy transition in the digitalized energy sector.

4. Illustrative example

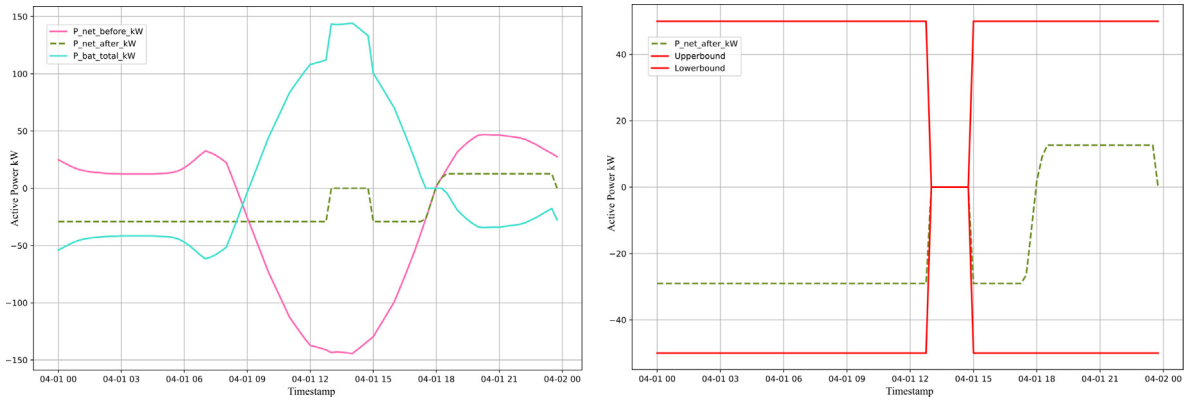
pymfm offers the capability to execute various scenarios across three primary UCs of UC1 (NRT operation mode with RB control logic), UC2 (SCH operation mode with RB control logic), and UC3 (SCH operation mode with OptB control logic). Both UC1 and UC2 are designed to operate with a single *cbes* unit. UC1 can additionally be associated with an NRT request for power delivery or reception, serving as a power boundary constraint for the MG. On the other hand, UC3 shows more flexibility and can manage either a single *cbes* unit or multiple storage units including *hbes* ones. Additionally, it can ensure achieving a target FSoC for *cbes*, facilitate bulk energy transactions with flexible storage units, curtail PV generation output, and finally maintain the MGs net power exchange within predetermined upper and lower limits.

Listing 1: Initialization and execution of the scheduling_optimization_based UC scenario.

```
    # Get the current directory of the script
fpath = os.path.dirname(os.path.abspath(__file__))

# Construct the file path for the input JSON file
filepath = os.path.join(fpath, "inputs/scheduling_optimization_based.json")

# Open and load the JSON data from the file
data = open_json(filepath)

# Create an InputData object from the loaded data
input_data = InputData(**data)

# Execute the control logic handler to process the input data
mode_logic, output_df, status = mode_logic_handler(input_data)
```

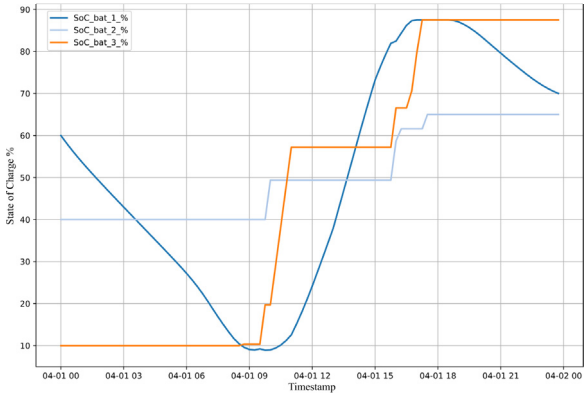
Table 1
Pymfm example UCs (x and o indicate respectively mandatory and optional entries).

UC	Operation mode		Control logic		Battery energy storage			Bulk	PV curtail	Power bound
	NRT	SCH	RB	OptB	cbes	FSoC	cbes&hbes			
1	x	–	x	–	x	–	–	–	–	o
2	–	x	x	–	x	–	–	–	–	–
3	–	x	–	x	o	o	o	o	o	o



(a) MG net power before and after control action according to the total storage flexibility power

(b) MG net power after control action vs power boundaries



(c) SoCs variations

Fig. 1. Simulation results visualization.

Therefore, from the sets of UCs shown in Table 1, the example script located at *src/pymfm/examples/control/scheduling_optimization_based.py* has been chosen to demonstrate the capabilities of the optimization-based control logic with the most extensive set of features that can be applied on the input parameters and profiles found in *src/pymfm/examples/control/inputs/scheduling_optimization_based.json*.

The illustrative code shown in Listing 1 demonstrates how to initialize and execute this example. First, the process of parsing JSON input data to create the necessary *InputData* object is initiated. Once the initialization is done, the *model_logic_handler* prepares the forecast, power boundary, and battery specification data and executes the *OptB.scheduling* control algorithm for this specific UC scenario. After completing the simulation and to analyze the simulated UC scenario, the code provided in Listing 2 is executed to export the simulation results to the corresponding JSON file and visualize them as different plots.

Fig. 1(a) shows how the optimum scheduling of the flexible storage units and their total (dis)charging powers can minimize the net power exchange of the MG before and after the control action according to

Listing 2: Exporting results and post-processing of the scheduling_optimization_based UC scenario.

```
    # Prepare and save control output data as JSON files
    data_output.prepare_json(mode_logic, output_df, output_directory="outputs/")

   # Visualize and save control output data as SVG plots
    data_output.visualize_and_save_plots(
        mode_logic, output_df, output_directory="outputs/"
    )
```

the total storage flexibility power setpoint (summation of dis/charging power setpoints). As depicted in Fig. 1(b), the power exchange after the control action is within the upper and lower power boundaries. The positive and negative power exchanges refer to respectively import and export of power while the zero power exchange highlights the islanding mode of the MG set by the zero upper and lower boundaries in the input JSON file. Finally, Fig. 1(c) shows the variations in SoCs of flexible storage units in accordance to their setpoints. It can be observed that the *cbes* unit (bat_1) reaches its predefined target FSoC by the end of the UC while the *hbes* units (bat_2 and bat_3) reach their maximum SoCs by the predefined end of the day at 18 o'clock.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work was supported in part by Platform for Operation of Distribution Networks (PLATONE) which is a European Project funded by the European Union's Horizon 2020 Research and Innovation Program under Grant 864300 and in part by the project Kopernikus Ensure Phase 3 which is funded by the German Federal Ministry for Education and Research under the funding code 03SFK1C0-3.

References

[1] IEEE standard for the specification of microgrid controllers, 2018, pp. 1–43, http://dx.doi.org/10.1109/IEEESTD.2018.8340204, IEEE Std 2030.7-2017.

[2] R. Moreno, D.N. Trakas, M. Jamieson, M. Panteli, P. Mancarella, G. Strbac, C. Marnay, N. Hatziargyriou, Microgrids against wildfires: Distributed energy resources enhance system resilience, IEEE Power Energy Mag. 20 (1) (2022) 78–89, http://dx.doi.org/10.1109/MPE.2021.3122772.

[3] pymfm in Python Package Index - PyPI, Python Software Foundation, URL https://pypi.org/project/pymfm/0.5.5/.

[4] C.R. Harris, et al., Array programming with NumPy, Nature 585 (7825) (2020) 357–362, http://dx.doi.org/10.1038/s41586-020-2649-2.

[5] 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, SciPy 1.0 Contributors, SciPy 1.0: Fundamental algorithms for scientific computing in Python, Nature Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.

[6] W.E. Hart, J.-P. Watson, D.L. Woodruff, Pyomo: Modeling and solving mathematical programs in Python, Math. Program. Comput. 3 (3) (2011) 219–260, http://dx.doi.org/10.1007/s12532-011-0026-8.

[7] L. Gurobi Optimization, Gurobi Optimizer Reference Manual, 2022, URL https://www.gurobi.com.

[8] T. Achterberg, SCIP: Solving constraint integer programs, Math. Program. Comput. 1 (1) (2009) 1–41, http://dx.doi.org/10.1007/s12532-008-0001-1.

[9] K. Bestuzheva, M. Besançon, W.-K. Chen, A. Chmiela, T. Donkiewicz, J. van Doornmalen, L. Eifler, O. Gaul, G. Gamrath, A. Gleixner, L. Gottwald, C. Graczyk, K. Halbig, A. Hoen, C. Hojny, R. van der Hulst, T. Koch, M. Lübbecke, S.J. Maher, F. Matter, E. Mühmer, B. Müller, M.E. Pfetsch, D. Rehfeldt, S. Schlein, F. Schlösser, F. Serrano, Y. Shinano, B. Sofranac, M. Turner, S. Vigerske, F. Wegscheider, P. Wellner, D. Weninger, J. Witzig, The SCIP Optimization Suite 8.0, Technical Report, Optimization Online, 2021, URL https://optimization-online.org/?p=18429.

[10] Anwendung der Repräsentativen VDEW-Lastprofile step by step, Python Software Foundation, URL https://www.bdew.de/energie/standardlastprofile-strom/.

[11] L. Thurner, A. Scheidler, F. Schäfer, J.-H. Menke, J. Dollichon, F. Meier, S. Meinecke, M. Braun, Pandapower—An open-source Python tool for convenient modeling, analysis, and optimization of electric power systems, IEEE Trans. Power Syst. 33 (6) (2018) 6510–6521, http://dx.doi.org/10.1109/TPWRS.2018.2829021.

[12] N. Hatziargyriou, Differences and synergies between local energy communities and microgrids, Oxford Open Energy 2 (2022) oiac013, http://dx.doi.org/10.1093/ooenergy/oiac013.

[13] A. Ahmadifar, M. Ginocchi, M.S. Golla, F. Ponci, A. Monti, Development of an energy management system for a renewable energy community and performance analysis via global sensitivity analysis, IEEE Access 11 (2023) 4131–4154, http://dx.doi.org/10.1109/ACCESS.2023.3235590.

[14] C. Jätz, B. Petters, N. Dult, A. Ahmadifar, A. Monti, Balancing PV generation in low voltage grids with limited data, in: 27th International Conference on Electricity Distribution (CIRED 2023), 2023, pp. 1255–1259, http://dx.doi.org/10.1049/icp.2023.0690.

[15] Platone, PLATform for operation of distribution networks, 2023, URL https://cordis.europa.eu/project/id/864300.

[16] SOGNO, Service-based Open-source Grid automation platform for Network Operation of the future, 2022, URL https://lfenergy.org/projects/sogno/.

[17] Platone, Platone educational video series, 2023, URL https://www.youtube.com/watch?v=d4E9ViHWnz4&list=PLgVFMwUk875O2YieP8EciRI8ViZoPigp6.

[18] Platone, Platone DSO technical platform, 2023, URL https://git.rwth-aachen.de/acs/public/deliverables/platone/dsptp.

[19] ENSURE, Neue EnergieNetzStruktURen für die Energiewende (New Energy Network Structures for the Energy Transition), 2019, URL https://www.kopernikus-projekte.de/projekte/ensure.

[20] E. Gümrükcü, A. Ahmadifar, A. Yavuzer, F. Ponci, A. Monti, Datafev—A Python framework for development and testing of management algorithms for electric vehicle charging infrastructures, Softw. Impacts 15 (2023) 100467, http://dx.doi.org/10.1016/j.simpa.2023.100467.

[21] SOGNO, SOGNO Platform on GitHub, 2023, URL https://github.com/sogno-platform/.