Software Impacts 9 (2021) 100092

Contents lists available at ScienceDirect

Software Impacts

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

Original software publication

# Gym-ANM: Open-source software to leverage reinforcement learning for power system management in research and education

Robin Henry [a,*], Damien Ernst [b]

[a] School of Engineering, Sanderson Building, The University of Edinburgh, Edinburgh, EH9 3FB, UK
[b] Department of Electrical Engineering and Computer Science, Montefiore Institute, University of Liège, B-4000 Liège, Belgium

ARTICLE INFO

*Keywords:*
Gym-ANM
Reinforcement learning
Active network management
Distribution networks
Renewable energy

ABSTRACT

Gym-ANM is a Python package that facilitates the design of reinforcement learning (RL) environments that model active network management (ANM) tasks in electricity networks. Here, we describe how to implement new environments and how to write code to interact with pre-existing ones. We also provide an overview of ANM6-Easy, an environment designed to highlight common ANM challenges. Finally, we discuss the potential impact of Gym-ANM on the scientific community, both in terms of research and education. We hope this package will facilitate collaboration between the power system and RL communities in the search for algorithms to control future energy systems.

Code metadata

| | |
|---|---|
| Current code version | 1.0.1 |
| Permanent link to code/repository used for this code version | https://github.com/SoftwareImpacts/SIMPAC-2021-56 |
| Permanent link to Reproducible Capsule | https://codeocean.com/capsule/2156962/tree/v1 |
| Legal Code License | MIT license (MIT) |
| Code versioning system used | git |
| Software code languages, tools, and services used | Python, JavaScript |
| Compilation requirements, operating environments & dependencies | Python 3.7 |
| If available Link to developer documentation/manual | https://gym-anm.readthedocs.io/en/latest/ |
| Support email for questions | robin@robinxhenry.com |

## 1. Introduction

Active network management (ANM) of electricity distribution networks is the process of controlling generators, loads, and storage devices for specific purposes (e.g., minimizing operating costs, keeping voltages and currents within operating limits) [1]. The modernization of distribution networks is taking place with the addition of distributed renewable energy resources and storage devices. This attempt to transition towards sustainable energy systems leaves distribution network operators (DNO) facing many new complex ANM problems (overvoltages, transmission line congestion, voltage coordination, investment issues, etc.) [2].

There is a growing belief that reinforcement learning (RL) algorithms have the potential to tackle these complex ANM challenges more efficiently than traditional optimization methods. This optimism results from the fact that RL approaches have been successfully and extensively applied to a wide range of fields with similarly difficult decision-making problems, including games [3–6], robotics [7–10], and autonomous driving [11–13].

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 addresses:* robin@robinxhenry.com (R. Henry), dernst@uliege.be (D. Ernst).

https://doi.org/10.1016/j.simpa.2021.100092
Received 18 May 2021; Accepted 27 May 2021

What games, robotics, and autonomous driving all have in common is that the environment in which the decisions have to be taken can be efficiently replicated using open-source software simulators. In addition, these software libraries usually provide interfaces tailored for writing code for RL research. Hence, the availability of such packages makes it easier for RL researchers to apply their algorithms to decision-making problems in these fields, without needing to first develop a deep understanding of the underlying dynamics of the environments with which their agents interact.

Put simply, we believe that ANM-related problems would benefit from a similar amount of attention from the RL community if open-source software simulators were available to model them and provide a simple interface for writing RL research code. With that in mind, we designed Gym-ANM, an open-source Python package that facilitates the design and the implementation of RL environments that model ANM tasks [14]. Its key features, which differentiate it from traditional power system modeling software (e.g., MATPOWER [15], pandapower [16]), are:

- Very little background in power system modeling is required, since most of the complex dynamics are abstracted away from the user.
- The environments (tasks) built using Gym-ANM follow the OpenAI Gym interface [17], with which a large part of the RL community is already familiar.
- The flexibility of Gym-ANM, with its different customizable components, makes it a suitable framework to model a wide range of ANM tasks, from simple ones that can be used for educational purposes, to complex ones designed to conduct advanced research.

Finally, as an example of the type of environment that can be built using Gym-ANM, we also released ANM6-Easy, an environment that highlights common ANM challenges in a 6-bus distribution network.

Both the Gym-ANM framework and the ANM6-Easy environment, including detailed mathematical formulations, were previously introduced in [14]. Here, our goal is to provide a short practical guide to the use of the package and discuss the impact that it may have on the research community.

## 2. The Gym-ANM package

The Gym-ANM package was designed to be used for two particular use cases. The first is the design of novel environments (ANM tasks), which requires writing code that simulates generation and demand curves for each device connected to the power grid (Section 2.1). The second use case is the training of RL algorithms on an existing environment (Section 2.2).

### 2.1. Design a Gym-ANM environment

The internal structure of a Gym-ANM environment is shown in Fig. 1. At each timestep, the agent passes an action $a_t$ to the environment. The latter generates a set of stochastic variables by calling the `next_vars()` function, which are then used along with $a_t$ to simulate the distribution network and transition to a new state $s_{t+1}$. Finally, the environment outputs an observation vector $o_{t+1}$ and a reward $r_t$ through the `observation()` and `reward()` functions.

The core of the power system modeling is abstracted from the user in the `next_state()` call. The gray blocks, `next_vars()` and `observation()`, are the only components that are fully customizable when designing new Gym-ANM environments.

In practice, new environments are created by implementing a subclass of `ANMEnv`. The general template to follow is shown in Listing 1. A more detailed description, along with examples, can be found in the online documentation.[1]

[1] https://gym-anm.readthedocs.io/en/latest/topics/design_new_env.html.

### 2.2. Use a Gym-ANM environment

A code snippet illustrating how a custom Gym-ANM environment can be used alongside an RL agent implementation is shown in Listing 2. Note that for clarity, this example omits the agent-learning procedure. Because Gym-ANM is built on top of the Gym toolkit [17], all Gym-ANM environments provide the same interface as traditional Gym environments, as described in their online documentation.[2]

## 3. Example: the ANM6-Easy environment

ANM6-Easy is the first Gym-ANM environment that we have released [14]. It models a 6-bus network and was engineered so as to highlight some of the most common ANM challenges faced by network operators. A screenshot of the rendering of the environment is shown in Fig. 2.

In order to limit the complexity of the task, the environment was designed to be fully deterministic: both the demand from loads (1: residential area, 3: industrial complex, 5: EV charging garage) and the maximum generation (before curtailment) profiles from the renewable energies (2: solar farm, 4: wind farm) are modeled as fixed 24-hour time series that repeat every day, indefinitely.

More information about the ANM6-Easy environment can be found in the online documentation.[3]

## 4. Research and educational impact

Many software applications exist for modeling steady-state power systems in industrial settings, such as PowerFactory [18], ERACS [19], ETAP [20], IPSA [21], and PowerWorld [22], all of which require a paid license. In addition, these programs are not well suited to conduct RL research since they do not integrate well with the two programming languages mostly used by the RL community: MATLAB and Python. Among the power system software packages that do not require an additional license and that are compatible with these programming languages, the commonly used in power system management research are MATPOWER (MATLAB) [15], PSAT (MATLAB) [23], PYPOWER (Python interface for MATPOWER) [24], and pandapower (Python) [16].

Nevertheless, using the aforementioned software libraries to design RL environments that model ANM tasks is not ideal. First, the user needs to become familiar with the modeling language of the library, which already requires a good understanding of the inner workings of the various components making up power systems and of their interactions. Second, these packages often include a large number of advanced features, which is likely to overwhelm the inexperienced user and get in the way of designing even simple ANM scenarios. Third, because these libraries were designed to facilitate a wide range of simulations and analyes, they often do so at the cost of solving simpler problems more slowly (e.g., simple AC load flows). Fourth, in the absence of a programming framework agreed upon by the RL research community interested in tackling energy system management problems, various research teams are likely to spend time and resources implementing the same underlying dynamics common to all such problems.

By releasing Gym-ANM, we hope to address all the shortcomings of traditional modeling packages described in the previous paragraph. Specifically:

- The dissociation between the design of the environment (Section 2.1) and the training of RL agents on it (Section 2.2) encourages collaboration between researchers experienced in power system modeling and in RL algorithms. Thanks to the general framework provided by Gym-ANM, each researcher may focus on their particular area of expertise (designing or solving the environment), without having to worry about coordinating their implementations.

[2] https://gym.openai.com/docs/.

[3] https://gym-anm.readthedocs.io/en/latest/topics/anm6_easy.html.

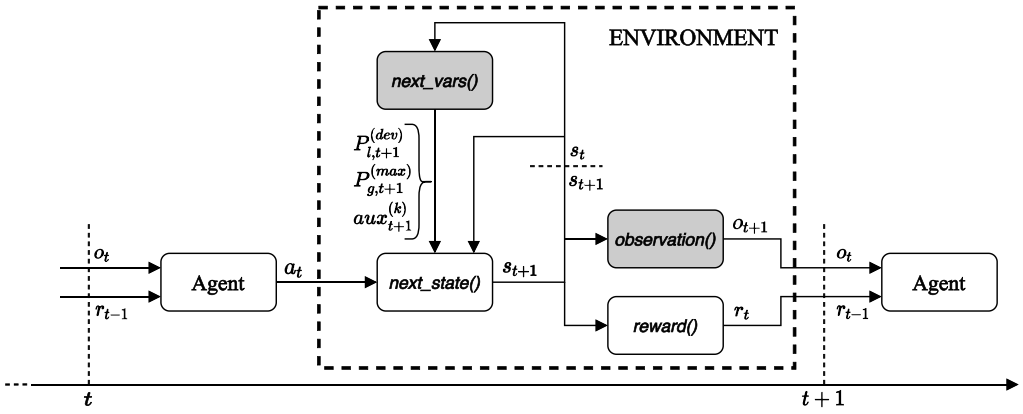


**Fig. 1.** Internal structure of a Gym-ANM environment.
*Source:* Taken from [14].

```
from gym_anm import ANMEnv

class CustomEnvironment(ANMEnv):
    def __init__(self):
        network = {'baseMVA': ..., 'bus': ...,
                   'device': ..., 'branch': ...}
                                  # power grid specs
        observation = ...         # observation space
        K = ...                   # number of auxiliary variables
        delta_t = ...             # timestep intervals
        gamma = ...               # discount factor
        lamb = ...                # penalty hyperparameter
        aux_bounds = ...          # bounds on auxiliary variable
        costs_clipping = ...      # reward clipping parameters
        seed = ...                # random seed

        super().__init__(network, observation, K, delta_t,
                         gamma, lamb, aux_bounds,
                         costs_clipping, seed)

    # Return an initial state vector s_0 ~ p_0(·).
    def init_state(self):
        ...
    # Return the next stochastic variables.
    def next_vars(self, s_t):
        ...
    # Return the bounds of the observation vector space.
    def observation_bounds(self):  # optional
        ...
```

**Listing 1:** Implementation template for new Gym-ANM environments.

- This dissociation also means that RL researchers are able to tackle the ANM tasks modeled by Gym-ANM environments without having to first understand the complex dynamics of the system. As a result, existing Gym-ANM environments can be explored by many in the RL community, from novices to experienced researchers. This is further facilitated by the fact that all Gym-ANM environments implement the Gym interface, which allows RL users to apply their own algorithms to any Gym-ANM task with little code modification (assuming they have used Gym in the past).
- Gym-ANM focuses on a particular subset of ANM problems. This specificity has two advantages. The first is that it simplifies the process of designing new environments, since only a few components need to be implemented by the user. The second is that, during the implementation of the package, it allowed us to focus on simplicity and speed. That is, rather than providing a large range of modeling features like most of the other packages, we focused on optimizing the computational steps behind the `next_state()` block of Fig. 1 (i.e., solving AC load flows). This effectively reduces the computational time required to train RL agents on environments built with Gym-ANM.

The simplicity with which Gym-ANM can be used by both the power system modeling and the RL communities has an additional advantage: it makes it a great teaching tool. This is particularly true for individuals interested in working at the intersection of power system management and RL research. One of the authors, Damien Ernst, has recently started incorporating the ANM6-Easy task in his RL course, *Optimal decision making for complex systems*, at the University of Liège [25].

```
env = gym.make('MyANMEnv')    # Initialize the environment.
obs = env.reset()             # Reset the env. and collect o_0.

for t in range(1, T):
  env.render()          # Update the rendering.
  a = agent.act(obs)    # Agent takes o_t as input and chooses a_t.
  obs, r, done, info = env.step(a)
              # The action a_t is applied, and are outputted:
              # - obs: the new observation o_{t+1},
              # - r: the reward r(s_t, a_t, s_{t+1}),
              # - done: True if s_{t+1} ∈ S^terminal,
              # - info: extra info about the transition.

env.close() # Close the environment and stop rendering.
```

**Listing 2:** A Python code snippet illustrating environment-agent interactions [14].

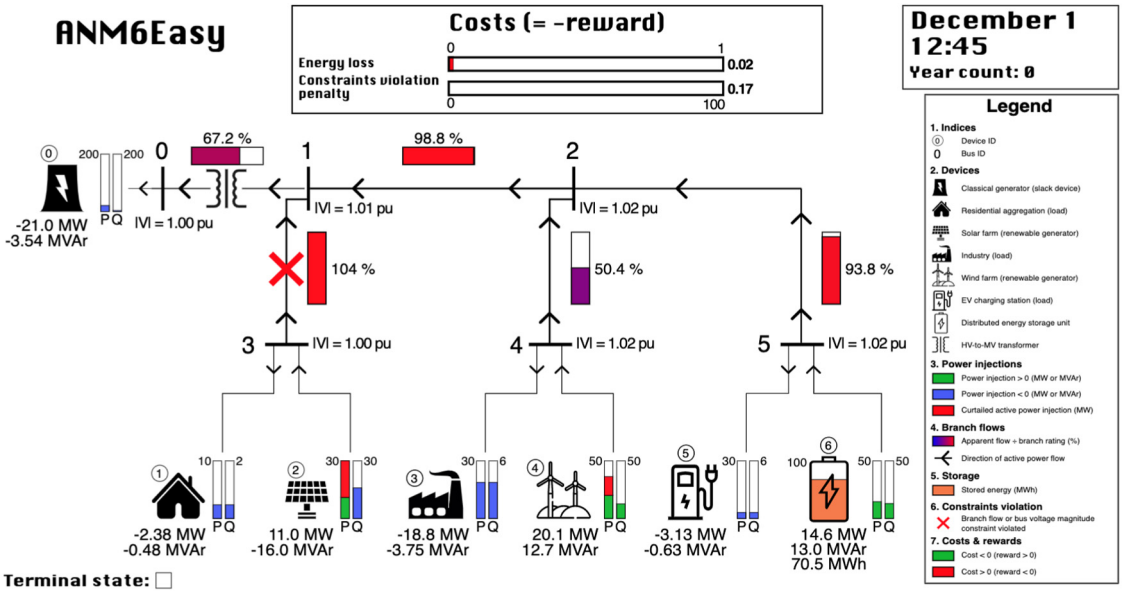


**Fig. 2.** The ANM6-Easy Gym-ANM environment.
*Source:* Taken from [14].

Finally, we also compared the performance of the soft actor-critic (SAC) and proximal policy optimization (PPO) RL algorithms against that of an optimal model predictive control (MPC) policy on the ANM6-Easy task in [14]. We showed that, with almost no hyperparameter tuning, the RL policies were already able to reach near-optimal performance. These results suggest that state-of-the-art RL methods have the potential to compete with, or even outperform, traditional optimization approaches in the management of electricity distribution networks. Of course, ANM6-Easy is only a toy example, and confirming this hypothesis will require the design of more complex and advanced Gym-ANM environments.

## 5. Conclusions and future works

In this paper, we discussed the usage of the Gym-ANM software package first introduced in [14], as well as its potential impact on the research community. We created Gym-ANM as a framework for the RL and energy system management communities to collaborate on tackling ANM problems in electricity distribution networks. As such, we hope to contribute to the gathering of momentum around the applications of RL techniques to challenges slowing down the transition towards more sustainable energy systems.

In the future, we plan to design and release Gym-ANM environments that more accurately model real-world distribution networks as opposed to that modeled by ANM6-Easy. However, we also highly encourage other teams to design and release their own Gym-ANM tasks and/or to attempt to solve existing ones.

## 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

We would like to thank Raphael Fonteneau, Quentin Gemine, and Sébastien Mathieu at the University of Liège for their valuable early feedback and advice, as well as Gaspard Lambrechts and Bardhyl Miftari for the feedback they provided as the first users of Gym-ANM.

## References

[1] S. Gill, I. Kockar, G.W. Ault, Dynamic optimal power flow for active distribution networks, IEEE Trans. Power Syst. 29 (1) (2013) 121–131.

[2] J. McDonald, Adaptive intelligent power systems: Active distribution networks, Energy Policy 36 (12) (2008) 4346–4351.
[3] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, M. Riedmiller, Playing atari with deep reinforcement learning, 2013, arXiv preprint arXiv:1312.5602.
[4] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, A. Graves, M. Riedmiller, A.K. Fidjeland, G. Ostrovski, et al., Human-level control through deep reinforcement learning, Nature 518 (7540) (2015) 529–533.
[5] D. Silver, A. Huang, C.J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al., Mastering the game of go with deep neural networks and tree search, Nature 529 (7587) (2016) 484.
[6] O. Vinyals, I. Babuschkin, W.M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D.H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al., Grandmaster level in StarCraft II using multi-agent reinforcement learning, Nature 575 (7782) (2019) 350–354.
[7] M.P. Deisenroth, G. Neumann, J. Peters, et al., A survey on policy search for robotics, Found. Trend. Robot. 2 (1–2) (2013) 1–142.
[8] P. Kormushev, S. Calinon, D.G. Caldwell, Reinforcement learning in robotics: Applications and real-world challenges, Robotics 2 (3) (2013) 122–148.
[9] J. Kober, J.A. Bagnell, J. Peters, Reinforcement learning in robotics: A survey, Int. J. Robot. Res. 32 (11) (2013) 1238–1274.
[10] S. Gu, E. Holly, T. Lillicrap, S. Levine, Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates, in: 2017 IEEE International Conference on Robotics and Automation (ICRA), IEEE, 2017, pp. 3389–3396.
[11] A.E. Sallab, M. Abdou, E. Perot, S. Yogamani, Deep reinforcement learning framework for autonomous driving, Electron. Imaging 2017 (19) (2017) 70–76.
[12] M. O'Kelly, A. Sinha, H. Namkoong, R. Tedrake, J.C. Duchi, Scalable end-to-end autonomous vehicle testing via rare-event simulation, in: Advances in Neural Information Processing Systems, 2018, pp. 9827–9838.
[13] D. Li, D. Zhao, Q. Zhang, Y. Chen, Reinforcement learning and deep learning based lateral control for autonomous driving [application notes], IEEE Comput. Intell. Magaz. 14 (2) (2019) 83–98.
[14] R. Henry, D. Ernst, Gym-ANM: Reinforcement learning environments for active network management tasks in electricity distribution systems, 2021, arXiv preprint arXiv:2103.07932.
[15] R.D. Zimmerman, C.E. Murillo-Sánchez, R.J. Thomas, MATPOWER: Steady-state operations, planning, and analysis tools for power systems research and education, IEEE Trans. Power Syst. 26 (1) (2010) 12–19.
[16] L. Thurner, A. Scheidler, F. Schäfer, J. 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.
[17] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, W. Zaremba, Openai gym, 2016, arXiv preprint arXiv:1606.01540.
[18] F.M. Gonzalez-Longatt, J.L. Rueda, PowerFactory Applications for Power System Analysis, Springer, 2014.
[19] H. Langley, K. Wright, ERACS-a comprehensive package for PCs, in: IEE Colloquium on Interactive Graphic Power System Analysis Programs, IET, 1992, p. 3/1-3/7.
[20] K. Brown, F. Shokooh, H. Abcede, G. Donner, Interactive simulation of power systems: ETAP applications and techniques, in: Conference Record of the 1990 IEEE Industry Applications Society Annual Meeting, IEEE, 1990, pp. 1930–1941.
[21] TNEI, Interactive Power System Analysis (IPSA) software, https://www.ipsa-power.com, (Accessed on 05/12/2021).
[22] PowerWorld Corporation, PowerWorld software, https://www.powerworld.com, (Accessed on 05/12/2021).
[23] F. Milano, An open source power system analysis toolbox, IEEE Trans. Power Syst. 20 (3) (2005) 1199–1206.
[24] R. Lincoln, PYPOWER library, https://github.com/rwl/PYPOWER, (Accessed on 05/12/2021).
[25] D. Ernst, Optimal decision making for complex problems course at the University of Liège, http://blogs.ulg.ac.be/damien-ernst/info8003-1-optimal-decision-making-for-complex-problems, (Accessed on 05/13/2021).