Novel Metrics to Evaluate Probabilistic Remaining Useful Life Prognostics with Applications to Turbofan Engines

Ingeborg de Pater, Mihaela Mitici

Faculty of Aerospace Engineering, Delft University of Technology, Kluyverweg 1, 2628HS Delft, The Netherlands

i.i.depater@tudelft.nl
m.a.mitici@tudelft.nl

Abstract

Well-established metrics such as the Root Mean Square Error or the Mean Absolute Error are not suitable to evaluate estimated distributions of the Remaining Useful Life (i.e., probabilistic prognostics). We therefore propose novel metrics to evaluate the quality of probabilistic Remaining Useful Life prognostics. We estimate the distribution of the Remaining Useful Life of turbofan engines using a Convolutional Neural Network with Monte Carlo dropout. The accuracy and sharpness of the obtained probabilistic prognostics are evaluated using the Continuous Ranked Probability Score (CRPS) and weighted CRPS. The reliability of the obtained probabilistic prognostics is evaluated using the α-Coverage and the Reliability Score. The results show that the estimated distributions of the Remaining Useful Life of turbofan engines are accurate, reliable and sharp when using a Convolutional Neural Network with Monte Carlo dropout. In general, the proposed metrics are suitable to evaluate the accuracy, sharpness and reliability of probabilistic Remaining Useful Life prognostics.

1. Introduction

Maintenance is undergoing a paradigm shift from time-based maintenance, where tasks are scheduled at fixed time intervals, to predictive maintenance. Under predictive maintenance, sensors continuously measure the condition of components. These measurements are used to predict the Remaining Useful Life (RUL) of components. In turn, RUL prognostics are integrated into maintenance planning. Predictive maintenance has the potential to reduce the maintenance costs, while maintaining the reliability of assets (Lee & Mitici, 2020).

Most studies focus on developing *point* RUL prognostics, i.e., one value for the RUL prediction. For example, a prognostic may indicates that the RUL equals 30 flight cycles for an aircraft component (see Figure 1-A). Point RUL prognostics for turbofan engines are developed in (de Pater, Reijns, & Mitici, 2022; Li, Ding, & Sun, 2018) using a Convolutional Neural Network (CNN) and in (Xia, Feng, Lu, Fei, & Xue, 2021) using a Long Short-Term Memory neural network. In (Mitici & de Pater, 2021), point RUL prognostics for aircraft Cooling Units are developed using particle filtering. In (Lee & Mitici, 2022), point RUL prognostics are obtained for aircraft landing gear brakes using linear regression.

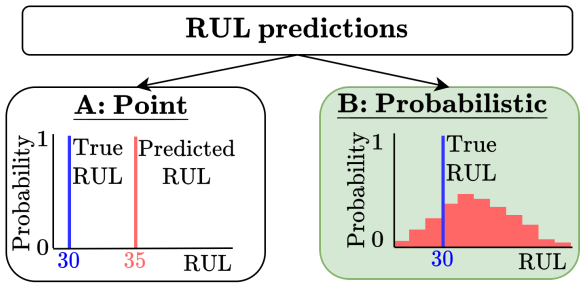


Figure 1. A) Point RUL prognostics, B) Probabilistic RUL prognostics.

For reliability purposes, however, it is key that the uncertainty associated with the estimated RUL is also determined. In this line, several studies estimate the distribution of RUL, i.e., probabilistic RUL prognostics (see Figure 1-B). In (Nguyen & Medjaher, 2019) and (Biggio, Wieland, Chao, Kastanis, & Fink, 2021) the RUL distribution of turbofan engines is obtained using a Long Short-Term Memory neural network and Deep Gaussian processes, respectively. In (de Pater & Mitici, 2021) the RUL distribution of aircraft Cooling Units is estimated using particle filtering. Probabilistic RUL prognostics for nuclear components are developed in (Baraldi, Mangili, & Zio, 2015) using Gaussian Process regression. Last, in (Le Son, Fouladirad, & Barros, 2016) the RUL distribution is estimated using a noisy Gamma deterioration process.

To evaluate probabilistic RUL prognostics, well-established metrics such as the Root Mean Square Error (RMSE) or the Absolute Mean Error (MAE) are not directly applicable. In

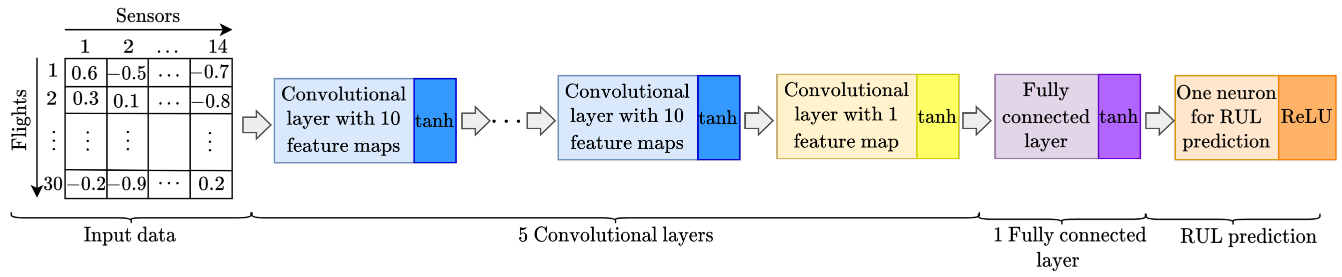


Figure 2. Schematic overview of the CNN architecture for dataset FD001.

principle, RMSE and MAE could be computed relative to the mean of the estimated RUL distribution. However, this would disregard the variance and sharpness of the estimates, and give little indication of the actual trustworthiness of the RUL prognostics. In (Saxena et al., 2008; Saxena, Celaya, Saha, Saha, & Goebel, 2009), a few metrics are proposed to evaluate probabilistic RUL prognostics such as prognostic horizon, probabilistic $\alpha - \lambda$, (cumulative) relative accuracy and convergence. These metrics evaluate the accuracy of the RUL prognostics, and specifically on how this accuracy changes over the lifetime of components. For an example of their usage, see (Lall, Lowe, & Goebel, 2011). However, these metrics all require a sequence of RUL prognostics over the lifetime of each component. Yet, for many publicly available degradation test sets, such as the C-MAPSS data set on turbofan engines (Saxena & Goebel, 2008), only one RUL prognostic per test instance can be determined. As such, the prognostic horizon, probabilistic $\alpha - \lambda$, relative accuracy and convergence cannot be used to evaluate these single probabilistic RUL prognostics. Most importantly, these metrics do not explicitly quantify the reliability of the probabilistic RUL prognostics.

In this paper we propose novel metrics to evaluate the accuracy and sharpness of probabilistic RUL prognostics (CRPS and weighted CRPS), and metrics to explicitly evaluate the reliability (α-Coverage and Reliability Score). Compared with existing metrics, the weighted CRPS uses penalties when the RUL is overestimated/underestimated. Depending on the type of component, these penalties can be adjusted. For example, for safety critical components it is important that the RUL is not overestimated. Otherwise, an overestimated RUL could lead to a missed failure. In such cases, the weighted CRPS applies a larger penalty for a RUL overestimation than for a RUL underestimation. The Reliability Diagram and Reliability Score provide a means to graphically visualize the performance of the RUL prognostics. Unlike existing numerical metrics, this metric provides a visual interpretation of the performance of the prognostics as well. We illustrate our metrics for probabilistic RUL prognostics for the turbofan engines of the C-MAPSS data set. Here, we estimate a distribution of the RUL of the turbofan engines using a Convolutional Neural Network with Monte Carlo dropout.

In Section 2, we introduce the Convolutional Neural Network with Monte Carlo dropout to estimate a RUL probability distribution for turbofan engines. We next propose metrics to evaluate these estimated RUL distributions in Section 3. We illustrate the proposed metrics in a case study in Section 4.

2. Probabilistic RUL prognostics for turbofan engines using a Convolutional Neural Network with Monte Carlo dropout

In this section, we generate *probabilistic* RUL prognostics for aircraft turbofan engines using a Convolutional Neural Network (CNN) and Monte Carlo dropout. Specifically, we estimate the probability density function (pdf) of the RUL of an engine, and not just one point value for the RUL. We apply our methodology to the turbofan engine degradation simulation C-MAPSS dataset, which is generated using the NASA Commercial Modular Aero-Propulsion System Simulation (C-MAPSS) program (Saxena & Goebel, 2008). The dataset contains measurements of 21 sensors that monitor the degradation of the turbofan engines. The C-MAPSS dataset consists of four data subsets, each with a different number of operational and fault conditions (see Table 1). Each subset contains a training set, with run-to-failure instances, and a test set. For each failure instance in the test set, the data is terminated somewhere before failure with the aim to predict the RUL. More information on this publicly available data set can be found in (Ramasso & Saxena, 2014).

Table 1. C-MAPSS datasets for turbofan engines.

	FD001	FD002	FD003	FD004
Training instances	100	260	100	249
Testing instances	100	259	100	248
Operating conditions	1	6	1	6
Fault conditions	1	1	2	2

We select 14 out of the 21 sensors available from C-MAPSS that have non-constant measurements. The remaining 7 sensors exhibit constant measurements and are thus not considered for RUL prediction. The selected sensor measurements are normalized using min-max normalization (Li et al., 2018) with respect to the operating condition (Babu, Zhao, & Li, 2016). We also include the history of the operating conditions in the input of the CNN, i.e., the number of flights spent in each operating condition, as in (Babu et al., 2016).

The architecture and hyperparameters of the CNN are similar to the CNN proposed in (Li et al., 2018). Specifically, the CNN consists of 5 convolutional layers, where the first four convolutional layers each have 10 kernels of size 10×1 (i.e., one-dimensional kernels). The last convolutional layer has one kernel of size 3×1, combining all 10 feature maps into one feature map. This last feature map is flattened in a flatten layer, and connected to a fully connected layer. All these layers use the tangent (tanh) activation function. Last, one single neuron is attached to the fully connected layer to predict the RUL using the Rectified Linear Unit (ReLU) activation function. A schematic overview of this CNN is in Figure 2. The weights of the CNN are optimized using the Adam optimizer (Kingma & Ba, 2014) with a batch size of 512 samples, and a maximum of 250 training epochs. The learning rate is 0.001 for the first 200 epochs, and 0.0001 for the last 50 epochs. A cut-off value R_{early} of 125 flights is applied. We use a window size of 30 flights for FD001 and FD003, of 20 flights for FD002 and of 15 flights for FD004.

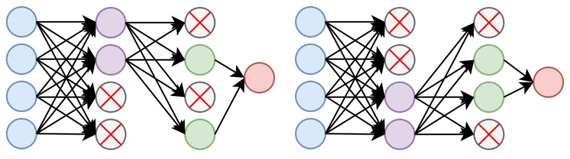
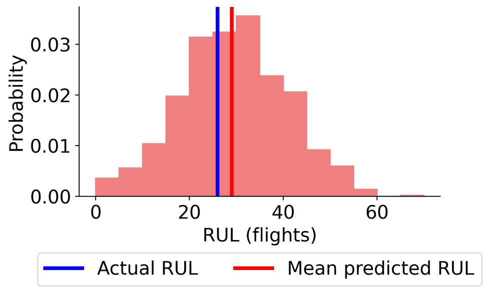

(a) First pass (b) Second pass

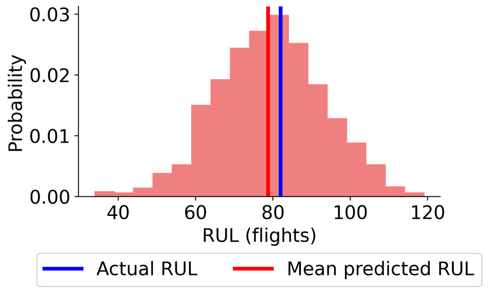
Figure 3. Monte Carlo dropout during two different passes through the network, in a neural network with two fully connected layers.

To obtain a probability distribution of the RUL using CNN, we additionally apply Monte Carlo dropout (Biggio et al., 2021; Gal & Ghahramani, 2016). During the training phase, we apply a dropout rate of $\rho = 0.5$ in each layer, with the exception of the last convolutional layer before the flatten layer, and the first convolutional layer (Gal, Hron, & Kendall, 2017). During the testing phase, we also use dropout and predict the RUL of each test instance i for $M_i > 1$ times, each time randomly selecting neurons to be dropped. This is illustrated in Figure 3. The pdf of the RUL for a test instance i is now created with the M_i RUL predictions.

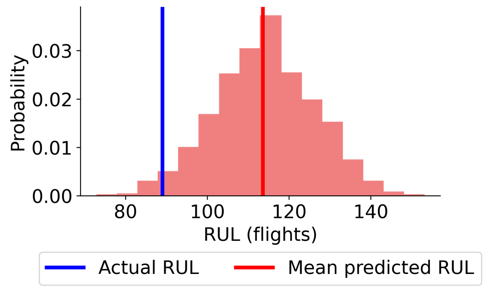
Figure 4 shows the obtained pdf of the RUL for engines $i \in \{53, 4, 86, 67\}$ of test set FD001. The pdf of the RUL of engine 53 is well centered around the actual RUL, and the variance is relatively low. The pdf of the RUL of engine 4 is well centered around the actual RUL as well, but the variance is larger, suggesting a larger uncertainty about the prediction. In contrast, the pdf's of the RUL of engines 86 and 67 are not well centered around the actual RUL. Moreover, the actual RUL of engine 67 falls outside the estimated RUL probability distribution.



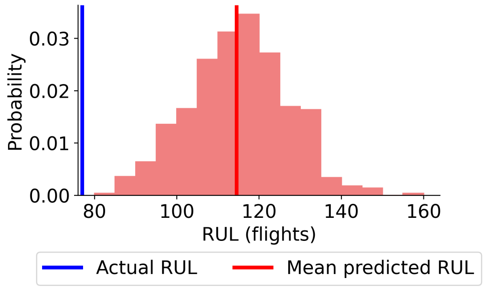
(a) PDF of RUL for engine 53, FD001.



(b) PDF of RUL for engine 4, FD001.



(c) PDF of RUL for engine 86, FD001.



(d) PDF of RUL for engine 67, FD001.

Figure 4. The estimated pfd of the RUL of four individual engines in the test set of FD001.

2.1. Metrics often used to evaluate RUL prognostics

The metrics often used to assess the performance of point RUL prognostics are the Mean Absolute Error (MAE), the Root Mean Square Error (RMSE) and the Mean Score. These metrics are computed based on the actual RUL vs. the predicted point RUL. When the pdf of the RUL is estimated instead, the MAE, RMSE and the Mean Score can be computed based on the actual RUL vs. the *mean* of the predicted RUL.

Formally, let N be the number of test instances in one C-MAPSS test set, and let y_i be the actual RUL for test instance i. Let $\hat{y}_{ij}$, $j \in \{1, 2, \ldots, M_i\}$, be the j^{th} RUL prediction for engine i. Let $\bar{y}_i$ be the mean predicted RUL of test instance i:

$$\bar{y}_i = \frac{1}{M_i} \sum_{j=1}^{M_i} \hat{y}_{ij}. \tag{1}$$

Then, when considering probabilistic RUL prognostics,

$$\text{MAE}^p = \frac{1}{N} \sum_{i=1}^{N} |\bar{y}_i - y_i|. \tag{2}$$

$$\text{RMSE}^p = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (\bar{y}_i - y_i)^2}. \tag{3}$$

$$\text{Mean Score}^p = \frac{1}{N} \sum_{i=1}^{N} s_i, \tag{4}$$

with

$$s_i = \begin{cases} e^{-\frac{\bar{y}_i - y_i}{\gamma}} - 1, & \bar{y}_i - y_i < 0 \\ e^{\frac{\bar{y}_i - y_i}{\delta}} - 1, & \bar{y}_i - y_i \geq 0 \end{cases},$$

with γ and δ user-defined metrics. For the C-MAPSS data set, $\gamma = 13$ and $\delta = 10$ are usually applied (Li et al., 2018).

Table 2. RMSE^p, MAE^p, and Mean Score^p with respect to the mean RUL prediction - C-MAPSS dataset.

Test set	RMSE^p	MAE^p	Mean Score^p
FD001	12.76	9.22	2.78
FD002	14.74	11.14	3.55
FD003	11.89	9.07	2.43
FD004	18.03	13.44	8.03

Table 2 shows the RMSE^p, MAE^p and Mean Score^p obtained for our probabilistic RUL prognostics when using the C-MAPSS dataset and a CNN with Monte Carlo dropout. Training the neural networ took between 12.1 (FD001) to 27.3 (FD002) seconds per epoch on a computer with an Intel Core i7 processor at 2.11 GHz and 8Gb RAM. Our results are comparable with state-of-the-art RUL prognostic results in (Xia et al., 2021).

However, these metrics do not fully capture the quality of the probabilistic RUL prognostics. The reliability and sharpness of the RUL prognostics is not evaluated, e.g, the variance of the generated pdf of the RUL. For example, for engine 4 (see Figure 4b) the absolute error with the mean predicted RUL is only 3.2 flights, and the Score with the mean predicted RUL is only 0.28. The mean predicted RUL is thus very close to the actual RUL. However, the standard deviation of the pdf of the RUL is large ($\sigma = 13.6$), suggesting a large uncertainty in the prediction. This large variance is not reflected in the mean predicted RUL, and thus neither in the RMSE^p, MAE^p and Mean Score^p metrics. Similarly, for engine 86 (see Figure 4c), the absolute error with the mean predicted RUL is 24.6 flights, and the score value with the mean predicted RUL is 10.67, which shows that the mean predicted RUL is far off the actual RUL. However, the actual RUL still falls within the pdf of the RUL. This is again not reflected in the mean RUL prediction and thus in the three metrics above. To analyze the full predicted pdf of the RUL with the corresponding uncertainty estimates, we introduce four additional metrics that characterize the reliability, the sharpness and the accuracy associated with the pdf's of the RUL.

3. Novel metrics to evaluate probabilistic RUL prognostics

In this section, we introduce the following novel metrics to characterize the reliability, the sharpness and the accuracy of probabilistic RUL prognostics (i.e, when estimating the pdf of the RUL): the Continuous Ranked Probability Score (CRPS), the weighted CRPS (CRPS^W), the α-Coverage, and the Reliability Score (RS). In the appendix, we provide the Python code to calculate the proposed metrics.

3.1. Continuous Ranked Probability Score (CRPS)

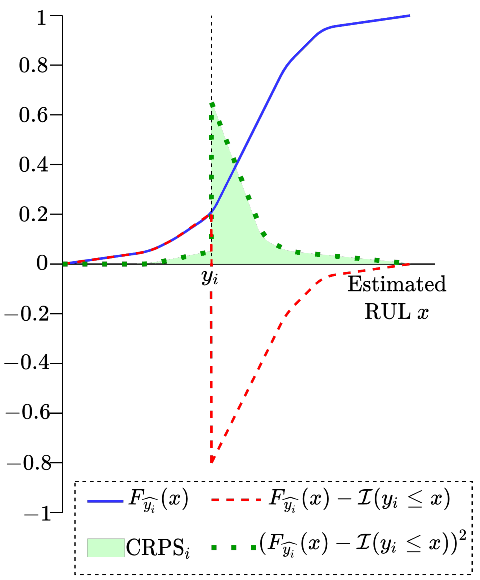


Figure 5. Illustration of the CRPS$_i$ metric for a single component i.

The Continuous Ranked Probability Score (CRPS) evaluates i) if the estimated RUL distribution is centered around the actual RUL of a component i, i.e., the accuracy of the RUL prognostic, and ii) if the variance of the RUL distribution is low, i.e., the sharpness of the RUL prognostic. In other words, a probabilistic RUL prognostic for a component i is best when all RUL predictions $\hat{y}_{ij}$ $j \in \{1, 2, \ldots, M_i\}$ are close to the actual RUL y_i.

CRPS has been used to evaluate probabilistic predictions for applications such as flight delays (Zoutendijk & Mitici, 2021), sea level pressure and surface temperature (Gneiting, Raftery, Westveld III, & Goldman, 2005) and electricity prices (Nowotarski & Weron, 2018). However, to the best of our knowledge, this metric has not yet been used to evaluate probabilistic RUL prognostics.

Let $F_{\widehat{y_i}}(x)$ denote the estimated, empirical CDF of the RUL of a component i. Then CRPS is as follows:

$$\text{CRPS} = \frac{1}{N}\sum_{i=1}^{N}\text{CRPS}_i, \tag{5}$$

$$\text{CRPS}_i = \int_{-\infty}^{\infty}(F_{\widehat{y_i}}(x) - \mathcal{I}\{y_i \le x\})^2 dx,$$

$$\text{with} \quad \mathcal{I}\{y_i \le x\} = \begin{cases} 1, & y_i \le x \\ 0, & y_i > x. \end{cases}$$

Intuitively, CRPS for a component i can be seen as a probabilistic generalization of the absolute error $|y_i - \hat{y}_i|$. Specifically, when calculating the CRPS of a point RUL prediction, we obtain the absolute error of this point RUL prediction. The smaller the CRPS metric is, the closer the RUL prediction is to the actual RUL. In an ideal case when a perfect RUL prediction without uncertainty (i.e., a point RUL prediction) is obtained, CRPS equals zero. A comprehensive explanation of this metric can be found in (Gneiting & Katzfuss, 2014).

Figure 5 shows a graphical representation of CRPS for a single, generic component i. The blue, solid line represents the empirical CDF of the RUL prognostic of this component i. The light-green area is the CRPS for this component i. This area (i.e., the CRPS value) is small if the accuracy and sharpness of the probabilistic RUL prognostic are high. In general, if the prognostics are accurate, then most RUL estimates are located close to the true RUL y_i. This is equivalent to a low CRPS value. If the prognostics are not only accurate, but also sharp, then the tails of the distribution are small and low. In this case, the CRPS value is smaller as well. Conversely, the CRPS value increases if the true RUL y_i falls outside the estimated RUL distribution (i.e., inaccurate prognostics).

3.2. Weighted CRPS (CRPS^W)

For most components and systems, overestimating the RUL is much more detrimental than underestimating the RUL (Li et al., 2018). A late prediction of the failure time is less desirable since missing a component failure may have severer consequences than replacing this component too early. We thus propose the weighted CRPS, which considers penalties for the RUL being overestimated/underestimated. Depending on the type of component, these penalties can be adjusted. In the case of safety critical component, for example, larger penalties are applied for RUL being overestimated. This is because a RUL overestimation may lead to a missed failure. The weighted CRPS is defined as follows:

$$\text{CRPS}^W = \frac{1}{N}\sum_{i=1}^{N}\text{CRPS}_i^W, \tag{6}$$

$$\begin{aligned}\text{CRPS}_i^W &= (2-\beta)\int_{-\infty}^{y_i}(F_{\widehat{y_i}}(x) - \mathcal{I}\{y_i \le x\})^2 dx \\ &\quad + \beta\int_{y_i}^{\infty}(F_{\widehat{y_i}}(x) - \mathcal{I}\{y_i \le x\})^2 dx, \\ &= (2-\beta)\int_{-\infty}^{y_i}(F_{\widehat{y_i}}(x))^2 dx + \beta\int_{y_i}^{\infty}(F_{\widehat{y_i}}(x) - 1)^2 dx,\end{aligned}$$

with $0 \le \beta \le 2$ an user-specific parameter. The magnitude of the penalty is specified through the weight β. The weight β is specified by the user, and depends on the domain application.

3.3. α-Coverage

CRPS evaluates the accuracy and sharpness of the probabilistic RUL prognostics. It is, however, also important to verify the reliability of the RUL predictions. To address this, we introduce the coverage of a RUL prediction, similar to (Baraldi et al., 2015). In this paper, however, we construct the coverage of a probabilistic RUL prognostic without assuming that this prognostic follows a specific distribution, such as the Gaussian distribution.

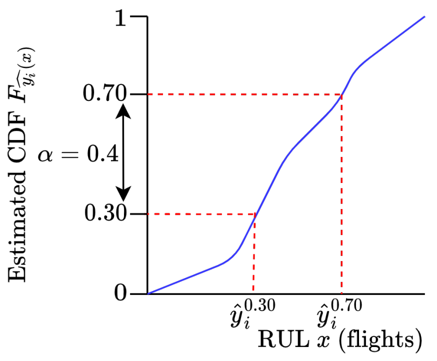


Figure 6. Illustration of the percentiles with the estimated CDF of the RUL of a test instance i

To calculate the coverage, we first construct a credible interval around the median of the estimated RUL distribution with width α. For example, let us assume that we have $M_i = 1000$ RUL predictions for a test instance i, i.e., $\hat{y}_{ij}, j \in \{1, 2, \ldots, M_i\}$. Let us consider the credible interval around the median with width $\alpha = 0.4 = 40\%$. Then, this credible interval is $[\hat{y}_i^{0.30}, \hat{y}_i^{0.70}]$, with $\hat{y}_i^{0.30}$ the RUL prediction belonging to the $50\% - 0.5\alpha = 30^{\text{th}}$ percentile. In our example, when we sort all $M_i = 1000$ predictions from small to large, this is the $j = 300^{\text{th}}$ RUL prediction $\hat{y}_{i,300}$. Also, $\hat{y}_i^{0.70}$ is the RUL prediction belonging to the $50\% + 0.5\alpha = 0.70^{\text{th}}$ percentile. In our example, when we sort all $M_i = 1000$ predictions from small to large, this is the $j = 700^{\text{th}}$ RUL prediction $\hat{y}_{i,700}$. The predicted probability that the actual RUL y_i of component i is within the credible interval $[\hat{y}_i^{0.30}, \hat{y}_i^{0.70}]$ is $\alpha = 40\%$. This example is illustrated in Figure 6.

We construct a credible interval with width $\alpha = 0.4$ for all $i \in \{1, 2, \cdots, N\}$ test instances. It is expected that for $\alpha = 40\%$ of the test instances, the actual RUL y_i is within the credible interval $[\hat{y}_i^{0.30}, \hat{y}_i^{0.70}]$. If the actual RUL of more than 40% of the test instances falls within the credible interval $[\hat{y}_i^{0.30}, \hat{y}_i^{0.70}]$, then the *uncertainty* for $\alpha = 0.4$ is *overestimated*. Otherwise, the *uncertainty* for $\alpha = 0.4$ is *underestimated*.

With the concept of a credible interval, the coverage of probabilistic RUL prognostics is defined as follows:

$$\alpha\text{-Coverage} = \frac{1}{N}\sum_{i=1}^{N} \mathcal{I}(\alpha)_i, \tag{7}$$

$$\text{with } \mathcal{I}(\alpha)_i = \begin{cases} 1, & y_i \in [\hat{y}_i^{0.5-0.5\alpha}, \hat{y}_i^{0.5+0.5\alpha}] \\ 0, & \text{Otherwise,} \end{cases}$$

where $\alpha \in [0, 1]$ is an user-defined parameter and $\hat{y}_i^k$ is the RUL prediction of the k^{th} percentile of the estimated RUL distribution of component i. The closer the coverage is to α, the more reliable the estimated RUL distribution is. The uncertainty is overestimated if the coverage is larger than α. Conversely, the uncertainty is underestimated if the coverage is smaller than α. For example, in Figure 4c, the true RUL does not fall within the 90% credible interval of the RUL distribution. If we predict a RUL distribution for ten individual components, we expect that for only one out of these ten components, the true RUL lies outside the 90% credible interval, as is the case in Figure 4c.

Last, if two RUL prediction methods have the same coverage for a width α, the method that provides tighter credible intervals is preferred. In other words, a higher sharpness of the RUL distributions is preferred. In this way, the predicted RUL distributions give a more precise picture of the actual RUL. A higher sharpness also leads to a lower CRPS. The tightness of the credible intervals, or the mean width of the credible intervals, is defined as (Baraldi et al., 2015):

$$\alpha\text{-Mean width} = \frac{1}{N}\sum_{i=1}^{N} (\hat{y}_i^{0.5+0.5\alpha} - \hat{y}_i^{0.5-0.5\alpha}). \tag{8}$$

3.4. Reliability Score (RS)

Though the Coverage metric indicates the reliability of the estimated RUL distribution, this reliability is evaluated only relative to a specific α. To conduct a generic, parameter-free reliability analysis of the estimated RUL distribution, we next introduce the Reliability Score (RS). We first introduce the concept of the reliability diagram.

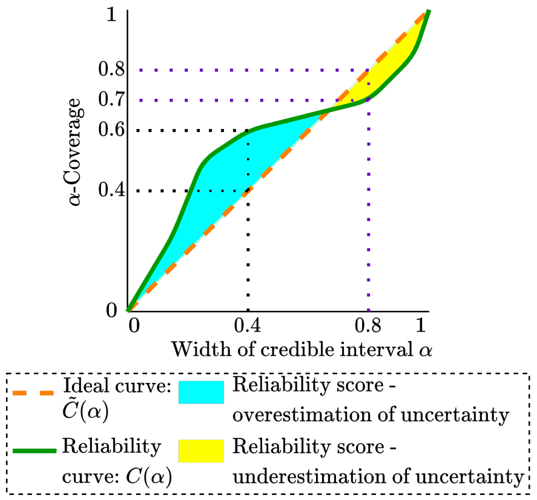


Figure 7. Illustration of the reliability diagram and the Reliability Scores.

For classification problems, a reliability diagram is used as a visual representation of the reliability of the uncertainty associated with the predictions. A reliability diagram is also referred to as a calibration curve. In (Saxena et al., 2008), the reliability diagram is proposed as a RUL prognostic metric. Here, the problem of RUL prognostics is posed as a classification problem with multiple classes. In contrast, in (Vandal, Livingston, Piho, & Zimmerman, 2018), the reliability diagram is defined based on the concept of coverage (see Section 3.3). In doing so, a regression problem does not have to be posed as a multi-class classification problem to construct a reliability diagram. The authors of (Vandal et al., 2018) determine a reliability diagram for flight delay estimations. Similarly, we define a reliability curve $C(\alpha)$ based on $\alpha-$Coverage (see Eq. (7)) for probabilistic RUL prognostics, i.e., $C(\alpha) = \{\alpha\text{-Coverage}, \alpha \in \{0.00, 0.01, 0.02, \ldots, 1.00\}\}$. The reliability diagram is then a visual representation of this reliability curve. Figure 7 gives an illustration of a reliability curve.

The reliability diagram is used to visually inspect whether the uncertainty associated with the RUL predictions is over- or underestimated. For example, when $\alpha = 0.4$, the ideal coverage would be 0.4 as well. In this case, the actual RUL of 40% of the test instances would fall inside a credible interval with width $\alpha = 0.4$. However, in the example in Figure 7, the 0.4-Coverage is 0.6, i.e., the actual RUL of 60% of the test instances falls inside the credible interval, instead of 40% of the test instances. The uncertainty of the RUL prognostics is thus overestimated.

In contrast, the uncertainty of the RUL prognostics is underestimated at $\alpha = 0.8$ in Figure 7. Here, the actual RUL of only 70% of the test instances falls inside the credible interval with a width of $\alpha = 0.8$.

In general, for classification problems, the Brier Score (Brier, 1950) is used to quantify the reliability of predictions. However, in our adaption of the reliability diagram, each test instance may fall into multiple credible intervals. The calculation of the Brier Score is thus not directly applicable. To address this, we define the following Reliability scores (RS) to quantify the reliability of the RUL prognostics:

$$\mathrm{RS}^{\text{under}} = \int_0^1 \mathcal{I}\{C(\alpha) \leq \alpha\}(\alpha - C(\alpha))d\alpha, \tag{9}$$

$$\mathrm{RS}^{\text{over}} = \int_0^1 (1 - \mathcal{I}\{C(\alpha) \leq \alpha\})(C(\alpha) - \alpha)d\alpha, \tag{10}$$

$$\mathrm{RS}^{\text{total}} = \mathrm{RS}^{\text{under}} + \mathrm{RS}^{\text{over}}, \tag{11}$$

$$\text{with } \mathcal{I}\{C(\alpha) \leq \alpha\} = \begin{cases} 1, & C(\alpha) \leq \alpha \\ 0, & \text{Otherwise} \end{cases}.$$

The $\mathrm{RS}^{\text{over}}$ quantifies the overestimation and $\mathrm{RS}^{\text{under}}$ the underestimation of the *uncertainty* associated with the probabilistic RUL prognostics. Let $\tilde{C}(\alpha) = \{\alpha, \alpha \in \{0.00, 0.01, \ldots, 1.00\}$ be the ideal curve, i.e., the curve where the Coverage is exactly the width of the credible interval α. To quantify the extent to which the uncertainty associated with the probabilistic RUL prognostics is *underestimated*, we calculate the area $\mathrm{RS}^{\text{under}}$ between the ideal curve and the reliability curve $C(\alpha)$ when the reliability curve is *below* the ideal curve (i.e., $C(\alpha) \leq \alpha$, yellow area in Figure 7). To quantify the extent to which the uncertainty associated with the probabilistic RUL prognostics is *overestimated*, we calculate the area $\mathrm{RS}^{\text{over}}$ between the ideal curve and the reliability curve $C(\alpha)$ when the reliability diagram is *above* the ideal curve (i.e., $C(\alpha) \geq \alpha$, blue area in Figure 7). The total RS ($\mathrm{RS}^{\text{total}}$) is then the sum of $\mathrm{RS}^{\text{over}}$ and $\mathrm{RS}^{\text{under}}$.

4. Results

In this section, we evaluate our metrics for the obtained probabilistic RUL prognostics for the turbofan engines in the C-MAPSS dataset. These probabilistic RUL prognostics are obtained with a CNN with Monte Carlo Dropout (see Section 2). Figure 9 shows the obtained probabilistic RUL prognostics, and Table 3 shows the corresponding values of the four proposed metrics. The CRPS is lowest for data subset FD003 (6.56), and highest for data subset FD004 (10.09). This is in line with the obtained MAE, which is also lowest for data subset FD003 and highest for data subset FD004. CRPS thus gives a good overview of the general performance of probabilistic RUL prognostics. Moreover, in contrast with MAE, the sharpness and accuracy of the estimated RUL distributions are also reflected by the CRPS values.

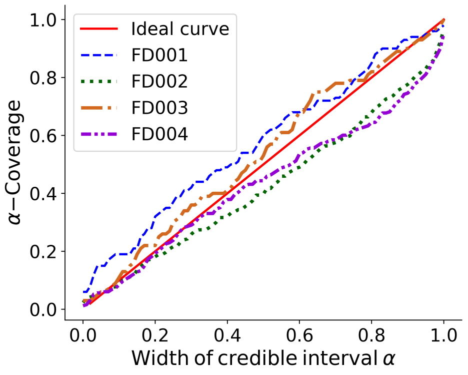
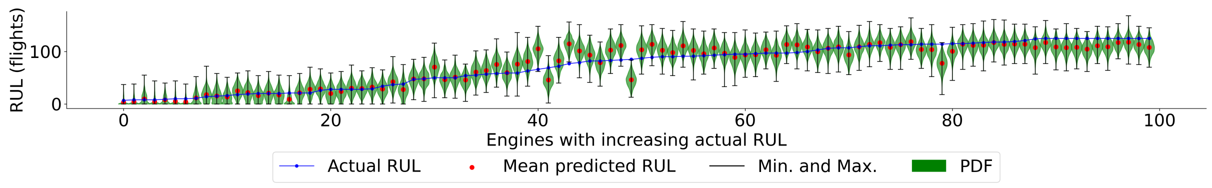


Figure 8. Reliability diagrams - C-MAPSS data subsets.

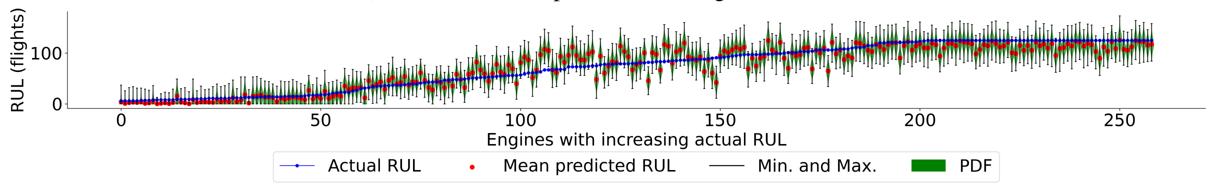
For data subset FD002, the CRPS^W is lower than the CRPS. This indicates that for this dataset, the RUL is relatively often underestimated. In contrast, for data subset FD003, the CRPS^W is higher than the CRPS. This indicates that for FD003, the RUL is relatively often overestimated. The weighted CRPS, compared to the standard CRPS, thus gives a good indication on whether the RUL is usually over- or underestimated.

The reliability diagram of the four data subsets is shown in Figure 8. For data subsets FD001 and FD003, the uncertainty of the RUL prognostics is slightly overestimated. In other words, the prognostics indicate that the RUL lies in an interval with a certain probability. However, these probabilities are too small, relative to the actual number of times the RUL falls within these intervals. For example, let us consider the 0.5-Coverage of data subset FD001. Here, the estimated probability that a test instance falls inside its credible interval with width 0.5 equals 0.5. We thus expect that 50% of the test instances fall inside their credible interval with width 0.5, and 50% fall outside their credible interval. However, 60% of the test instances fall inside their credible interval with width 0.5, i.e., the observed probability is 0.6 instead of 0.5. This shows that the uncertainty associated with the RUL estimates is overestimated. In contrast, for data subsets FD002 and FD004, the uncertainty is underestimated, i.e., the prognostics indicate that the RUL lies in an interval with a certain probability. These probabilities are too high relative to the actual number of times the RUL falls within these intervals. Table 3 shows that the over- and underestimation of the uncertainty associated with the RUL prognostics is well quantified by the reliability scores.

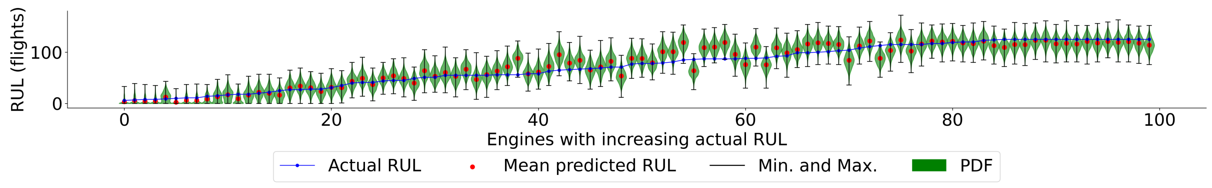
Table 3 also shows the 0.5-Coverage and the 0.95-Coverage. Also this metric indicates that the RUL prognostics for data subsets FD001 and FD003 overestimate the uncertainty associated with the prognostics, while for data subsets FD002



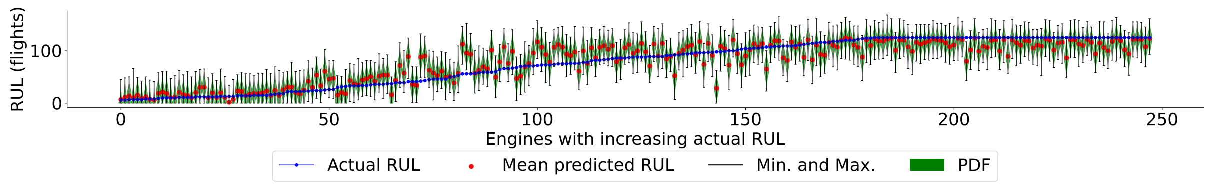
(a) Individual RUL predictions of engines, FD001.



(b) Individual RUL predictions of engines, FD002.



(c) Individual RUL predictions of engines, FD003.



(d) Individual RUL predictions of engines, FD004.

Figure 9. Mean predicted RUL and associated RUL distribution - FD001, FD002, FD003, FD004 of the C-MAPSS test sets.

Table 3. Results for the four C-MAPSS data sets with respect to the uncertainty estimation.

Test set	MAE^p	CRPS	CRPS^W ($\beta = 1.5$)	Coverage- $\alpha = 0.5$	Mean width- $\alpha = 0.5$	Coverage- $\alpha = 0.95$	Mean width- $\alpha = 0.95$	RS$^{\text{over}}$	RS$^{\text{under}}$	RS$^{\text{total}}$
FD001	9.22	6.97	7.03	0.60	16.9	0.95	48.0	0.073	0.001	0.074
FD002	11.14	8.44	7.80	0.40	13.2	0.83	38.0	0.001	0.077	0.078
FD003	9.07	6.56	7.27	0.53	15.4	0.93	44.7	0.034	0.001	0.035
FD004	13.44	10.09	10.38	0.44	15.5	0.81	43.8	0.001	0.065	0.065

Table 4. Performance metrics for engines 53, 4, 86 and 67 in the test set of FD001.

Engine number i	Actual RUL y_i (flights)	Mean predicted RUL $\bar{y}_i$ (flights)	Error: $y_i - \bar{y}_i$ (flights)	Score^p s_i	CRPS$_i$	CRPS$_i^W$ $\beta = 1.5$	$\mathcal{I}(\alpha)_i$ $\alpha = 0.5$	$\hat{y}_i^{0.75} - \hat{y}_i^{0.25}$	$\mathcal{I}(\alpha)_i$ $\alpha = 0.95$	$\hat{y}_i^{0.975} - \hat{y}_i^{0.025}$
53	26	29.0	-3.0	0.35	2.96	3.72	1	15	1	45
4	82	78.8	3.2	0.28	3.49	2.68	1	19	1	54
86	89	113.6	-24.6	10.67	17.98	26.96	0	16	1	48
67	77	114.5	-37.5	41.61	30.74	46.11	0	16	0	46

and FD004 the uncertainty associated with the prognostics is underestimated. Moreover, the mean width of the 0.95 credible interval is large, ranging from 38.0 flights (data subset FD002) to 48.0 flights (data subset FD001), i.e., the sharpness of the RUL distributions is low.

4.1. RUL prognostics for individual engines

In this section, we analyze our proposed metrics for probabilistic RUL prognostics for four specific engines 53, 4, 86 and 67 of data subset FD001, see Table 4. The probabilistic RUL prognostics of these four engines is already shown in

Figure 4.

For engine 53, the actual RUL is very close to the mean predicted RUL. The error is thus only -3.0 flights. Also CRPS_{53}, which is the generalization of the absolute error, is only 2.96. However, most of the mass of the predicted distribution of the RUL is on the right of the actual RUL, i.e., the RUL is overestimated (see Figure 4a). This is reflected in the relatively high CRPS_{53}^{W} of 3.72. The actual RUL falls both within the $\alpha = 0.5$ and $\alpha = 0.95$ credible interval, and the widths of these intervals (15 and 45 flights respectively) are relatively small compared to engines 4, 86 and 67.

For engine 4, the mean predicted RUL is close to the actual RUL, with an error of 3.2 flights. Thus CRPS_4 is only 3.49. Also, $\text{CRPS}_4^W = 2.68$, which is less than CRPS_4. This is because most of the mass of the predicted pdf of the RUL is on the left of the actual RUL, i.e., the RUL is underestimated (see Figure 4b). The $\alpha = 0.5$ and $\alpha = 0.95$ credible interval both contain the actual RUL, but the width of these intervals (19 and 54 flights respectively) is relatively large compared to the other 3 engines. The low sharpness of this RUL distribution is thus reflected in the large widths of the credible intervals.

For engines 86 and 67, the mean predicted RUL is far off the actual RUL. This is reflected in the high CRPS values of 17.98 and 30.74, respectively. Moreover, nearly all the mass of the predicted pdf of the RUL of both engines is on the right of the actual RUL, i.e., the RUL is overestimated (see Figures 4c and 4d). The weighted CRPS metric is thus 26.96 and 46.11, respectively. This is higher than the standard CRPS metric for these two engines. The actual RUL of engine 86 falls within the $\alpha = 0.95$ credible interval, but the actual RUL of engine 67 does not.

5. Conclusions

In this paper, we have introduced novel metrics to evaluate the predicted probability distribution (pdf) of the RUL of components. The CRPS and CRPS^W metrics evaluate the accuracy and sharpness of the estimated RUL distributions. The α-Coverage and Reliability Scores evaluate the reliability of the RUL prognostics.

We illustrate the four metrics for probabilistic RUL prognostics of the turbofan engines in the C-MAPSS dataset. We obtain these probabilistic RUL prognostics using a CNN with Monte Carlo dropout. The results show the distribution of the RUL of the turbofan engines is well estimated using this method. Moreover, the accuracy, sharpness and reliability of the obtained probabilistic RUL prognostics are shown to be well evaluated by our proposed metrics. Future studies that determine probabilistic RUL prognostics could therefore benefit from evaluating their results using these proposed metrics.

References

Babu, G. S., Zhao, P., & Li, X. L. (2016). Deep convolutional neural network based regression approach for estimation of Remaining Useful Life. In *International conference on database systems for advanced applications* (pp. 214–228).

Baraldi, P., Mangili, F., & Zio, E. (2015). A prognostics approach to nuclear component degradation modeling based on gaussian process regression. *Progress in Nuclear Energy*, *78*, 141–154.

Biggio, L., Wieland, A., Chao, M. A., Kastanis, I., & Fink, O. (2021). Uncertainty-aware prognosis via deep gaussian process. *IEEE Access*, *9*, 123517–123527.

Brier, G. W. (1950). Verification of forecasts expressed in terms of probability. *Monthly Weather Review*, *78*(1), 1–3.

de Pater, I., & Mitici, M. (2021). Predictive maintenance for multi-component systems of repairables with remaining-useful-life prognostics and a limited stock of spare components. *Reliability Engineering & System Safety*, *214*, 107761.

de Pater, I., Reijns, A., & Mitici, M. (2022). Alarm-based predictive maintenance scheduling for aircraft engines with imperfect remaining useful life prognostics. *Reliability Engineering & System Safety*, 108341.

Gal, Y., & Ghahramani, Z. (2016). Dropout as a Bayesian approximation: Representing model uncertainty in deep learning. In *International conference on machine learning* (pp. 1050–1059).

Gal, Y., Hron, J., & Kendall, A. (2017). Concrete dropout. *arXiv preprint arXiv:1705.07832*.

Gneiting, T., & Katzfuss, M. (2014). Probabilistic forecasting. *Annual Review of Statistics and Its Application*, *1*, 125–151.

Gneiting, T., Raftery, A. E., Westveld III, A. H., & Goldman, T. (2005). Calibrated probabilistic forecasting using ensemble model output statistics and minimum crps estimation. *Monthly Weather Review*, *133*(5), 1098–1118.

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lall, P., Lowe, R., & Goebel, K. (2011). Prognostics and health monitoring of electronic systems. In *2011 12th international conference on thermal, mechanical & multi-physics simulation and experiments in microelectronics and microsystems* (pp. 1–17).

Lee, J., & Mitici, M. (2020). An integrated assessment of safety and efficiency of aircraft maintenance strategies using agent-based modelling and stochastic petri nets. *Reliability Engineering & System Safety*, *202*, 107052.

Lee, J., & Mitici, M. (2022). Multi-objective design of aircraft maintenance using gaussian process learning and adaptive sampling. *Reliability Engineering & System*

Safety, *218*, 108123.

Le Son, K., Fouladirad, M., & Barros, A. (2016). Remaining useful lifetime estimation and noisy gamma deterioration process. *Reliability Engineering & System Safety*, *149*, 76–87.

Li, X., Ding, Q., & Sun, J.-Q. (2018). Remaining useful life estimation in prognostics using deep convolution neural networks. *Reliability Engineering & System Safety*, *172*, 1–11.

Mitici, M., & de Pater, I. (2021). Online model-based remaining-useful-life prognostics for aircraft cooling units using time-warping degradation clustering. *Aerospace*, *8*(6), 168.

Nguyen, K. T., & Medjaher, K. (2019). A new dynamic predictive maintenance framework using deep learning for failure prognostics. *Reliability Engineering & System Safety*, *188*, 251–262.

Nowotarski, J., & Weron, R. (2018). Recent advances in electricity price forecasting: A review of probabilistic forecasting. *Renewable and Sustainable Energy Reviews*, *81*, 1548–1568.

Ramasso, E., & Saxena, A. (2014). Review and analysis of algorithmic approaches developed for prognostics on cmapss dataset. In *Annual conference of the prognostics and health management society 2014.*

Saxena, A., Celaya, J., Balaban, E., Goebel, K., Saha, B., Saha, S., & Schwabacher, M. (2008). Metrics for evaluating performance of prognostic techniques. In *2008 international conference on prognostics and health management* (pp. 1–17).

Saxena, A., Celaya, J., Saha, B., Saha, S., & Goebel, K. (2009). Evaluating algorithm performance metrics tailored for prognostics. In *2009 IEEE aerospace conference* (pp. 1–13).

Saxena, A., & Goebel, K. (2008). Turbofan engine degradation simulation data set. *NASA Ames Prognostics Data Repository (https://ti.arc.nasa.gov/tech/dash/groups/pcoe/prognostic-data-repository/)*, 878–887.

Vandal, T., Livingston, M., Piho, C., & Zimmerman, S. (2018). Prediction and uncertainty quantification of daily airport flight delays. In *International conference on predictive applications and APIs* (pp. 45–51).

Xia, J., Feng, Y., Lu, C., Fei, C., & Xue, X. (2021). LSTM-based multi-layer self-attention method for Remaining Useful Life estimation of mechanical systems. *Engineering Failure Analysis*, *125*, 105385.

Zoutendijk, M., & Mitici, M. (2021). Probabilistic flight delay predictions using machine learning and applications to the flight-to-gate assignment problem. *Aerospace*, *8*(6), 152.

BIOGRAPHIES

Ingeborg de Pater is a PhD candidate at the Faculty of Aerospace Engineering, Delft University of Technology, the Netherlands. Her research interests are predictive aircraft maintenance scheduling and Remaining-Useful-Life estimation of aircraft components.

Mihaela Mitici is an Assistant professor at the Faculty of Aerospace Engineering, Delft University of Technology. She has a PhD in Stochastic Operations Research, Department of Applied Mathematics, University of Twente, the Netherlands. Her research interests are stochastic operations research, stochastic processes and machine learning algorithms with applications to predictive aircraft maintenance scheduling and RUL prognostics.

APPENDIX: PYTHON CODE FOR THE NOVEL METRICS

```
#-------------Necessary packages-----------------#
import numpy as np
import matplotlib.pyplot as plt
import sys

def CRPS(true_RULs, RUL_distributions, beta = 1.5):
    """
    This function calculates the CRPS and the weighted CRPS.
    Parameters
    ----------
    true_RULs: Dictionary
        A dictionary with for each test instance (key, integer), the true RUL (value).
    RUL_distributions : Dictionary
        A dictionary with for each test instance (key, integer), a list (value) with all RUL predictions
        of this test instance. true_RULs and RUL distributions should have the same set of keys.
    beta :  Float between 1 (included) and 2 (included)
        Penalty for overestimating the RUL relative to underestimating the RUL.
        The default is 1.5.
    Returns
    -------
    crps : Float
        The CRPS metric.
    weighted_crps : Float
        The weighted CRPS metric,
    """
    crps_sum  = 0 #The value of the sum of the CRPS metric
    weighted_crps_sum = 0 #The value of the sum of the weighted CRPS metric

    #Calculate the CRPS and the weighted CRPS for each individual test instance
    for i in true_RULs.keys():
        #Initiliaze the CRPS and the weighted CRPS for test instance i
        crps_i = 0
        weighted_crps_i = 0

        #Get the probability distribution of the RUL of test instance i, and the true RUL
        distribution = RUL_distributions.get(i)
        true_RUL = true_RULs.get(i)
        distribution.sort()
        number_of_predictions = len(distribution) #The number of RUL predictions in the distribution

        for j in range(0, number_of_predictions -1, 1): #Go over all the predictions
            #Calculate the distance between two RUL predictions
            RUL_prediction = distribution[j]
            next_RUL_prediction = distribution[j+1]
            delta_RUL = next_RUL_prediction - RUL_prediction

            #Each RUL prediction has a probability of 1 over the number of predictions.
            #We use j+1, since j starts at 0, and since we consider the CDF
            probability = (j+1) / number_of_predictions

            #Check if the RUL prediction is larger, or smaller than the true RUL,
            #and update the CRPS and the weighted CRPS accordingly
            if RUL_prediction < true_RUL:
                probability_squared = probability ** 2
                crps_i = crps_i + (probability_squared * delta_RUL)
                weighted_crps_i = weighted_crps_i + (2 - beta) * (probability_squared * delta_RUL)
            else:
                probability_minus_one = probability - 1
                probability_squared = probability_minus_one ** 2
                crps_i = crps_i + (probability_squared * delta_RUL)
                weighted_crps_i = weighted_crps_i + beta * (probability_squared * delta_RUL)

        #Also consider the difference between the true RUL and the last prediction
        last_prediction = distribution[-1]
        if last_prediction < true_RUL:
             crps_i = crps_i + (1 * (true_RUL - last_prediction))
```

```python
                weighted_crps_i = weighted_crps_i + (2 -beta) * (1 * (true_RUL - last_prediction))

        #Also consider the difference between the true RUL and the first prediction
        first_prediction = distribution[0]
        if first_prediction > true_RUL:
            crps_i = crps_i + (1 * (first_prediction-true_RUL))
            weighted_crps_i = weighted_crps_i + beta * (1 * (first_prediction-true_RUL))

        #Update the sum of the CRPS and the sum of the weighted CRPS
        crps_sum = crps_sum + crps_i
        weighted_crps_sum = weighted_crps_sum + weighted_crps_i
    #Take the average value of the CRPS and the weighted CRPS
    crps = crps_sum / len(RUL_distributions.keys())
    weighted_crps = weighted_crps_sum / len(RUL_distributions.keys())
    return crps, weighted_crps

def coverage(true_RULs, RUL_distributions, alpha):
    """
    This function computes the alpha-coverage and corresponding alpha-mean width.
    Parameters
    ----------
    true_RULs: Dictionary
        A dictionary with for each test instance (key, integer), the true RUL (value).
    RUL_distributions : Dictionary
        A dictionary with for each test instance (key, integer), a list (value) with all RUL predictions
        of this test instance. true_RULs and RUL distributions should have the same set of keys.
    alpha : Float between 0 (included) and 1 (included)
        The desired width of the credible interval.
    Returns
    -------
    coverage : Float between 0 (included) and 1 (included)
        The coverage belonging to alpha.
    mean_width : Float
        The mean width of the credible interval belonging to alpha.
    """
    #Initialize the parameters of the credible interval
    total_width = 0 #Total width of all credible intervals
    in_ci = 0 #The number of components for which the true RUL falls within the credible interval
    percentile_lower = 0.5 - 0.5 * alpha #Lower percentile of the credible interval
    percentile_higher = 0.5 + 0.5 * alpha #Upper percentile of the credible interval

    #Check for each test instance i if the true RUL falls inside,
    #or outside the credible interval of test instance i
    for i in true_RULs.keys():
        #Get the probability dstributions of the RUL test instance i, and the true RUL
        distribution = RUL_distributions.get(i)
        true_RUL = true_RULs.get(i)
        distribution.sort()
        number_of_predictions = len(distribution) #The number of RUL predictions in the distribution

        #Get the indeces of the RUL predictions belonging to the considered percenticles.
        #We use -1, since a list in python starts at 0 instead of 1
        index_lower = max(0,  int(percentile_lower * number_of_predictions) - 1)
        index_higher = int(percentile_higher * number_of_predictions) - 1
        lower_bound_ci = distribution[index_lower] #Lower bound credible interval
        upper_bound_ci = distribution[index_higher] #Upper bound credible interval

        #Check if the true RUL is within the credible interval
        if true_RUL >= lower_bound_ci and true_RUL <= upper_bound_ci:
            in_ci = in_ci + 1

        #Update the total width of all credible interval
        total_width = total_width + (upper_bound_ci - lower_bound_ci)
    #Calculate the coverage and the mean width of the credible interval
    coverage = in_ci / len(true_RULs.keys())
    mean_width = total_width /  len(true_RULs.keys())
    return coverage, mean_width
```

```python
def area_under(x_1, x_2, f_1, f_2):
    """
    This functions calculates the area between the ideal curve and the reliability curve, between
    x_1 and x_2. Here, the reliability curve is under the ideal curve between x_1 and x_2.
    Parameters
    ----------
    x_1 : Float
        Start value of alpha.
    x_2 : Float
        End value of alpha.
    f_1 : Float
        Coverage at alpha = x_1.
    f_2 : Float
        Coverage at alpha = x_2.
    Returns
    -------
    area : Float
        Area between the ideal curve and the reliability curve, between x_1 and x_2.
    """
    area = (x_2 - f_2) * (x_2 - x_1) - 0.5 * (x_2 - x_1) * (x_2 - x_1)
    area = area + 0.5 * (x_2 - x_1) * (f_2 - f_1)
    return area

def area_above(x_1, x_2, f_1,  f_2):
    """
    This functions calculates the area between the ideal curve and the reliability curve, between
    x_1 and x_2. Here, the reliability curve is above the ideal curve between x_1 and x_2.
    Parameters
    ----------
    x_1 : Float
        Start value of alpha.
    x_2 : Float
        End value of alpha.
    f_1 : Float
        Coverage at alpha = x_1.
    f_2 : Float
        Coverage at alpha = x_2 .
    Returns
    -------
    area : Float
        Area between the ideal curve and the reliability curve, between x_1 and x_2.
    """
    area = (f_1 - x_1) * (x_2 - x_1) - 0.5 * (x_2 - x_1) * (x_2 - x_1)
    area = area + 0.5 * (x_2 - x_1) * (f_2 - f_1)
    return area

def reliability_score(true_RULs, RUL_distributions, name, stepsize = 0.01 ):
    """
    This functions calculates the reliability scores (under, over and total),
    and plots the reliability diagram.
    Parameters
    ----------
    true_RULs: Dictionary
        A dictionary with for each test instance (key, integer), the true RUL (value).
    RUL_distributions : Dictionary
        A dictionary with for each test instance (key, integer), a list (value) with all RUL predictions
        of this test instance. true_RULs and RUL distributions should have the same set of keys.
    Returns
    -------
    RS_total : Float
        Total Reliability Score.
    RS_under : Float
        Reliability Score - underestimation of uncertainty.
    RS_over : Float
        Reliability Score - overestimation of uncertainty.
    """
```

```
#-------------Calculate the Reliability curve
reliability_curve = []
for alpha in np.arange(0, 1 + sys.float_info.epsilon, stepsize): #One is included
    alpha_coverage = coverage(true_RULs, RUL_distributions, alpha)[0]
    reliability_curve.append(alpha_coverage)

#-------------Plot the reliability diagram
ideal_curve = list(np.arange(0, 1 + sys.float_info.epsilon, stepsize)) #ideal curve, where y = x
fig, ax = plt.subplots()
ax.set_ylabel(r"$\alpha\mathrm{-Coverage}$", fontsize = 16)
ax.set_xlabel(r"$\mathrm{Width \: of \: credible \: interval \: }\alpha$", fontsize = 16)
ax.plot(ideal_curve, ideal_curve, label = "Ideal curve", c = "red")
ax.plot(ideal_curve,reliability_curve , label = "Reliability curve", color = "blue", \
        linestyle = "dashed")
ax.legend()
plt.show()

#-------------Calculate the reliability score
RS_under = 0 #The reliability score: underestimation of the uncertainty
RS_over = 0 #The reliability score: overestimation of the uncertainty

for alpha in np.arange(0, 1, stepsize): #Loop over all alpha's
    next_alpha = alpha + stepsize
    coverage_alpha = coverage(true_RULs, RUL_distributions, alpha)[0]
    coverage_next_alpha =  coverage(true_RULs, RUL_distributions, next_alpha)[0]

    #If the reliability curve is beneath the ideal curve:
    if coverage_alpha <= alpha and coverage_next_alpha <= next_alpha:
        surface = area_under(alpha, next_alpha, coverage_alpha, coverage_next_alpha)
        RS_under = RS_under + surface

    #If the reliability curve is above the ideal curve:
    elif coverage_alpha >= alpha and coverage_next_alpha >= next_alpha:
        surface = area_above(alpha, next_alpha, coverage_alpha, coverage_next_alpha)
        RS_over = RS_over + surface

    #If the reliability curve starts under the ideal curve, and ends above the ideal curve
    elif coverage_alpha <= alpha and coverage_next_alpha >= next_alpha:
        #Find the place where the reliability curve crosses the ideal curve
        dy = coverage_next_alpha - coverage_alpha
        a = dy / stepsize
        alpha_cross = (coverage_alpha - a * alpha) / (1-a)
        coverage_cross = alpha_cross

        #Calculate the surface under the ideal curve
        surface_under = area_under(alpha, alpha_cross, coverage_alpha, coverage_cross)
        RS_under = RS_under + surface_under
        #Calculate the surface above the ideal curve
        surface_above = area_above(alpha_cross, next_alpha, coverage_cross, coverage_next_alpha)
        RS_over = RS_over + surface_above

    #If the reliability curve starts above the ideal curve, and ends under the ideal curve
    elif coverage_alpha >= alpha and coverage_next_alpha <= next_alpha:
        #Find the place where the reliability curve crosses the ideal curve
        dy = coverage_next_alpha - coverage_alpha
        a = dy / stepsize
        alpha_cross = (coverage_alpha - a * alpha) / (1-a)
        coverage_cross = alpha_cross

        #Calculate the surface above the ideal curve
        surface_above = area_above(alpha, alpha_cross, coverage_alpha, coverage_cross)
        RS_over = RS_over + surface_above
        #Calculate the surface under the ideal curve
        surface_under = area_under(alpha_cross, next_alpha, coverage_cross, coverage_next_alpha)
        RS_under = RS_under + surface_under
RS_total = RS_under + RS_over #Total reliability score
return RS_total, RS_under, RS_over
```