ProgPy: Python Packages for Prognostics and Health Management of Engineering Systems

Christopher Teubert[1], **Katelyn Jarvis**[1], **and Matteo Corbetta**[2]

1 NASA Ames Research Center, United States **2** KBR, Inc.

DOI: 10.21105/joss.05099

Software

- Review
- Repository
- Archive

Editor: Kelly Rowland

Reviewers:

- @tbsexton
- @nkrusch

Submitted: 12 December 2022
Published: 05 July 2023

Summary

Prognostics of engineering systems or systems of systems is the prediction of future performance and/or the time at which one or more events of interest occur. Prognostics can be applied in a variety of applications, from spacecraft and aircraft to wind turbines, oil and gas infrastructure, and assembly lines. Prognostic results are used to inform action to extend life or prevent failures through changes in use or predictive maintenance.

The NASA Prognostics Python Packages (ProgPy)(Teubert et al., 2022) are a set of open-sourced Python packages supporting research and development of prognostics and health management for engineering systems, as described in (Goebel et al., 2017). ProgPy builds upon the architecture of the Matlab Prognostics Libraries (Daigle, 2016c, 2016a, 2016b), Generic Software Architecture for Prognostics (Teubert et al., 2017), and Prognostics As-A-Service (Watkins et al., 2019). ProgPy implements architectures and common functionalities of prognostics, supporting both researchers and practitioners.

Statement of need

Prognostics and Health Management (PHM) is a fast-growing field. Successful PHM application can reduce operational costs and prevent failure, making systems safer. There has been limited application of prognostics in practice. This is partially because prognostics is a multi-faceted and complex problem, including data availability, sensor design and placement, and, of interest to us, software.

Often, software is written for an ad-hoc single prognostic application and cannot be transferred to others, or is limited in scope. A few related packages are described here. Simantha is a discrete manufacturing system simulation package that simulates degradation, but it is limited to a Discrete-Time Markov Chain and doesn't include prognostic capabilities (Dadfarnia & Drozdov, 2013). Lifelines is a survival analysis tool that can be used for reliability analysis to establish fixed-interval maintenance schedules, a different problem than that solved by ProgPy (Davidson-Pilon, 2022). Pomegranate is a Python probabilistic modeling package effective for data science applications (Schreiber, 2017). However, Pomegranate does not include explicit state estimation capabilities, prognostics tools, or physics-based degradation modeling features. Finally, there are a number of general machine-learning packages such as TensorFlow, scikit learn, and PyTorch. These are general tools that can be used for diagnostics and prognostics, but are not designed specifically for that application.

There is a need for a foundational set of efficient tools to enable new PHM technologies.

ProgPy provides a set of support packages for individuals researching and developing prognostic technologies. ProgPy consists of three packages: prog_models, prog_algs, and prog_server. prog_models provides tools aiding the development, evaluation, simulation, and tuning of prognostic models, whether physics-based or data-driven. prog_models also supports downloading select relevant datasets (Bole et al., 2014) (Saxena & Goebel, 2008). prog_algs supports uncertainty representation, state estimation, prognostics, the evaluation and visualization of prognostic results, and the creation of new prognostic algorithms. prog_server is a Service-Oriented Architecture for prognostics and state estimation. prog_server is also distributed with a Python client, prog_client.

The following sections describe some ways ProgPy could be used. There are many features and use cases beyond those illustrated here. See the ProgPy documentation for more information.

Selected use case: building and simulating models

One of the primary use-cases of ProgPy is building new models. Prognostic models are created by subclassing the PrognosticsModel class. Users can copy the model template as a starting point, replacing the representative member functions with model logic.

Prognostic models have inputs (load/control applied to a system), internal states, outputs (measurable quantities), and events of interest (what we're predicting). Logic of a prognostic model is defined using the state transition (dx or next_state), output, event_state, and threshold_met functions.

In the below example, a user creates a physics-based model of a Lithium-ion battery. In this model (see here), state transition equations (i.e., internal states) relate the voltage discharge from the battery (i.e., the output) given an applied current (i.e., the input).

```python
class Battery(PrognosticsModel):
    inputs = [
    ‘i’ # current applied to battery
    ]
    states = [
        # internal battery model states, e.g., temperature, surface potentials
        # nasa.github.io/progpy/api_ref/prog_models/IncludedModels.html
        ‘x_1’, # State 1
        ‘x_2’,  # State 2
              …
        ]
    outputs = [
        ‘t’, # Battery temperature
        ‘v’  # Voltage supplied by battery
    ]
    events = [
       'EOD' # battery end-of-discharge
    ]

    # Default parameters. Overwritten by passing parameters into constructor
    default_parameters = {
       'x0':{  # Initial State
        },
       'param1':p_1,
        ….
        # Include parameters to define battery model
        # nasa.github.io/progpy/api_ref/prog_models/IncludedModels.html
    }

    def dx(self, x, u):
        # calculate derivative of the battery state
        return self.StateContainer({})  # Return state container with derivative
    def output(self, x):
        # From the state, calculate temperature and voltage
        return self.OutputContainer({'t': x['t'], 'v': x['v']})

    def event_state(self, x):
        # From current state, calculate progress towards EOD
        return {
         'EOD': v_now - v_threshold
         # EOD occurs when voltage drops below threshold
        }
```

The resulting model can then be used in simulation:

```
m = Battery()
def future_load(t, x=None):  # system loading
    return m.InputContainer({‘i’:1})  # Constant 1 amp applied

simulated_results = m.simulate_to_threshold(future_load, dt=0.005)

print(f'EOD was reached in {round(simulated_results.times[-1],2)}seconds')
```

ProgPy also includes data-driven models such as the LSTM State Transition and Dynamic Mode Decomposition models. These are trained using data and then used for simulation or prognostics, as above.

Selected use case: prognostics of battery discharge cycle

Models can be used for prognostics with prog_algs. Prognostics is often split into two steps: state estimation and prediction. In state estimation, the system state is estimated, with uncertainty, using the prior state estimate and sensor data. In prediction, the state estimate is predicted forward.

This example illustrates predicting the battery discharge. Here data is retrieved from some unspecified source (data_source). This can be a data stream, playback file, or any other source. This is similar to the sim_battery_eol example (see here).

```
batt = Battery()
x0 = batt.initialize()
# Create Particle Filter State Estimator
state_estimator = state_estimators.ParticleFilter(batt, x0)
# Create Monte Carlo Predictor
predictor = predictors.MonteCarlo(batt)

# Future loading as function of time (t) and state (x)
# In this case- constant load
def future_loading(t, x=None):
    return batt.InputContainer({'i':2.35})

while RUNNING:
    u, z = data_source.get_data()
    # Estimate state using loading (u) and output measurements (z)
    state_estimator.estimate(t, u, z)
    eod = batt.event_state(filt.x.mean)['EOD']
    print(f"  - State of charge (mean): {eod}")
    # Only predict every PREDICTION_UPDATE_FREQ steps
    if (step%PREDICTION_UPDATE_FREQ==0):
```

```
mc_results = mc.predict(filt.x, future_loading, t0 = t, dt=TIME_STEP)
metrics = mc_results.time_of_event.metrics()
eod_mean = metrics['EOD']['mean']
eod_std = metrics['EOD']['std']))
print(f'  - Predicted end of discharge: {eod_mean} (sigma: {eod_std})')
```

NASA use cases

ProgPy has been used in various NASA projects. Two are described below.

Data and Reasoning Fabric

ProgPy functionality predicting battery degradation was implemented to assess the Li-ion batteries state of charge during unmanned aerial vehicle (UAV) flight. Based on planned trajectories, ProgPy provided UAV operators with statistics on expected battery health during flight and helped to ensure safety in the national airspace. (Jarvis et al., 2022)

Autonomous Spacecraft Operations

ProgPy was used to create models predicting the ISS life support system degradation informing maintenance. Researchers evaluated the performance of multiple potential models with data from the system and ProgPy metrics and visualization. Researchers updated models based on performance results. The selected model will be integrated with ProgPy state estimation and prediction into a prognostic application for crew or ground support.

Acknowledgements

ProgPy is supported by NASA's Autonomous Spacecraft Operations, Data and Reasoning Fabric, System-Wide Safety, and Transformative Tools and Technologies projects. Additionally, development is supported by Northrop Grumman Corporation, Vanderbilt University, the German Aerospace Center (DLR), Research Institutes of Sweden and others.

References

Bole, B., Kulkarni, C., & Daigle, M. (2014). Randomized battery usage data set. In *NASA Ames Research Center, Moffett Field, CA*. NASA Prognostics Data Repository. https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository

Dadfarnia, M., & Drozdov, S. (2013). Simantha. In *NIST*. National Institute of Standards; Technology [Software]. https://github.com/usnistgov/simantha

Daigle, M. (2016a). *Prognostics algorithm matlab library*. https://github.com/nasa/PrognosticsAlgorithmLibrary

Daigle, M. (2016b). *Prognostics metrics matlab library*. https://github.com/nasa/PrognosticsMetricsLibrary

Daigle, M. (2016c). *Prognostics models matlab library*. https://github.com/nasa/PrognosticsModelLibrary

Davidson-Pilon, C. (2022). *Lifelines, survival analysis in python* (Version v0.27.4). Zenodo. https://doi.org/10.5281/zenodo.7329096

Goebel, K., Daigle, M. J., Saxena, A., Roychoudhury, I., Sankararaman, S., & Celaya, J. (2017). *Prognostics: The science of making predictions*.

Jarvis, K., Corbetta, M., Teubert, C., & Schuet, S. (2022). Enabling in-time prognostics with surrogate modeling through physics-enhanced dynamic mode decomposition method. *Annual Conference of the PHM Society*, *14*. https://doi.org/10.36001/phmconf.2022.v14i1.3238

Saxena, A., & Goebel, K. (2008). Turbofan engine degradation simulation. In *NASA Ames Research Center, Moffett Field, CA*. NASA Prognostics Data Repository. https://www.nasa.gov/content/prognostics-center-of-excellence-data-set-repository

Schreiber, J. (2017). Pomegranate: Fast and flexible probabilistic modeling in python. *The Journal of Machine Learning Research*, *18*(1), 5992–5997.

Teubert, C., Daigle, M. J., Sankararaman, S., Goebel, K., & Watkins, J. (2017). A generic software architecture for prognostics (GSAP). *International Journal of Prognostics and Health Management*, *8*(2). https://doi.org/10.36001/ijphm.2017.v8i2.2618

Teubert, C., Jarvis, K., Corbetta, M., Kulkarni, C., & Daigle, M. (2022). *ProgPy packages* (Version 1.4). https://nasa.github.io/progpy

Watkins, J., Teubert, C., & Ossenfort, J. (2019). Prognostics as-a-service: A scalable cloud architecture for prognostics: A scalable cloud architecture for prognostics. *Annual Conference of the PHM Society*, *11*.