pyTEP

A Python package for interactive simulations of the Tennessee Eastman process

Reinartz, Christopher; Enevoldsen, Thomas T.

Published in:
SoftwareX

Link to article, DOI:
10.1016/j.softx.2022.101053

Publication date:
2022

Document Version
Publisher's PDF, also known as Version of record

SoftwareX 18 (2022) 101053

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

pyTEP: A Python package for interactive simulations of the Tennessee Eastman process

Christopher Reinartz *, Thomas T. Enevoldsen

Automation and Control Group, Department of Electrical Engineering, Technical University of Denmark, Elektrovej 326, 2800 Kgs. Lyngby, Denmark

ARTICLE INFO

Article history:
Received 13 April 2021
Received in revised form 13 January 2022
Accepted 15 March 2022

Keywords:
Tennessee Eastman process
Chemical process simulation
Simulation framework
Python

ABSTRACT

pyTEP is an open-source simulation API for the Tennessee Eastman process in Python. It facilitates the setup of complex simulation scenarios and provides the option of interactive simulation. The Tennessee Eastman process has been the go-to benchmark for statistical process monitoring and machine learning based fault-detection approaches for continuous chemical processes in recent years, but its potential outside these domains remains largely untapped. Existing simulators are tailored towards simulations of stationary operating conditions in the presence of faults, but further extensions for more complex simulation scenarios are time-consuming, which may discourage researchers from adopting the process. Through pyTEPs API, users can configure simulations, change operating conditions and store simulation data without being exposed to the underlying mechanics of the simulator. In addition to the newly introduced features, pyTEP promises more versatility and more straightforward usage than existing TEP simulators.

Code metadata

Current code version	v0.1.0
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-21-00069
Legal Code License	BSD 3-Clause License
Code versioning system used	git
Software code languages, tools, and services used	Python3.7, MATLAB
Compilation requirements, operating environments & dependencies	MATLAB/Simulink (R2019a-R2020b) and Python packages listed in requirements
Link to developer documentation/manual	https://pytep.readthedocs.io/en/latest/
Support email for questions	ccrein@elektro.dtu.dk

1. Motivation and significance

Safe and profitable operation of complex industrial plants at a large-scale is one of the significant research fields within the chemical- and petrochemical engineering communities. Two open research areas are the detection and identification of process faults and the automated generation of plans for process reconfiguration. Such research requires access to sufficient amounts of healthy and faulty process data and, for the study of reconfigurations, data that describes the behaviour of the process under varying operating conditions.

Complex, high fidelity simulators can greatly aid such research, because they provide the option to simulate faults and test experimental process configurations with varying tuning parameters in a safe environment. Due to the benefits mentioned above, sophisticated process simulators have gained popularity, both in industry as well as in multiple research domains. For high-fidelity simulations of wind-turbines in both the testing and design phase, the National Renewable Energy Laboratory present [1]. For the simulation of complex finite element method (FEM) computations in the fields of electromagnetics and fluid dynamics, [2] present a coupling of the software packages Elmer and OpenFOAM. Such simulators profit from being easily accessible and extendable, ideally in a programming language that is adopted by researchers in the primary application domain of the software. Recently, Python based frameworks have gained popularity, due to both the available ecosystem and large open-source communities. The high degree of adoption of Python as

* Corresponding author.
E-mail address: ccrein@elektro.dtu.dk (Christopher Reinartz).

https://doi.org/10.1016/j.softx.2022.101053

a primary analysis tool, especially for machine learning centred projects, provides a strong incentive to adapt existing research software and simulators to be easily accessible through a Python framework. [3] present a Python library that interacts with the OpenSees simulation framework, identifying the need of users to access OpenSees through alternative languages to expand upon existing use cases. A Python interface for the Dutch Atmospheric Large Eddy Simulation (DALES) suite, originally written in Fortran, is presented by [4] in order to facilitate access to complex features of the original suite. [5] present a framework that bridges the gap between CoppeliaSim, an existing robot simulation environment, and a virtual reality (VR) toolbox, thus allowing users to rapidly explore and study human–computer interactions using the VR support added to the simulation environment.

In chemical engineering research, the Tennessee Eastman process simulator (TEP) serves as the go-to benchmark for the evaluation of fault-detection and isolation schemes for continuous processes. Current implementations of the simulator are available as FORTRAN and MATLAB/Simulink packages. Following the original publication of TEP by [6], the simulator was primarily used to study plant-wide control design [7–9] and plant optimization approaches [10]. The most widely adopted control schemes by [7,8] were integrated into the original simulator and made publicly available at [11,12], respectively. The Tennessee Eastman Challenge Archive [11] features multiple iterations of TEP simulators, the latest of which [13] introduces a significantly extended set of simulation features, such as additional process faults and the ability to scale fault magnitudes. More recently, an implementation of the Tennessee Eastman process in Modelica has been proposed by [14], including a validated implementation of the process in open-loop. In order to facilitate access to TEP process data, the existing simulators were used to generate datasets [12,15–17] to provide benchmarks for fault-detection, isolation and alarm management studies. All of the mentioned datasets primarily feature single-fault simulations in a specific operating mode of TEP that are well suited to test the performance of data-driven fault detection approaches. [15] include five additional operating modes as well as healthy data of transient process behaviour. The majority of current research utilizes these datasets, because accessing the data directly is significantly more convenient than using one of the currently available simulators, but both the datasets and simulators have been used extensively in research papers in recent years. Examples of applying fault detection were investigated by [18], who utilized convolutional networks, [19], who investigated Kantorovich Distance Independent Component Analysis and [20,21], who applied Gaussian process regression. [22] investigated the base operating mode of TEP using logical analysis of data (LAD), a machine-learning based classification technique. [23] used an adaptive threshold scheme to replace traditional, fixed control limits by applying a combination of multivariate and univariate statistical analysis. It is observed that the current state-of-the-art using TEP is mostly limited to fault-detection and isolation, for which comprehensive datasets are available or for which the current simulators offer simple means of data generation. [24] presented work that concerns optimization of TEP in the presence of faults, which is a topic that remains largely unexplored, because of the limitations of current TEP simulators. Existing simulators, including the latest and most sophisticated simulator by [13], are designed for the simulation of process faults at otherwise stationary process configurations, but lack flexibility otherwise. Changes to the process operating conditions currently necessitate direct changes to the Simulink model.

1.1. Novelty and contribution

The presented Python interface for the Tennessee Eastman simulator is a wrapper for a modified version of the existing MATLAB/Simulink simulator presented by [13]. The underlying Simulink model is extended to enable dynamic process re-configurations and to allow interactive pausing and data retrieval during simulations. A high-level Python API provides methods that enable the user to modify process setpoints, enable preset faults and pause and continue the simulation without the need to modify the underlying simulation model. Simulation data is processed and stored as pandas DataFrames in order to facilitate the re-use and export to other formats. The main contributions of the presented work are the direct integration of the Tennessee Eastman simulator into a user-friendly Python framework and the newly added feature that allows interactive simulations.

2. Software description

pyTEP is written in Python3.7 and accesses an extended version of the Simulator by [13], tested in MATLAB/Simulink (2019a-2020b). The package is installable from PyPI using `pip install pytep`. Further documentation can be found in the link to the developer manual in Table Code Metadata. Communication between the Python modules and the simulator is handled using the MATLAB Engine for Python [25]. The software is distributed under the BSD License to allow for continuous application, modification and extension by interested parties.

2.1. Architecture and functionalities

The software integrates two main Python classes, a set of MATLAB functions and a Simulink model. All simulation settings, commands and data queries can be set through the high-level Python API, namely the `SimInterface` class (see Fig. 1).

2.1.1. Simulink model

The Simulink model presented by [13] is extended in two ways. In order to allow simulations of non-stationary operating conditions, previously hard-coded controller setpoints ('Constant' input blocks) are replaced by step- or ramp-blocks, depending on the chosen setting. Each setpoint input block requires four parameters; initial and final values for the setpoint, the duration of the change and the simulation time at which the change is initiated. In order to enable timed start, stop, and pause commands from the command line, an assertion block is added, which triggers if the simulation time is larger than a given parameter. Triggering the assertion block pauses the current simulation and updates the MATLAB workspace.

2.1.2. MATLAB utility functions

In order to minimize the amount of internal MATLAB/Simulink code that is exposed to the Python API, a set of MATLAB utility functions is added to encapsulate pure MATLAB/Simulink functionality. Excluding generic queries to the MATLAB workspace, pyTEPs low-level API issues commands and queries data through the utility functions.

2.1.3. Low-level API

The `MatlabBridge` class is the primary, and only, interface between the MATLAB/Simulink model and the Python modules. Its purpose is to provide the functionalities added by the MATLAB utility functions and the Simulink model inside Python. All settings can and should be set via the `MatlabBridge`, rendering direct manipulation of the Simulink model and MATLAB workspace in order to change the simulation settings unnecessary.

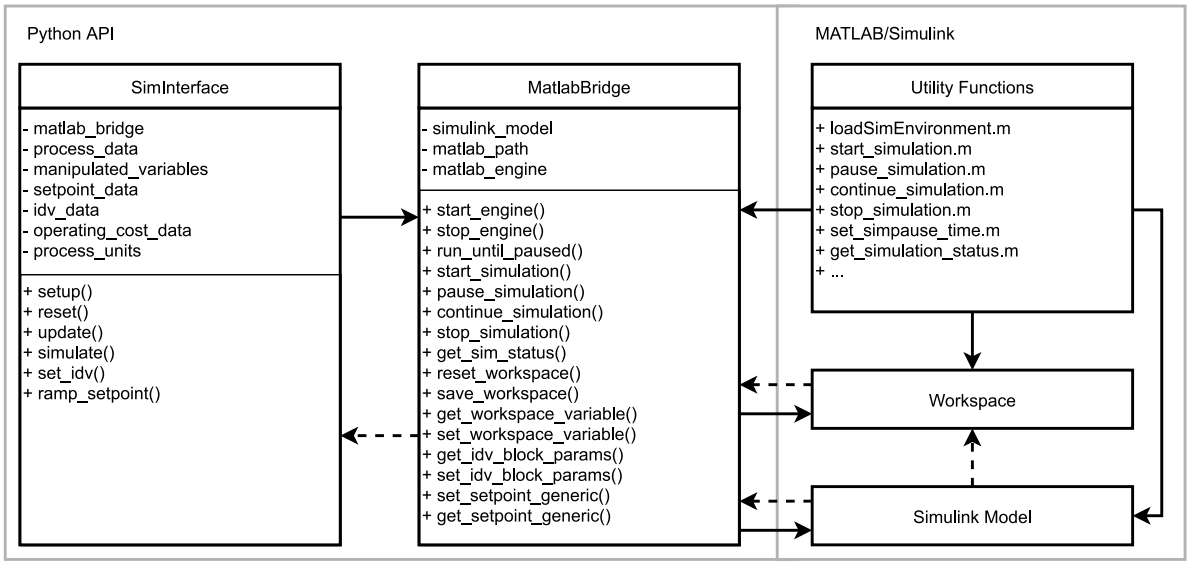

Fig. 1. Dependencies between pyTEP modules. Plain edges signify the direction of command and dashed edges the direction of simulation data flow.

2.1.4. High-level API

The `SimInterface` class is pyTEPs high-level API and the main access point for users. Through the `SimInterface`, users can load the simulation environment, set simulation parameters, run the simulation interactively, and query and save the current state as well as the state history if required. Many of the functions in the `SimInterface` are meant to facilitate the generation of simulation scenarios and datasets through pyTEP by extending functions in the `MatlabBridge`. Amongst others, generating simulation scenarios is facilitated by accounting for the current simulation time when setting inputs for setpoints and fault signatures. Further features include the automatic reset of the operating condition to a default setting and the export of the current scenario data. Data that can be queried from the `SimInterface` include the process measurements, manipulated variables, setpoints, operating cost, fault-signatures and the respective units of these variables. Simulation output is updated and appended to pandas Dataframes in the `SimInterface` whenever the Simulink model pauses or terminates.

2.1.5. Interactive simulations

On-line re-configuration of the Tennessee Eastman process requires the ability to pause an active simulation and change process inputs (the control setpoints) at run-time based on the current measured state of the process. The simulator by [13] enables the user to run a simulation and save its final state upon completion. It was however observed that initializing a new simulation with this final state produces a slight discontinuity in the process variables. The magnitude of this discontinuity is negligible, but it could potentially result in erroneous results if simulations are frequently terminated and re-initialized. To avoid this, pyTEPs default behaviour is to pause the simulation after the specified simulation time has passed using an Assertion block in the Simulink model. Continuation of the simulation from a paused state does not introduce the aforementioned discontinuity, resulting in identical results for interactive and non-interactive simulations. When simulating with pyTEP, users can choose to extend the active simulation by calling the `simulate` method again, as shown in the example in Section 3.2. By default, simulations remain active in a paused state until the simulation environment is reset, so that extensions of active simulations do not lead to discontinuities in the results. The total length and the rate at which output is updated are configurable simulation parameters. Output can be updated up to every three minutes, although the authors recommend lower rates for most applications, as explained in Section 2.1.6.

2.1.6. Performance

Since PyTEP is a wrapper for an existing simulator, it inevitably introduces computational overhead. The main factors that could affect performance are the utilization of the MATLAB engine for Python and the interactive use of the simulator, especially in relation to the frequency at which simulation data is updated. In order to quantify the effect of the introduced overhead, the performance of simulations with a duration of 50 and 100 simulation hours (durations used in current TEP datasets) in MATLAB and pyTEP is compared. Using pyTEP instead of interacting with the Simulink model through the baseline simulator resulted in no noticeable difference in performance for non-interactive simulations, with consistent computation times of approximately 21 s and 42 s for 50 h and 100 h of simulation, respectively. Running the simulation in pyTEP interactively, meaning pausing and resuming the simulation at fixed intervals, increases computation time as shown in Fig. 2. It can be observed that the performance starts decreasing significantly if the simulation is paused and evaluated more frequently than once per simulated hour and that this effect scales with the absolute simulation time. Considering that changes of the process operating conditions will typically take significantly more than one hour to take effect due to the slow process dynamics of TEP, we currently see no incentive to pause the simulation more often than once per hour for most practical applications. If a specific application should require more frequent pausing, we recommend setting the frequency as conservatively as possible in order to avoid unnecessary loss of performance.

3. Illustrative examples

This section presents two examples that highlight the utility of pyTEP for the generation of a reference dataset featuring changes of the process operating point, and for the integration of a planning framework with TEP for on-line process re-configurations.

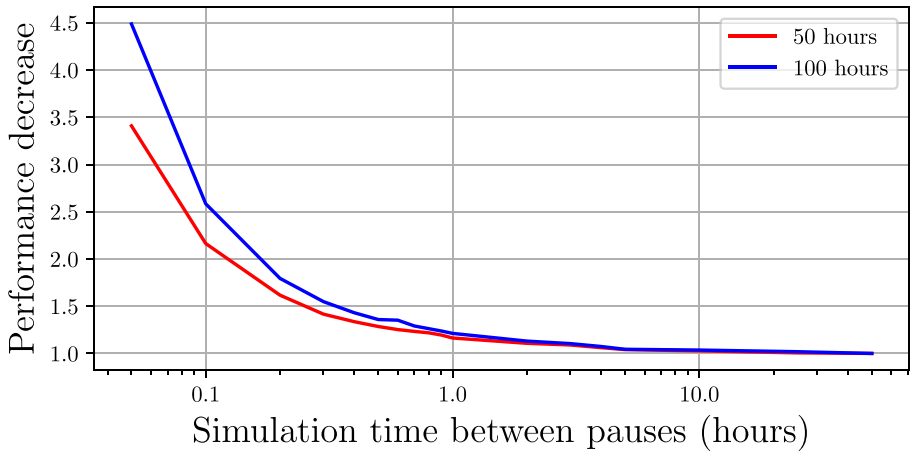

Fig. 2. Relative computation time (computation time/smallest measured computation time) for interactive simulations with different pause-intervals.

```python
from pytep.siminterface import SimInterface
from itertools import product
from pathlib import Path

tep = SimInterface.setup()

save_dir = Path("./example_save").absolute()
durations = [0, 10, 20, 30, 40]
gains = [0.85, 0.9, 0.95, 1, 1.05, 1.1, 1.15]
for sp in tep.setpoint_labels:
    sp_value = tep.current_setpoint_value(sp)
    for d, g in product(durations, gains):
        sp_target = g * sp_value
        tep.reset()
        tep.ramp_setpoint(sp, sp_target,
                          duration=d, delay=30)
        tep.simulate(duration=100)
        tep.save_all(save_dir)
```

Listing 1: Generation of part of the reference dataset by [15] using pyTEP

3.1. Dataset generation

The code snippet presented in Listing 1 shows how pyTEP can be used to generate part of the reference dataset presented by [15], which features setpoint changes of varying magnitudes and profiles in TEP. Setpoint profiles featured in the dataset include step changes as well as ramp changes with durations of 10 to 40 h and magnitude changes within a 15 percent interval of the initial setpoint value. Previously, generating this dataset required manual modification of the Simulink model as well as multiple MATLAB scripts to correctly setup the simulation environment, adjust the simulation parameters for each run and post-process the generated simulation output.

3.2. Online re-configuration

Listing 2 details the use of pyTEP to investigate on-line process re-configurations, directly leveraging the newly introduced feature that simulations can be paused, modified and continued based on an evaluation of the available state history of the simulated process. Since the planning tool is not subject of this contribution, it will only be introduced to the degree that is necessary to comprehend the presented example. At each iteration of the while loop, the planner compares the current state of the process to a pre-defined goal-state and determines an optimal action, which is then executed for one hour of simulation time, before the planner re-evaluates. The chosen action is a suggested process modification in the form of a setpoint change, including at least a new reference value for the setpoint. If the current state of the process happens to lie within a pre-defined goal set, the planner returns that no further action is required and declares the re-configuration complete.

4. Impact

Section 1 lists multiple reasons why simulation based analysis is of vital importance to a variety of research fields and why the Tennessee Eastman process has a pivotal role in chemical engineering and generic analysis of complex large-scale processes. In order for research that utilizes TEP as a reference to be conducted efficiently, it is vital that TEP simulators provide all features required for the given line of research with high utility, where the utility of a feature is defined as a combination of the provided functionality and its accessibility. As summarized in previous sections, existing TEP simulators generally exhibit high functionality but very low accessibility.

With pyTEP, we address this issue, focusing on the main shortcoming that many lines of research require substantially more varied data than can be generated with existing simulators. The authors believe that researchers must be able to generate

```python
from pytep.siminterface import SimInterface
from planning.tep_planner import TEPPlanner

tep = SimInterface.setup()
plnnr = TEPPlanner()

save_dir = Path("./example_save").absolute()
tep_state = tep.current_process_data()
while not plnnr.goal_reached(tep_state):
    best_action = plnnr.plan(tep_state)
    tep.ramp_setpoint(**best_action)
    tep.simulate(duration=1)
    tep_state = tep.current_process_data()
tep.save_all(save_dir)
```

Listing 2: On-line process re-configuration using pyTEP in combination with a planning tool for TEP.

data tailored to their needs without extensive modification of the simulation tool and that any redundant effort towards simulator development should instead be spent on the research itself.

To summarize the points that were stated more exhaustively in the previous sections, it can be stated that pyTEP improves upon the current state of TEP simulation tools in three major ways. Firstly, the set of features that is available 'out-of-the-box' is extended significantly by providing the functionality to run interactive simulations with transient inputs and disturbance profiles. Secondly, it provides a high-level API that allows users to set-up configurations, simulate and save data to an easily accessible data format without being exposed to the underlying mechanics of the simulation. Previous simulators do either not provide APIs or do not specify which part of the provided interface is meant for pure users and which for developers, who want to make modifications to the simulator in addition to using it. Thirdly, pyTEP is hosted in a shared development environment, which allows interested parties to collaborate upon future improvements and thus hopefully minimize redundancy related to development of the simulator. While this third point should no longer be of note in the present day, it needs to be pointed out that, to the authors knowledge, none of the TEP simulators listed in Section 1 are hosted in a way that would allow collaborative work. Additionally, the listed MATLAB/Simulink simulators are not as well suited to e.g. git-based collaboration environments in cases where a significant part of the development involves modifications of the Simulink-model.

5. Conclusion

We have developed a Python package named pyTEP, which provides a comprehensive API for the Tennessee Eastman simulation software, featuring high-level functions for simulation configuration, modification, execution and data retrieval. The current version of pyTEP provides a foundation for a more advanced and versatile utilization of the Tennessee Eastman process by simplifying data generation and monitoring, and an interface in one of the most extensively used programming languages; Python. It further allows interactive simulations of TEP, enabling on-line process modification based on process feedback.

6. Active development

pyTEP is being developed continuously in order to improve and extend its functionalities. By making it available to the public now, the authors hope to spark interest in other research groups and are open to collaborative development in order to make pyTEP as useful as possible. Current work on pyTEP is focused on the development of a dash-based [26] graphical user-interface that features all basic functionalities provided by pyTEPs high-level API, allowing users to explore TEP through a GUI without any additional coding effort. The interface is meant to enable the user to control the simulation interactively and make decisions about proper setpoint trajectories for controlled variables based on process feedback, allowing him/her to emulate the role a control room operator. We believe this dashboard can be a crucial tool for investigating and developing decision support systems for the Tennessee Eastman process.

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.

Acknowledgement

This research was funded by the Danish Hydrocarbon Research and Technology Centre, Denmark.

References

[1] Openfast. 2022, Accessed January 2022, URL https://github.com/OpenFAST/openfast.

[2] Vencels J, Råback P, Geža V. EOF-library: Open-source elmer FEM and OpenFOAM coupler for electromagnetics and fluid dynamics. SoftwareX 2019;9:68–72.

[3] Zhu M, McKenna F, Scott MH. OpenSeesPy: Python library for the OpenSees finite element framework. SoftwareX 2018;7:6–11.

[4] van den Oord G, Jansson F, Pelupessy I, Chertova M, Grönqvist JH, Siebesma P, Crommelin D. A python interface to the dutch atmospheric large-eddy simulation. SoftwareX 2020;12:100608.

[5] Bogaerts B, Sels S, Vanlanduit S, Penne R. Connecting the CoppeliaSim robotics simulator to virtual reality. SoftwareX 2020;11:100426.

[6] Downs JJ, Vogel EF. A plant-wide industrial process control problem. Comput Chem Eng 1993;17(3):245–55. http://dx.doi.org/10.1016/0098-1354(93)80018-I, arXiv:1722.

[7] Ricker NL. Decentralized control of the Tennessee eastman challenge process. J Process Control 1996;6(4):205–21. http://dx.doi.org/10.1016/0959-1524(96)00031-5.

[8] Lyman PR, Georgakis C. Plant-wide control of the Tennessee eastman problem. Comput Chem Eng 1995;19(3):321–31. http://dx.doi.org/10.1016/0098-1354(94)00057-U.

[9] Larsson T, Hestetun K, Hovland E, Skogestad S. Self-optimizing control of a large-scale plant: The Tennessee eastman process. Ind Eng Chem Res 2001;40(22):4889–901. http://dx.doi.org/10.1021/ie000586y.

[10] Ricker NL. Optimal steady-state operation of the Tennessee eastman challenge process. Comput Chem Eng 1995;19(9):949–59. http://dx.doi.org/10.1016/0098-1354(94)00043-N, URL http://dx.doi.org/10.1016/0098-1354(94)00043-N.
[11] Ricker NL. Tennessee eastman challenge archive. 2020, URL https://depts.washington.edu/control/LARRY/TE/download.html.
[12] Braatz RD. Tennessee eastman problem simulation data. 2020, URL http://web.mit.edu/braatzgroup/links.html.
[13] Bathelt A, Ricker NL, Jelali M. Revision of the Tennessee eastman process model. IFAC-PapersOnLine 2015;28(8):309–14. http://dx.doi.org/10.1016/j.ifacol.2015.08.199.
[14] Martin-Villalba C, Urquia A, Shao G. Implementations of the Tennessee eastman process in modelica. IFAC-PapersOnLine 2018;51(2):619–24. http://dx.doi.org/10.1016/j.ifacol.2018.03.105, 9th Vienna International Conference on Mathematical Modelling, URL https://www.sciencedirect.com/science/article/pii/S2405896318301095.
[15] Reinartz C, Kulahci M, Ravn O. An extended Tennessee eastman simulation dataset for fault-detection and decision support systems. Comput Chem Eng 2021;149:107281. http://dx.doi.org/10.1016/j.compchemeng.2021.107281, URL https://www.sciencedirect.com/science/article/pii/S0098135421000594.
[16] Manca G. "Tennessee-Eastman-Process" Alarm Management Case Study. IEEE Dataport; 2020, http://dx.doi.org/10.21227/326k-qr90.
[17] Rieth C, Amsel B, Tran R, Cook M. Additional Tennessee eastman process simulation data for anomaly detection evaluation. 2017, http://dx.doi.org/10.7910/DVN/6C3JR1.
[18] Dong Y, Qin SJ. Dynamic latent variable analytics for process operations and control. Comput Chem Eng 2018;114. http://dx.doi.org/10.1016/j.compchemeng.2017.10.029.
[19] Kini KR, Madakyaru M. Improved process monitoring strategy using kantorovich distance-independent component analysis: An application to Tennessee eastman process. IEEE Access 2020;8:205863–77. http://dx.doi.org/10.1109/access.2020.3037730.
[20] Fezai R, Mansouri M, Abodayeh K, Nounou H, Nounou M. Online reduced gaussian process regression based generalized likelihood ratio test for fault detection. J Process Control 2020;85:30–40. http://dx.doi.org/10.1016/j.jprocont.2019.11.002.
[21] Beena AM, Pani AK. Fault detection of complex processes using nonlinear mean function based Gaussian process regression: Application to the Tennessee eastman process. Arab J Sci Eng 2020. http://dx.doi.org/10.1007/s13369-020-05052-x.
[22] Ragab A, El-Koujok M, Poulin B, Amazouz M, Yacout S. Fault diagnosis in industrial chemical processes using interpretable patterns based on logical analysis of data. Expert Syst Appl 2018;95. http://dx.doi.org/10.1016/j.eswa.2017.11.045.
[23] Bakdi A, Kouadri A. An improved plant-wide fault detection scheme based on PCA and adaptive threshold for reliable process monitoring: Application on the new revised model of Tennessee eastman process. J Chemometrics 2018;32(5):1–16. http://dx.doi.org/10.1002/cem.2978.
[24] Capaci F, Vanhatalo E, Kulahci M, Bergquist B. The revised Tennessee eastman process simulator as testbed for SPC and DoE methods. Qual Eng 2018;1–18. http://dx.doi.org/10.1080/08982112.2018.1461905.
[25] MATLAB engine for python. 2022, Accessed January 2022, URL http://mathworks.com/help/matlab/matlab_external/install-the-matlab-engine-for-python.html.
[26] Plotly - dash. 2022, Accessed January 2022, URL https://plotly.com/dash/.