Assessment of *ProgPy* - An Open-Source Condition Monitoring and Diagnostics Tool

Magnus S. Nordeide[1], Chetan S. Kulkarni[2], Anne-Lena Kampen[1], Maneesh Singh[1]

[1.] Western Norway University of Applied Sciences, Inndalsveien 28, 5063 Bergen, Norway
magnus.nordeide@gmail.com
maneesh.singh@hvl.no
anne-lena.kampen@hvl.no
[2.] KBR Inc, NASA Ames Research Center, Mountain View, California, U.S.A.
chetan.s.kulkarni@nasa.gov

Abstract. Traditional maintenance programs, such as corrective and preventive strategies, may lead to high costs and operational inefficiencies. Condition Monitoring and Diagnostics (CM&D) aims to improve these maintenance strategies by enabling timely insights into equipment health and performance. However, implementation of CM&D can be challenging without a robust framework that manages data efficiently, supports interoperability and simplifies integration. To address these challenges ProgPy, an open-source Python-based prognostics tool developed by NASA Ames Research Center, offers a structured solution for broader Prognostics and Health Management (PHM) applications.

Ongoing research is assessing the feasibility of implementing ProgPy as a Condition Monitoring and Diagnostics (CM&D) solution by comparing its framework to the guidelines for open CM&D systems recommended in the ISO 13374-2 standard. This evaluation aims to highlight ProgPy's strengths and identify opportunities for improvement, thereby, contributing to its advancement as an effective tool for Prognostics and Health Management (PHM).

This paper presents the results of an initial assessment of the ProgPy toolbox through a gearbox case study using open-source datasets.

Keywords: Condition-Monitoring, Diagnostics, Failure, Gearbox, ISO 13374, Maintenance, Prognostics, ProgPy.

1 Introduction

Traditional maintenance strategies for industrial equipment, for example, wind turbine gearboxes, have predominantly been corrective and preventive in nature. Corrective maintenance, the most common approach, is a reactive approach where repairs or replacements are made only after a failure occurs. While this method is commonly used, this approach often results in unplanned downtimes. In large systems such as wind turbines, these downtimes can be particularly prolonged and

costly due to the complex designs and challenging accessibility. The unpredictability of failures under corrective maintenance not only disrupts energy production but also results in significant maintenance expenses

To reduce chances of unplanned failures, preventive maintenance involving scheduled inspections and routine servicing based on predetermined time-intervals, usage metrics or risk assessments are commonly used. This proactive approach aims to reduce the likelihood of equipment failure by addressing potential issues before significant degradation takes place. However, preventive maintenance does not consider the actual condition of equipment, hence components may be replaced or serviced unnecessarily, resulting in increased maintenance costs without proportional improvements in reliability [3].

Preventive and corrective maintenance strategies each have distinct strengths and weaknesses. Traditional maintenance approaches are widely adopted across industries due to their familiarity, ease of implementation, predictability and minimal reliance on high-quality data. However, despite these advantages, they can sometimes be unreliable, inefficient, and costly [4].

To address the limitations of traditional maintenance approaches and improve equipment reliability, condition monitoring solutions have been proposed. These solutions integrate various techniques for data acquisition, processing, diagnostics, prognostics, and predictive maintenance planning. By continuously monitoring the equipment and analyzing the collected data, this approach may be able to predict the remaining useful life (RUL) of components and determine optimal times for inspection and maintenance.

Several standards guide structured procedures and frameworks for condition monitoring, supporting early fault detection and recommending predictive maintenance. Notable among them, ISO 17359 provides general guidelines for machine condition monitoring, while ISO 13374 [5] focuses on data processing and management. ISO 13379-1:2012 [15] and ISO 13381-1:2015 [16] offer guidance on data interpretation and prognostics, helping estimate time-to-failure. Together, these standards enable consistent, reliable monitoring practices essential for proactive maintenance.

2 Architecture of a Condition Monitoring and Diagnostics (CM&D) Software System

2.1 ISO 13374 (Condition Monitoring and Diagnostics of Machines)

ISO 13374 Parts 1-4 are a set of international standards that provide guidelines for creating open Condition Monitoring and Diagnostics (CM&D) software systems. **Fig. 1** shows recommendation of ISO 13374-2 (Data Processing, Communication, and Presentation) for organizing the architecture of the system into eight functional data processing blocks. Each block performs specific tasks, transforming and archiving data as it moves to the next block in the sequence.

2.2 Data Acquisition (DA)

The Data Acquisition block is responsible for collecting raw, real-time data on structural and operational conditions using transducers or sensors (such as accelerometers, thermocouples, and acoustic sensors) mounted on equipment (example, gearbox). The objective is to ensure data is collected in a structured, consistent and reliable manner for further processing in subsequent stages. This block receives inputs in the form of analog, digital, and/or manual signals, while its outputs consist of digitized data, often including both UTC and local time-stamped records.

2.3 Data Manipulation (DM)

The Data Manipulation block receives input from the Data Acquisition block and cleanses it by reducing imperfections such as drifts, noise, outliers, and missing values. It then processes the cleansed data using signal processing techniques, including filtering, normalization, Fourier transforms, wavelet analysis, and feature extraction, to transform raw data into clean, rational information ready for further analysis by subsequent blocks.

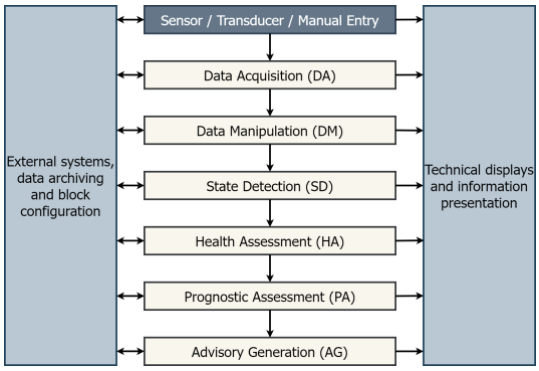

Fig. 1. Flowchart depicting the architecture of an open Condition Monitoring and Diagnostics (CM&D) software system as recommended by ISO 13374-2.

2.4 State Detection (SD)

The primary function of the State Detection block is to analyze the cleaned data from the Data Manipulation block to identify deviations or anomalies from expected normal behavior in structural and operational conditions. The goal is to classify the system's state and detect the early signs of potential issues based on the threshold being set by the users/operators. Outputs from this block include anomaly alerts, rate-of-change notifications, threshold boundary alarms, and enumerated state indicators [5].

2.5 Health Assessment (HA)

When deviations or anomalies have been detected by the previous block, the Health Assessment block performs a thorough evaluation to assess the overall condition of the system. Using techniques such as signature analysis, it diagnoses

and quantifies faults to provide a comprehensive health status. In gearbox condition monitoring, this includes detailed analysis to identify and measure degradation mechanisms like bearing wear, gear tooth damage, or gear pitting. The output from this block includes a report detailing the current condition of the equipment. [5].

2.6 Prognostic Assessment (PA)

The Prognostic Assessment block utilizes the equipment status report from the previous block to predict the progression of faults and estimate the remaining useful life (RUL) of degraded components. This is achieved using predictive models - physical, data-driven, or hybrid - that factor in anticipated future load conditions. Expected outputs from this block include RUL estimates, projections of future health conditions, and the likelihood of potential failures.

2.7 Advisory Generation

The Advisory Generation block is the final stage in the Condition Monitoring & Diagnostics (CM&D) process. It provides actionable recommendations based on the analyses conducted in all preceding block. This block thus aids in planning inspection and maintenance activities.

2.8 External Systems and Block Configuration

External Systems and Block Configuration support all previously described blocks by providing necessary configuration and interaction with external inputs and outputs. It specifies the importance of being able to retrieve historical data from the maintenance system and provides configuration information. Block configuration involves the setup and tailored configuration to each block to operate them effectively. External systems provide access to maintenance records and operational data, such as start/stop events and load histories. The purpose is to facilitate communication of recommendations such as operational adjustments and changes in procedures.

2.9 Technical Displays and Information Presentation.

Technical displays and Information Presentation also support all previously described blocks by facilitating the visualization and interpretation of their outputs. ISO 13374-2 specifies that technical displays should provide
end users/operators with essential information to support corrective action decisions. This involves converting complex data into clear, actionable insights that enable operators to make corrective action decisions confidently. When anomalies occur, users may also need the capability to delve into the State

Detection, Data Manipulation, and Data Acquisition displays for deeper analysis. [5].

3 Overview of ProgPy

ProgPy, or the Prognostics Python Package, is an open-source prognostics python toolbox developed by NASA for Prognostics and Health Management (PHM) of engineering systems and subsystems. This toolbox aims to support researchers and developers in creating prognostic models and tools for predicting the future performance and health of various complex systems, including aerospace, energy and industrial applications [6].

ProgPy's architecture is built on three core components: Prognostic Models, Prognostic Engine and Prognostic Support Tools [7]. These components enable researchers and developers to build predictive models, estimate system health states, and visualize outcomes across applications.

- The **Prognostic Models** can range from physics-based approaches to data-driven or mixed hybrid techniques.
- With the **Prognostics Engine**, users can apply these models to estimate health states and make prognostic predictions, creating a flexible, extendable framework.
- The **Prognostics Support Tools** assist with model tuning, benchmarking and visualization, making it easier to develop and refine predictive solutions.

In **Fig. 2** Prognostic Models is represented by "Prognostic Models", the Prognostics Engine is represented by "State Estimator", "Predictor" and "Analysis", while the Prognostic Support Tools is not explicitly outlined in the figure. The "Analysis" component incorporates output from the Predictor as an event or future estimates and provides functionalities for evaluating, visualizing and decision making based on the prognostic results.

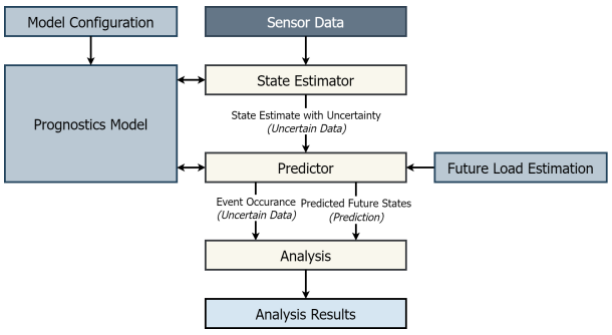

Fig. 2. Flowchart depicting the architecture of ProgPy.

ProgPy has been implemented in several NASA projects and has proven to be a valuable tool for system health management applications. For example, use of ProgPy for the prognostics and health management of UAV batteries offers operators with essential insights into battery status and provides support for safe operations [6,8].

3.1 Prognostic Models

ProgPy takes all the input information and passes it through a derived prognostics model to estimate the current (through StateEstimation) and future (through Predictor) state of the system. For these tasks ProgPy can use both – physics-based or data-driven – for predicting. Physics-Based Models rely on system-specific underlying physics models to simulate system behavior; where as Data-Driven Models apply machine learning modeling techniques to historical data to generate state estimations and predict future behavior. It can also use hybrid models that combine elements of both approaches to reduce uncertainty, improve interpretation, enhance accuracy and decrease dependency on historical data. ProgPy already has a library of prognostic models, including, some physics models for batteries, pumps, BLDC motors etc, and data-driven models based on LSTM [9].

In the case of high-fidelity models that are computationally expensive, substitute models can be -developed which serve the purpose of approximating the behavior of the model to make inference time shorter. “Surrogate models” are models which serve the purpose of approximating the behavior of another model to shorten inference time or for testing purposes [10] which are computationally less expensive and efficient.

In addition, ProgPy also supports “Composite” and “Ensemble” models [10]. A Composite Model combines multiple models to represent interconnected systems, such as linking a gearbox model with a motor model. Ensemble models is a mixture of models to leverage different strengths, useful for cases where different models predict aspects of system behavior. Ensemble models mix different models to benefit their unique strengths, for example when various models predict different aspects of system behavior.

Core elements for the configuration of these prognostic models include “Inputs”, “Outputs”, “States” and “Events” [10]. These are discussed briefly as below:

- Inputs represent how the system is loaded and external factors influencing the system, such as the voltage and external load applied to a motor or the temperature surrounding the motor.
- Outputs include measureable characteristics of the system, such as the torque or speed of a motor, vibration, current drawn etc.
- States define the internal conditions of the system, encapsulated by the StateContainer in ProgPy, which assess the system’s health and operational capacity.

- Events mark specific points of interest, like when a set failure threshold is passed. In the context of prognostics, these are often RUL (remaining useful life) and ToE (time of event). Systems may have multiple failure modes, each tracked as a separate event.

ProgPy models operate primarily as “state transition models”, predicting the next state based on the current state, external and internal factors (inputs and outputs). When the state of the system passes an explicitly defined threshold, an event is triggered, and relevant data is stored and used for analysis. ProgPy can also operate through a “direct-prediction” approach that estimates RUL directly from current state of system without any state tracking. This open framework provides a comprehensive and intuitive pipeline for extensive PHM applications [10].

3.2 Sensor Data

Sensor data - includes temperature vibrations, torque, current and voltage, pressure and more. Multiple sensor input may improve the state estimation of a system [11].

3.3 State Estimator

State estimation is necessary if the state of the system isn’t directly -observable [10]. It is a description of a system’s state should be available to assess health state and to propagate for further prognostics estimates.

State Estimator step estimates the internal state of a system using calibration methods (Kalman filter [12, 10], Unscented Kalman filter [12, 10], Particle filter [10], etc.). ProgPy introduces flexibility to its framework by providing helper methods along with already implemented filters, including the option to add custom state estimators. ProgPy’s state estimate comes with the respective uncertainty quantification, represented by the ProgPy UncertainData structure module.

3.4 Predictor

After estimating the current state of the system, ProgPy’s Predictor can be used to predict its future states - for either detecting a fault or estimating RUL. ProgPy generalizes faults as events, which may also include other points of interest within the system. [10].

For a prediction to take place, an initial state estimation, a prognostics model and an estimate of future load are required. Prognostics can be done without uncertainty quantification and is then referred to as “Simulation” by ProgPy, or it can be done with an uncertainty quantification using methods such as Monte Carlo Simulation or Unscented Transform [10].

3.5 Analysis

After results from preceding blocks (state estimation and prediction) are generated, Analysis in ProgPy is designed to process and interpret the outputs.

ProgPy provides various data structures to manage simulation and prediction data. The results of State Estimation are stored in an object of type UncertainData, which includes several methods for analyzing the distribution, such as calculating the mean, median and displaying a summary of key distribution metrics in a readable format [10]. The methods and output of the analysis is also stored under object type Prediction. Predictions can be made of multiple types of data, including future internal states, inputs, outputs and state of health (ProgPy event state). These analyses give information on statistics of the distributions and metrics which can be used for decision making

3.6 Additional features of ProgPy

ProgPy provides a Service-Oriented Architecture test implementation, designed to perform prognostics on engineering systems through a REST API [10]. The Prognostics-as-a-Service approach includes a server providing access to prognostic functions, and a client for interfacing with the server. The tool can be implemented for online prognostics estimates if future estimated loading profiles are provided. Given the increasing shift towards cloud-based analysis and storage in prognostics [9], ProgPy is in step with this trend by implementing Prognostics-as-a-Service functionality.

4 Motivation and Aim of the Research

Although substantial research has been carried out to develop condition monitoring systems for gearboxes, new tools with new applications and models are evolving..

4.1 Aim of the Research

The aim of this research is to assess the feasibility and effectiveness of using *ProgPy* for prognostics and health assessment of a gearbox and as part of future work integrate developed models into the toolbox. To carry out this work, a comparative analysis between the architecture of ProgPy and the architecture recommended by the ISO 13374-2 standard has been conducted.

ProgPy's performance will be further evaluated using a publicly available dataset from a faulty gearbox. It is expected that the study will offer valuable insights into the similarities and differences between the two architectures. Any identified gap or limitation in ProgPy's alignment with ISO 13374-2 can provide insights into areas where suggestions could be provided to enhance ProgPy to better align with industry standards.

4.2 Scientific Novelty and Importance of Research

This work presents an evaluation of ProgPy's suitability for gearbox prognostics and health assessment. The scientific novelty of this paper is the comparative analysis of ProgPy's architecture versus the architecture recommended by ISO 13374-2 for open Condition Monitoring and Diagnostics (CM&D) software systems.

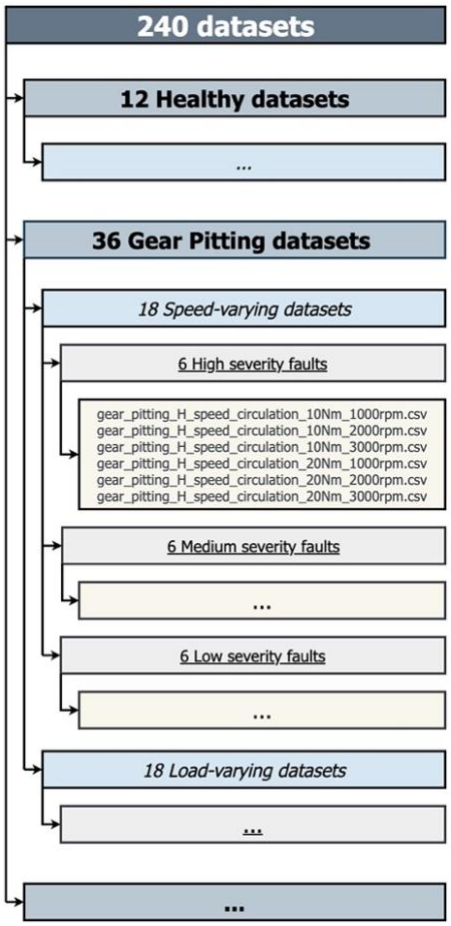

Fig. 3. Flowchart representation of structure of MCC5-THU gearbox fault diagnosis dataset.

5 Method

The research is conducted in two parts. The first part compares ProgPy's architecture with the architecture recommended by the ISO 13374-2 standard.

In the second part, ProgPy is evaluated using the publicly available MCC5-THU gearbox fault diagnosis dataset. The dataset has been generated for a two-stage parallel gearbox operating under variable speeds and loads by manually introducing a spectrum of fault types and severity levels, ranging from single faults (such as gear wear, pitting, wear, cracks, and missing teeth) to compound faults involving both gears and bearings through precise machining [13].

The dataset comprises of 240 time series recordings of vibration, speed, and torque signals, sampled at 12.8 kHz over 60-second intervals. By capturing the vibrations of the 36-tooth gear on the intermediate shaft and its support bearings, this dataset offers valuable insights into how faults manifest in vibration signals across different operational conditions [13]. **Fig. 3** shows the organization of the dataset.

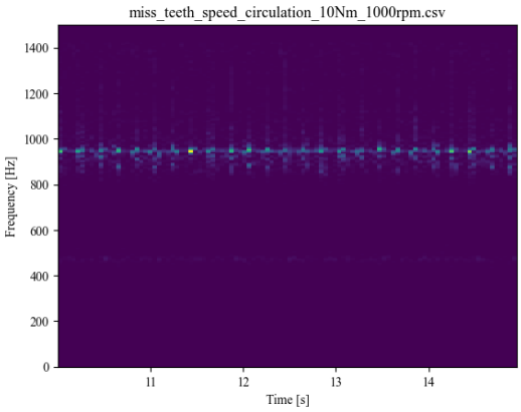

Fig. 4. Spectrogram generated from vibration data of a gear with a missing tooth.

In this example, the dataset has been transformed into visual representations as spectrograms by converting the time-domain signal into a frequency-domain signal using the Short-Time Fourier Transform (STFT). **Fig. 4** presents a spectrogram generated from a five-second segment of an experiment where a fault has been introduced by removing a tooth from a gear on the intermediate shaft. The spectrogram displays changes in frequency intensity at regular intervals, offering a comprehensive view of all frequencies and enabling visualization of fault signatures.

6 Study of ProgPy's Architecture

In this part each functional block in ProgPy is compared to the corresponding block in the ISO 13374-2 standard architecture **(Fig. 5)**. The comparison examines the operation of individual blocks to identify ProgPy's strengths and limitations relative to the guidelines from the ISO standard.

6.1 Data Acquisition

ProgPy has capability to directly collect data streams from sensors, but data acquisition is not managed directly; instead, it relies on external systems or tools for any prior data cleanup.

In the gearbox prognostics example, the data was obtained in digital format and was directly utilized for further analysis.

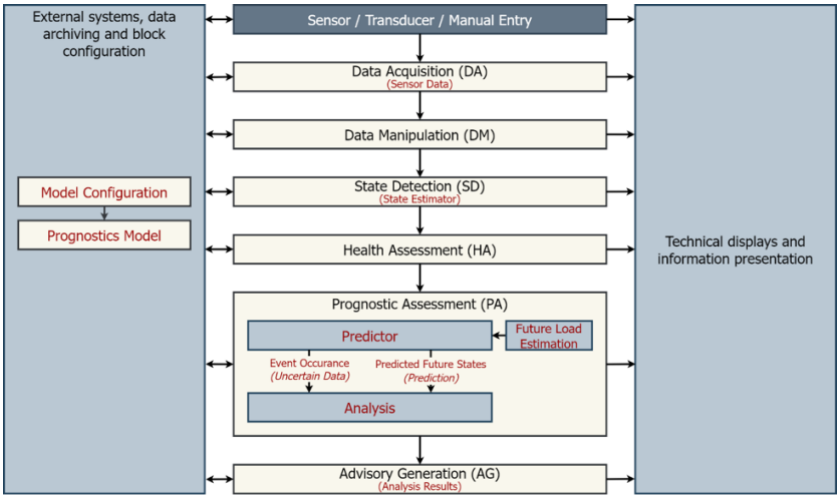

Fig. 5. Flowchart illustrating the architecture of ProgPy compared to the architecture recommended by the ISO 13374-2 standard Project flowchart.

6.2 Data Manipulation

As discussed earlier, ProgPy does not include built-in data manipulation functionalities such as feature extraction, normalization, or filtering. These preprocessing steps are expected to be performed externally using tools like NumPy, SciPy, or Pandas before the data is input into ProgPy.

Initial approaches for the data manipulation of the gearbox prognostics example may include envelope analysis and filtering for isolating frequencies of interest, time-frequency analysis [14].

6.3 State Detection

ProgPy features the State Estimation block, which assesses the current health or operational state of a system based on input data. ProgPy leverages selected prognostic models to continuously evaluate the system's condition.

A key difference between the State Detection function in ISO 13374-2 and ProgPy's State Estimation is that the ISO approach typically recommends identifying discrete states, such as "normal," "warning," or "failure," using rules, thresholds, or basic pattern recognition techniques. In contrast, ProgPy's State Estimation provides a more detailed, continuous assessment − often estimating a numerical degradation level by applying physics-based or machine learning models.

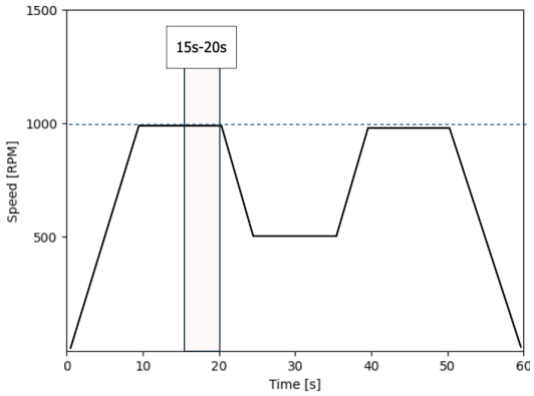

Fig. 6. Speed profile of gearbox run and time-interval of interest

State Detection detects anomalies by identifying deviations from baseline operation. While ISO 13374-2 recommends a standalone feature for State Detection, ProgPy does not have this independent functionality; rather it integrates this block with the subsequent block of Health Assessment.

In the gearbox prognostics example, faults can be detected by comparing spectrograms of healthy and faulty gears using techniques like Convolutional Neural Networks (CNNs).

6.4 Health Assessment

The Health Assessment block in ISO 13374-2 involves assessing the severity of any detected faults. ProgPy includes State Estimation functional block, which is equivalent to the Health Assessment block. It provides degradation metrics that offer insights into fault severity by quantifying the system's health on a continuous scale.

In the gearbox prognostics example, each fault type produces a distinct spectrogram pattern, which can be used for identification. **Fig. 6** shows the speed profile of the gearbox run and the time-interval of interest for generation of spectrograms. **Fig. 7** displays spectrograms generated from five-second segments of experiments with various introduced fault types in the frequency range of 400Hz to 1250Hz. These spectrograms can be further analysed using techniques like Convolutional Neural Networks (CNNs). Additionally, it may be possible to quantify fault severity by calibrating frequency intensities against fault degrees.

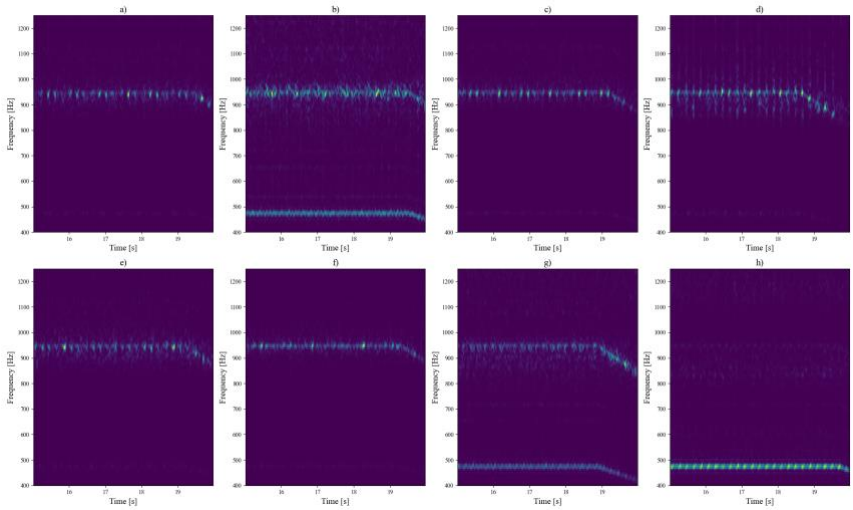

Fig. 7. Spectrograms from interval 15s-20s during stable RPM at 1000rpm and constant torque at 10Nm. The frequency dips in the last second of Figure 7 spectrograms are caused by speed decrease, showing a discrepancy between the set speed profile, as shown in Figure 6, and the actual speed profile.

a) Healthy b) Gear pitting c) Gear Wear d) Missing Teeth e) Teeth Break f) Teeth Crack g) Teeth Break and Bearing Inner Ring h) Teeth Break and Bearing Outer Ring

6.5 Prognostic Assessment

ProgPy includes a specialized Prognostic Assessment block (Predictor), making it well-suited for estimating remaining useful life (RUL). This block allows for modeling system degradation patterns and predicting RUL under varying load conditions, along with associated uncertainties. For effective performance, this block requires reliable prognostic models, accurate inputs from preceding blocks and estimates of future loading. When the system's current condition and expected usage is well understood, ProgPy can deliver precise RUL estimates, along with quantified uncertainties.

In the context of ISO 13374-2, the Predictor aligns with the standard’s Prognostic Assessment block, where future states and time-to-event estimations are calculated to enable timely maintenance actions. This predictive capability supports proactive decision-making by estimating events such as RUL, aligning with the structure of ISO 13374-2.

In the gearbox prognostics example, this step has not yet been investigated and part of the future work.

6.6 Advisory Generation

In ProgPy, the "Analysis" function closely aligns with the Advisory Generation block in ISO 13374-2. ProgPy's analysis can include visualizing predictions and generating actions based on model outputs, offering valuable recommendations to support informed decision-making.

In the gearbox prognostics example, this step has not yet been investigated and will be touched upon in future work.

6.7 External Systems and Block Configuration

In ProgPy, the external systems and block configuration house the details of prognostic models, which are utilized for making predictions and calculating the remaining useful life (RUL). In addition, ProgPy allows for saving results from simulations, enabling storing of outputs such as estimated current states, estimated future states, and event predictions such as RUL predictions.

6.8 Technical Displays and Information Presentation

ProgPy includes features for visualizing calculations using various plot types. For more complex or specialized visualizations, users can generate external plots using the estimated outputs.

ProgPy's analysis tools provides detailed outputs of the distributions connected to state estimation and prediction, however, to fully conform to the standard's technical display requirements, additional features for interactive drill-down, visualization, and comprehensive anomaly analysis may be necessary.

7 Conclusions

This paper explores the suitability of ProgPy as an open Condition Monitoring and Diagnostics (CM&D) software tool. A comparison of its architecture against that recommended by the ISO 13374-2 standard reveals that ProgPy's design aligns closely with the recommended structure. ProgPy's flexible architecture enables users to integrate a wide variety of domains into a cohesive framework. Its open structure allows models, state estimators, and predictors to be easily modified, eliminating the need for users to adapt their cases to fit the software.

The comparative study has also identified some areas for improvement. ProgPy may be improved by including a library of tools for data manipulation, health assessment and additional features for technical displays.

The gearbox examples discussed in this work will be further enhanced to develop a usable model and be integrated with the ProgPy toolbox for estimating the

degradation as well as estimating its RUL based on operational profiles. The aim is to make these models available as open source through the ProgPy toolbox.

8 References

1. J. F. Manwell, J. G. McGowan and A. L. Rogers, Wind Energy Explained: Theory, Design and Application, Wiley, 2010, p. 704.
2. J. Ribrant and L. M. Bertling (2007) Survey of Failures in Wind Power Systems With Focus on Swedish Wind Power Plants During 1997–2005. IEEE Transactions on Energy Conversion 22(1):167-173.
3. C. Huang, S. Bu, H. H. Lee, C. H. Chan, S. W. Kong and W. K. Yung (2024) Prognostics and health management for predictive maintenance: A review. Journal of Manufacturing Systems 75. pp. 78-101.
4. C. Wagner and B. Hellingrath (2019) Implementing Predictive Maintenance in a Company: Industry Insights with Expert Interviews. 2019 IEEE International Conference on Prognostics and Health Management (ICPHM). pp. 1-8.
5. International Organization for Standardization (2004) ISO 13374-2:2004 - Condition monitoring and diagnostics of machines — Data processing, communication, and presentation — Part 2: Data processing.
6. C. Teubert, K. Jarvis, M. Corbetta, C. Kulkarni and M. Daigle (2023) ProgPy: Python Packages for Prognostics and Health Management of Engineering Systems. Journal of Open Source Software 8(87):5099.
7. B. Keeter (2024) 2024 Software of the Year Award Co-Winner -Prognostics Python Packages (ProgPy). https://www.nasa.gov/organizations/otps/2024-software-of-the-year-award-co-winner-prognostics-python-packages-progpy/, last accessed 2024/11/08.
8. K. Jarvis, M. Corbetta and C. Teubert (2022) Enabling in-time prognostics with surrogate modeling through physics-enhanced Dynamic Mode Decomposition method. Annual Conference of the PHM Society 14(1).
9. J. Guo, Z. Li and M. Li (2020) A Review on Prognostics Methods for Engineering Systems. IEEE Transactions on Reliability 69(3):1110-1129.
10. C. Teubert, K. Jarvis Griffith, M. Corbetta, C. Kulkarni, P. Banerjee, J. Watkins and M. Daigle (2023) ProgPy Python Prognostics Packages. https://github.com/nasa/progpy, last accessed 2024/11/11.
11. R. Jung and S. Weiss (2021) Modular Multi-Sensor Fusion: A Collaborative State Estimation Perspective. IEEE Robotics and Automation Letters 6(4):6891-6898.
12. G. Chindriş, A. I. Ilieş and D. Pitică (2020) A Comparison between State of Charge Estimation Methods: Extended Kalman Filter and Unscented Kalman Filter. 2020 IEEE 26th International Symposium for Design and Technology in Electronic Packaging (SIITME). pp. 376-381
13. S. Chen, Z. Liu, X. He, D. Zou and D. Zhou (2024) Multi-mode fault diagnosis datasets of gearbox under variable working conditions. Data in Brief, no. 54.
14. M. Tiboni, R. Carlo, R. Bussola and C. Amici (2022) A Review on Vibration-

Based Condition Monitoring of Rotating Machinery. Applied Sciences 12(3):972.
15. International Organization for Standardization (2012) ISO 13379-1:2012 - Condition monitoring and diagnostics of machines — Data interpretation and diagnostics techniques Part 1: General guidelines
16. International Organization for Standardization (2015) ISO 13381-1:2015 - Condition monitoring and diagnostics of machines — Prognosttics Part 1: General guidelines
17. International Organization for Standardization (2018) ISO 17359 : 2018 - Condition monitoring and diagnostics of machines