POLITECNICO
MILANO 1863

Digital Twin for Health and Usage Monitoring Report

a.y. 2024-2025

Enrico Rozzi
enrico.rozzi@mail.polimi.it
258655

Lorenzo Vignoli
lorenzo.vignoli@asp-poli.it
252085

Contents

1 Pressure Vessel

Two Digital Twins have been developed for pressure vessels, one containing pressurized gas and the other an isothermal liquid.

In the first Digital Twin, a crack propagates according to established mechanical laws, such as the Paris law and the Palmgren-Miner rule, allowing the estimation of the expected life cycle cost through forced sampling.

In the second Digital Twin, damage is an unknown leakage, indirectly measured by flow rate sensors. To reduce computational overhead, an Artificial Neural Network (ANN) approximates flow sensor outputs to infer leakage size, estimated probabilistically via Monte Carlo Markov Chain (MCMC) with the Metropolis-Hastings algorithm.

1.1 Modeling activity

1.1.1 Pressurized gas pressure vessel

The first pressure vessel consists of a cylindrical shell with a hemispherical end and is filled with pressurized gas. Daily pressure cycles induce fatigue, leading to defects in the component.

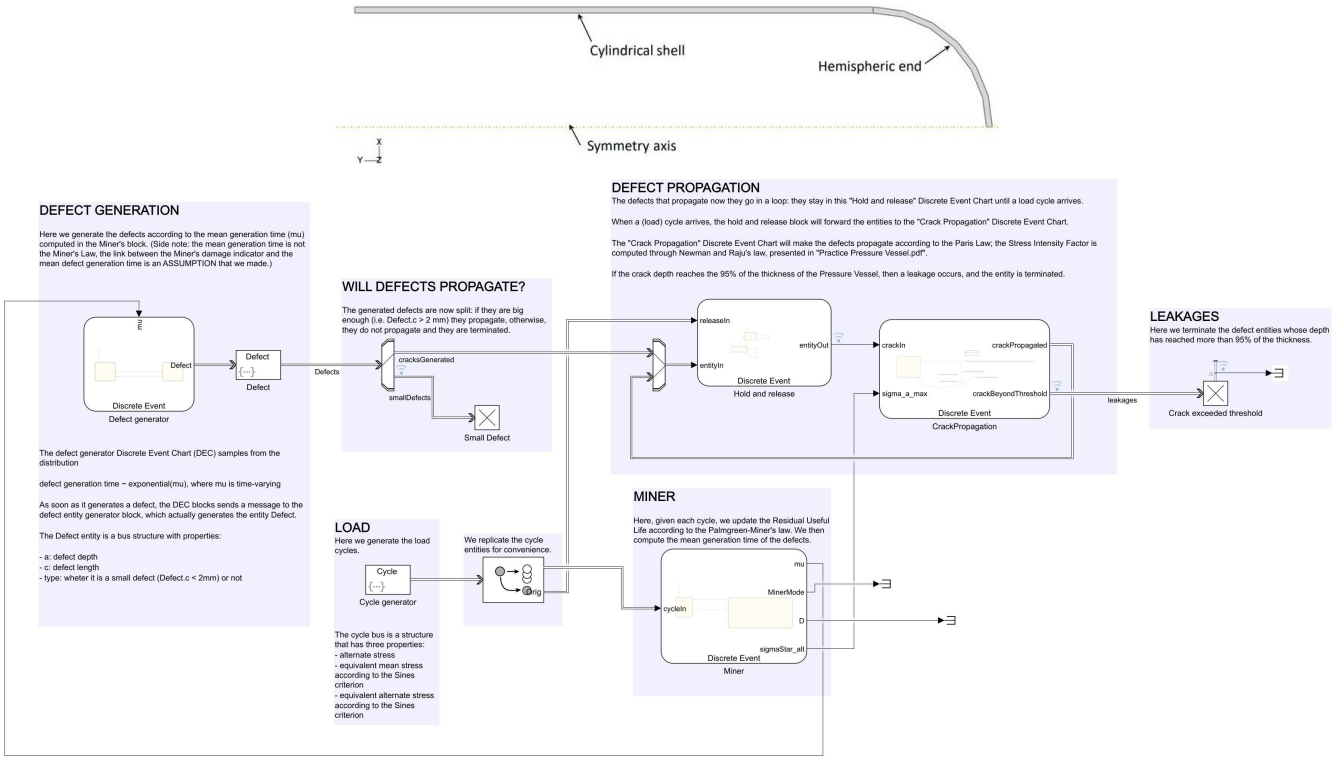


Figure 1: pressurized gas pressure vessel

The simplified Simulink model is shown in *figure* 1. Its functionalities are described as follows:

- **Load generation**: the vessel is assumed to undergo one cycle per day ($R = 0$), with the maximum pressure normally distributed as $\Delta p \sim \mathcal{N}(21\,\text{MPa}, 1\,\text{MPa}^2)$. Axial and circumferential (hoop) stress components are analytically calculated as:

$$\sigma_{axial,a} = \frac{p \cdot D_m}{4 \cdot t} \cdot \frac{1}{2}, \quad \sigma_{axial,m} = \sigma_{axial,a},$$

$$\sigma_{\theta,a} = \frac{p \cdot D_m}{2 \cdot t} \cdot \frac{1}{2}, \quad \sigma_{\theta,m} = \sigma_{\theta,a}.$$

Since radial stress is negligible for small thicknesses, the equivalent stresses are computed using the Sines criterion:

$$\sigma_a^* = \sqrt{\sigma_{axial,a}^2 + \sigma_{\theta,a}^2 - \sigma_{axial,a} \cdot \sigma_{\theta,a}},$$

$$\sigma_m^* = \sigma_{axial,m} + \sigma_{\theta,m},$$

As $R = 0$:

$$\sigma_{max}^* = \sigma_a^* + \sigma_m^*.$$

- **Miner Residual Useful Life**: used to evaluate the Mean Time to Generation μ, based on Miner damage parameter D. Given the Sines equivalent alternating stress:

$$\sigma_a' = \frac{\sigma_a^*}{1 - \frac{\sigma_m^*}{UTS}}$$

To compute the degree of damage of the component due to multiple stress cycles, the number of cycles to failure associated with each σ_a' is calculated using the S-N curve analytical expression:

$$\sigma_a' = A \cdot N_f^b$$

The Miner damage indicator is continuously updated, starting from $D = 0$, as more cycles pass. At each cycle, the damage increases by the ratio between the current cycle (1) and the number of cycles to failure (N_f), as follows:

$$D \leftarrow D + \frac{1}{N_f}$$

To compute the expected inter-generation time (i.e. Mean Time to Generation, for defect formation):

$$\mu = \beta_\mu - \alpha_\mu \cdot D,$$

where β_μ and α_μ are given parameters, and $\mu > 0$ is initialized at the beginning. Moreover, D is kept below 1 (as the component would otherwise exceed its fatigue limit), and $\mu > 1$ (as it represents a time interval).

- **Defects generation**: each defect is generated after a stochastic amount of time:

$$t_{\text{defect}} \sim \text{Exp}\left(\frac{1}{\mu}\right)$$

The generated defects are assumed to have an elliptical shape, with depth a and length c:

$$c \sim \text{W}(3, 2), \quad a = \frac{c}{5}$$

If c exceeds a certain threshold (2 mm), the flaw becomes a crack and propagates; otherwise, it is classified as a small defect.

- **Defect Propagation**: both newly generated and previously propagated cracks are considered. Crack propagation is evaluated daily, caused by new load cycles. To evaluate the crack size, the Paris law is applied, assuming ΔK lies within the stable propagating region:

$$\frac{da}{dN} = C \cdot (\Delta K)^m$$

where $\frac{da}{dN}$ is the crack growth rate per cycle, ΔK is the stress intensity factor range, and C and m are material-dependent constants.

ΔK_a and ΔK_c are computed using Newman and Raju's formulation, as a function of a, c, and σ^*_{max} (see *algorithm* 1). The loop is concluded when $a > 0.95 \cdot s$, with s the pressure vessel's thickness.

- **Leakage**: a leakage is assumed to occur whenever $a_{lim} = 0.95 \cdot s$ is reached.

Algorithm 1 Crack Propagation Algorithm - Paris Law

Initialize variables a_0, c_0, σ^*_{max}
Set $i \leftarrow 0$
repeat
  $\Delta K_{a,i} = f_a\left(a_i, c_i, \sigma^*_{max}\right)$
  $\Delta K_{c,i} = f_c\left(a_i, c_i, \sigma^*_{max}\right)$
  $a_{i+1} \leftarrow a_i + \left[C \cdot \Delta K^m_{a,i}\right]$
  $c_{i+1} \leftarrow c_i + \left[C \cdot \Delta K^m_{c,i}\right]$
  $i \leftarrow i + 1$
until $a_{i+1} > a_{\text{lim}}$
return "Crack exceeded threshold"

Simulations (with different *seeds*) are then performed, continuously spanning a period of 10 years.

1.1.2 Water plant pressure vessel

The second pressure vessel operates within a water plant and is currently experiencing leakage. A simplified SimScape model is shown in *figure* 2.

Its features are as follows:

- Water is modeled as an isothermal liquid at a temperature of 293.15 K (20 °C). The liquid is supplied by a pump to the pressure vessel.

- The pump acts as a controlled pressure source, maintaining a constant pressure of 1000 Pa. The required liquid flow is defined by the pressure drop of the tank, as the pump must compensate for both required and inefficiency losses.

- The pressure vessel is modeled as a tank with one inlet port (connected to the pump) and two outlet ports, representing pressure drop and leakage. The three ports (A, B, C), shown in *figure* 2, are positioned at different heights: respectively, 0.1 m, 0.05 m, and 0 m. These heights are necessary to calculate the pressure at each level relative to atmospheric pressure. All ports share the same cross-sectional area of 0.01 m^2. Moreover, the initial liquid level in the tank is set to 5 m.

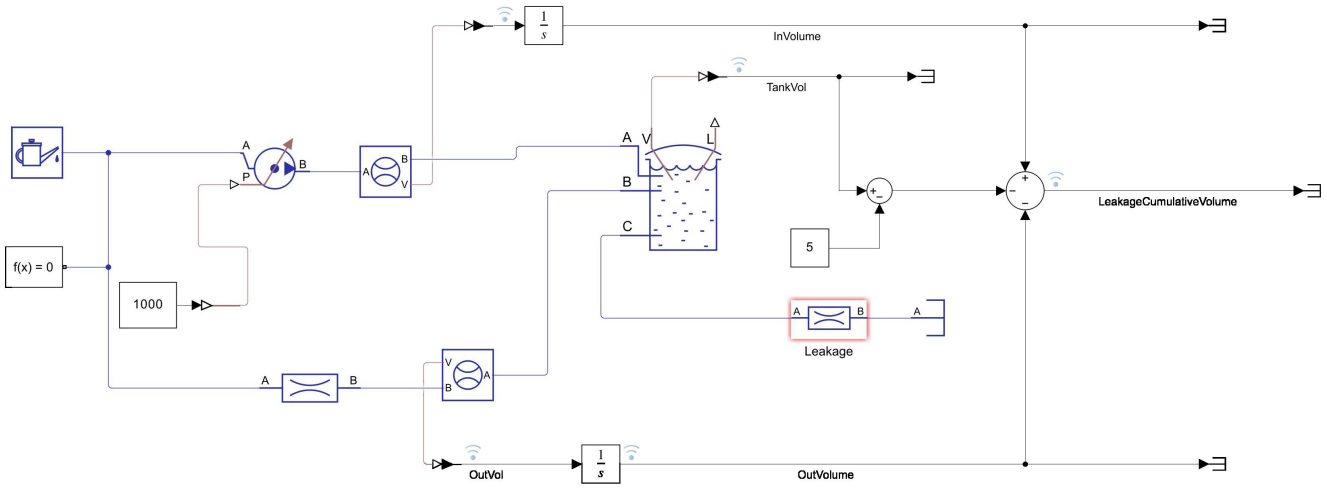


Figure 2: water plant pressure vessel

- The first output port (B) is connected to an orifice that models the pressure drop caused by:

  1. The user (e.g. an industrial boiler)
  2. Distributed losses within the hydraulic circuit

  The orifice area is assumed to be constant ($0.01\ m^2$), representing a scenario with a constant load. As mentioned above, the pressure drop at this port is directly linked to the pump.

- The leakage is modeled as an orifice connected to the third output port (C), which is positioned at the lowest height and thus has the highest pressure. The flow exiting this port is directed to a sink, leading to irreversible fluid losses. To model different leakage diameters, the orifice area is set variably according to:

$$A_{leak} = \frac{\pi \cdot D_{leak}^2}{4}$$

  It should be noted, however, that the leakage could alternatively be modeled by directly modifying the output port cross-sectional area.

The system must be equipped with sensors to monitor the leakage over time. Since it is impossible to measure the leakage size directly, the measurement apparatus shown in *figure* 2 is used.

Two flow rate sensors are employed: one measures the flow rate related to the pump, and the other measures the one associated with pressure drops. These data are then combined with the liquid volume in the tank, which can be easily determined by monitoring the liquid level. The initial liquid volume is 5 m^3, since the cross-sectional area is 1 m^2.

V_{tank} represents the volume of the tank, while q_{in}, q_{out}, and $q_{leakage}$ are the volumetric flow rates corresponding to the pump, the pressure drop, and the leakage (unknown and to be determined). Being water an incompressible fluid, the conservation of flow rate gives the following relationship:

$$\frac{dV_{tank}}{dt} = -q_{out} + q_{in} - q_{leakage}$$

Quantities q_{out} and q_{in} are measured by the two flow rate sensors, but $\frac{dV_{tank}}{dt}$ might be difficult to measure in practice. The solution is to measure $V_{tank}(t)$. By integrating the signals from the aforementioned sensors between 0 and t, and considering that $V_{tank}(0) = 5\ (m^3)$, the leakage volume can be computed as:

$$V_{leakage}(t) = -V_{out}(t) + V_{in}(t) - (V_{tank}(t) - 5)$$

This equation gives the cumulative leakage volume, which increases over time. Simulations are then performed across different leakage diameters, ranging from 10^{-4} to $3 \cdot 10^{-4}\ m$, resulting in leakage areas on the order of 10^{-9} to $3 \cdot 10^{-8}\ m^2$.

1.2 Forced sampling

For the pressurized gas pressure vessel model described in *subsection* 1.1.1, Monte Carlo sampling, including importance sampling, is used to estimate its 10-year expected life cycle cost.

Only the cost related to cracks is considered, without specifying cost units. Further economic evaluations are outside the scope of this analysis. The cost function is defined as follows:

$$C = \begin{cases} 5000 & \text{if leakage,} \\ 0 & \text{if defect does not propagate,} \\ 100 \cdot (a - 0.4) & \text{if defect propagates.} \end{cases}$$

Here, a represents the crack depth (log of simulations, see *figure* 1). Cracks with a depth below the threshold value of 0.4 mm do not propagate further (i.e. they are classified as small defects, having $c < 2\ mm$). Keeping different *seeds*, 100 simulations are performed. The outcomes are categorized as follows:

- **Leakages**: these occur when $a > a_{lim}$, with $a_{lim} = 0.95 \cdot s$. Leakages are accounted for throughout the entire life cycle.
- **Cracks**: defects where $0.4 < a < a_{lim}$, which propagate without leakage. They are evaluated only at the end of the 10-year period, with costs proportional to their size.
- **Defects**: flaws that do not propagate. They are excluded from cost computations as they incur no cost.

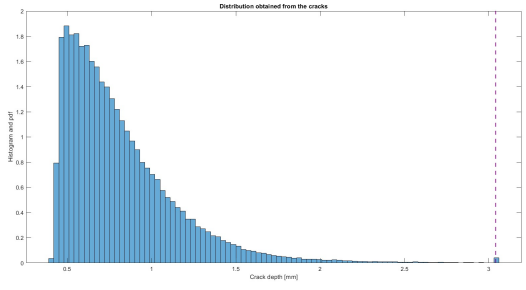


Figure 3: simulations' distribution

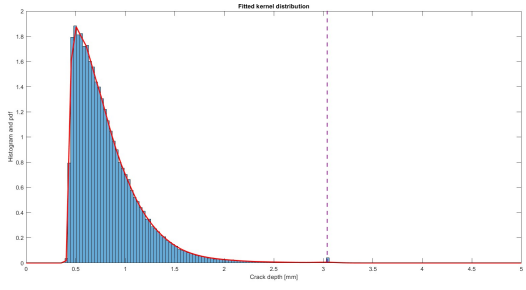


Figure 4: fitted kernel distribution

The cost function heavily depends on the number of leakages (total failures); however, the distribution of cracks is concentrated at lower depths (*figure* 3).

Consequently, the standard Monte Carlo approach can be used for the evaluation of the expected cost, but the result is less reliable than the one obtained with forced sampling.

Moreover, as running several simulations for the cost estimation would require significant computational effort, a workaround can be used. It consists of fitting the probability distribution of the crack depth at the end of the considered period with a kernel distribution (i.e. a way to smooth data points as the sum of kernel functions), and then applying the cost function to samples drawn from this approximate distribution.

However, it is worth remembering that, in practice, one should apply importance sampling to several Simulink simulations.

The workaround is shown in *figure* 4: the histogram is fitted over the support $[0.4, 5]$, although the kernel function is guaranteed to be zero beyond $0.95 \cdot 3.2 = 3.04\ mm$.

Monte Carlo sampling. The expected cost is computed, without forced sampling in this case. Defining the kernel distribution of crack depths as a $\sim p(a)$ and the cost $C(a)$, the expected value will be:

$$\mathbb{E}[C(a)] = \int_{-\infty}^{\infty} C(a)\, p(a)\, da$$

Monte Carlo sampling can be performed through the following steps:

1. Draw a set of N independent samples a_n (with $n = 1, \ldots, N$) from $p(a)$.
2. Compute the estimate of the expected cost (i.e. the mean):

$$\widehat{\mathbb{E}}[C(a)] \approx \frac{1}{N} \sum_{n=1}^{N} C(a_n),$$

   where $C(a_n)$ is the cost associated with each sample a_n.

3. Since the estimator $\widehat{\mathbb{E}}[C(a)]$ is a random variable, its variance can be computed as:

$$\widehat{\mathrm{Var}}(\widehat{\mathbb{E}}[C(a)]) = \frac{1}{N^2} \sum_{n=1}^{N} \Big(C(a_n) - \widehat{\mathbb{E}}[C(a)] \Big)^2 = \frac{\widehat{\mathrm{Var}}(C(a))}{N}$$

   The Monte Carlo (MC) standard error is then the square root of the variance:

$$MC_{se} = \sqrt{\widehat{\mathrm{Var}}(\widehat{\mathbb{E}}[C(a)])}$$

   As a consequence, the uncertainty of the cost estimation decreases as the number of samples N increases.

4. The estimate of the cost variance is analogously given by:

$$\widehat{\mathrm{Var}}(C(a)) = \frac{1}{N} \sum_{n=1}^{N} \Big(C(a_n) - \widehat{\mathbb{E}}[C(a)] \Big)^2.$$

   Similarly to the case of the mean, the MC standard error of the variance estimate can also be computed.

The aforementioned procedure is applied in several cases, and the results are compared in the following paragraph to those obtained using importance sampling.

However, whenever $C(a)$ is high, the corresponding probability distribution $p(a)$ is here extremely low, leading to huge inaccuracies. Thus, importance sampling is almost compulsory.

Forced sampling. Since $C(a)$ is large where $p(a)$ is small, this can result in incorrect estimation of the expected cost, as it is dominated by low-probability regions. This implies that the tail of distribution $p(a)$ - where leakages occur or cracks are larger - is more critical than other regions.

The previous steps are modified as follows:

1. Draw a set of N samples a_n from a *simpler* distribution $q(a)$ (rather than $p(a)$).

2. Compute the estimate of the expected cost as:

$$\widehat{\mathbb{E}}[C(a)] \approx \frac{1}{N} \sum_{n=1}^{N} \frac{p(a_n)}{q(a_n)} \cdot C(a_n),$$

   This is justified by the following:

$$\mathbb{E}[C(a)] = \int_{-\infty}^{\infty} C(a) \, \frac{p(a)}{q(a)} \, q(a) \, da$$

3. Variance of $\widehat{\mathbb{E}}[C(a)]$ becomes:

$$\widehat{\mathrm{Var}}(\widehat{\mathbb{E}}[C(a)]) = \frac{1}{N^2} \sum_{n=1}^{N} \left(C(a_n) \cdot \frac{p(a_n)}{q(a_n)} - \widehat{\mathbb{E}}[C(a)] \right)^2 = \frac{\widehat{\mathrm{Var}}(C(a))}{N}$$

In *figure* 5 q, chosen as a uniform distribution, is compared to p and to the cost function.

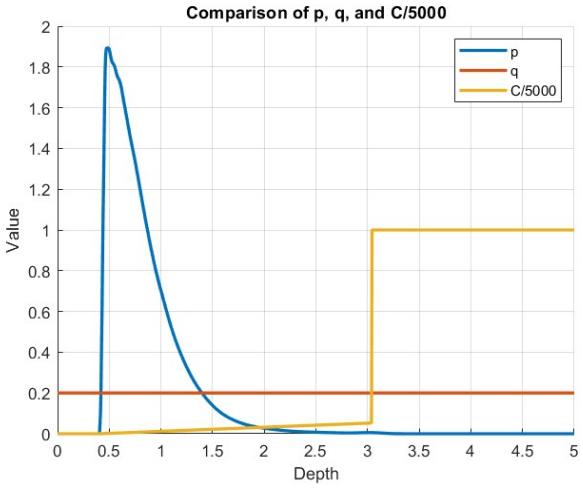


Figure 5: comparison

In this analysis, we compare Monte Carlo sampling with its forced variant across several metrics: expected value (*figure* 6), mean estimation standard error (MC_{se}, *figure* 7), variance (*figure* 8), and variance estimation standard error (MC_{se}, *figure* 9). The analysis is based on 100 computations, leading to the following considerations:

- **Cost expected values:** unforced Monte Carlo sampling leads to a more spread distribution, while importance sampling results in a peak between 75 and 80. Interestingly, these values have a very low frequency in the unforced case.

- **Mean estimation standard error:** higher for the unforced case (up to 100) and considerably lower for the forced one (below 8).
- **Variance estimation:** spread over a higher range of values for the unforced case — in the order of 10^6 — while more concentrated around $4 \cdot 10^4$ for importance sampling. Note that variance is a parameter describing $C(a)$, so it remains the same in both cases.
- **Variance estimation standard error:** in the order of thousands for importance sampling, while significantly higher for the unforced case.

In *figure* 10, the expected cost and the corresponding Monte Carlo standard error are shown for both unforced and forced sampling as the number of samples N varies between 100 and 10,000. In the unforced case, both graphs exhibit more pronounced oscillations. Specifically, MC_{se} decreases as N increases, though it continues to oscillate due to the stochastic nature of the computations. Meanwhile, the expected cost gradually converges to a value similar to that observed with importance sampling. In contrast, the forced case shows significantly smaller oscillations in both the expected cost and MC_{se}, with MC_{se} remaining consistently lower (notice the different scales).

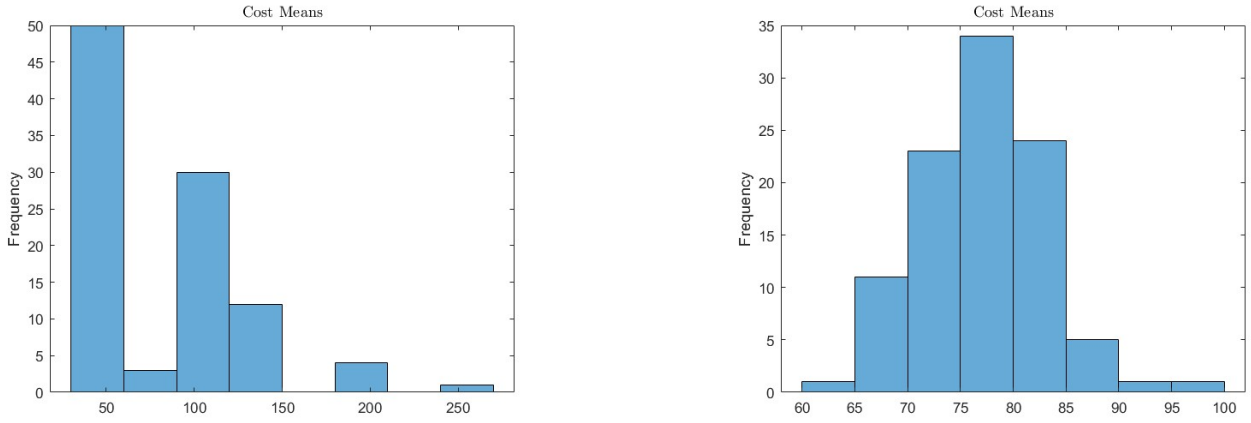


Figure 6: cost means

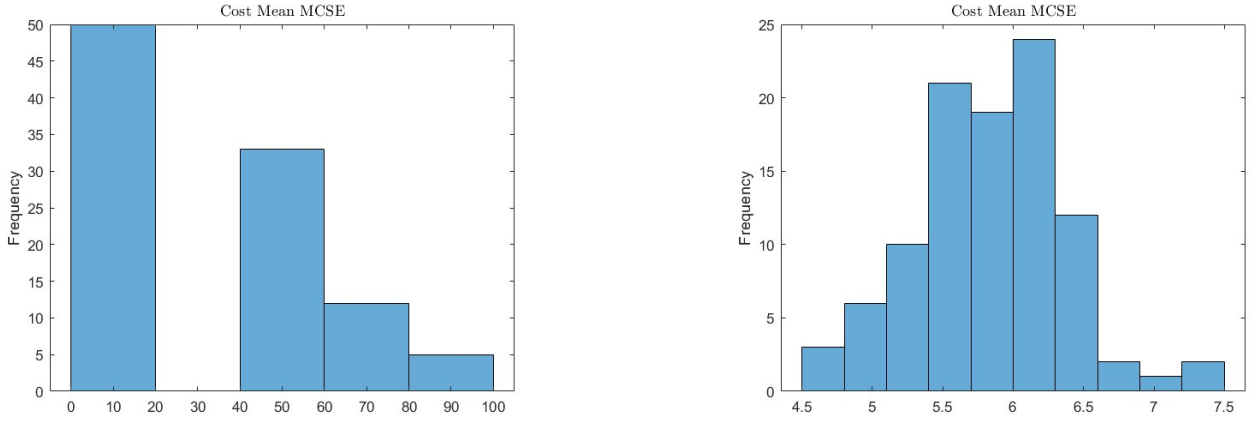


Figure 7: mean estimation MC_{se}

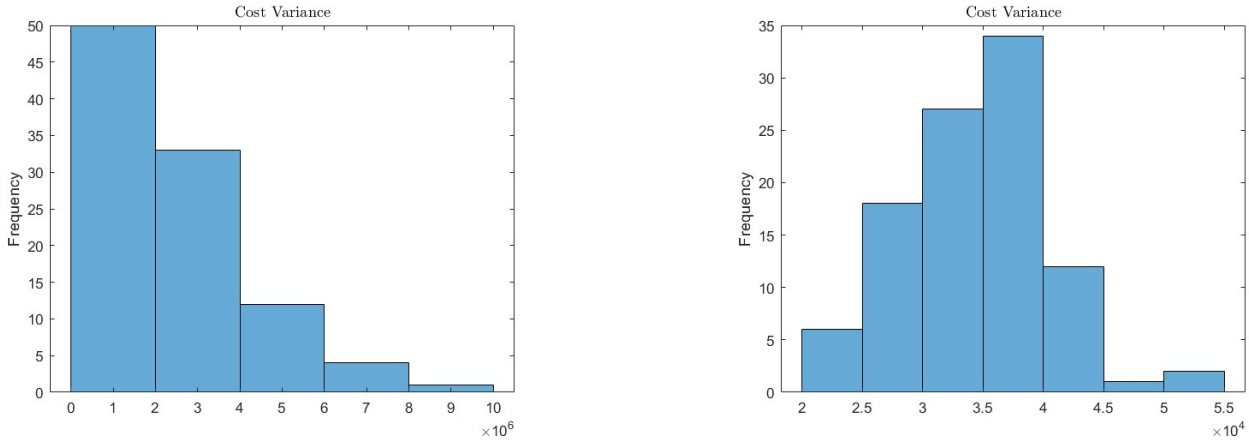


Figure 8: variances

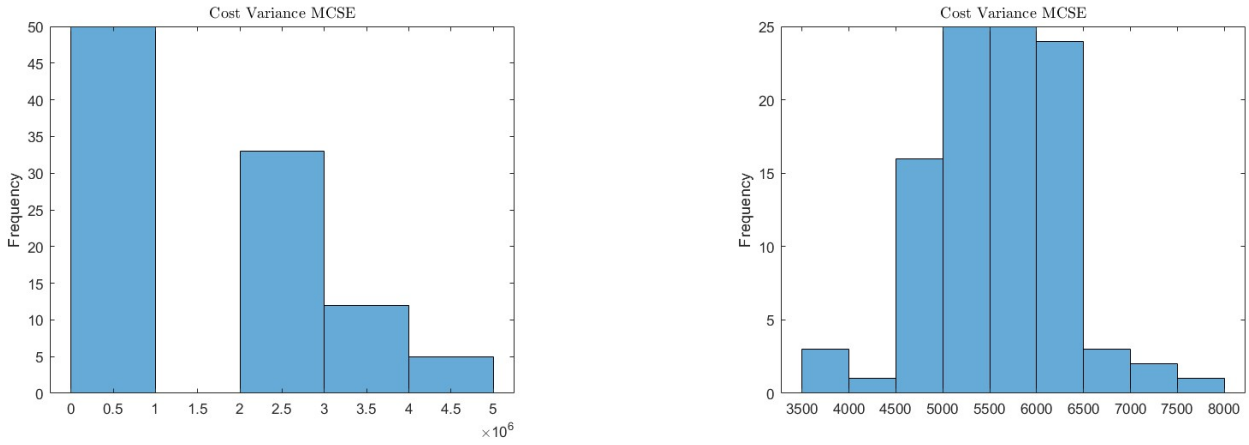


Figure 9: variance estimation MC_{se}

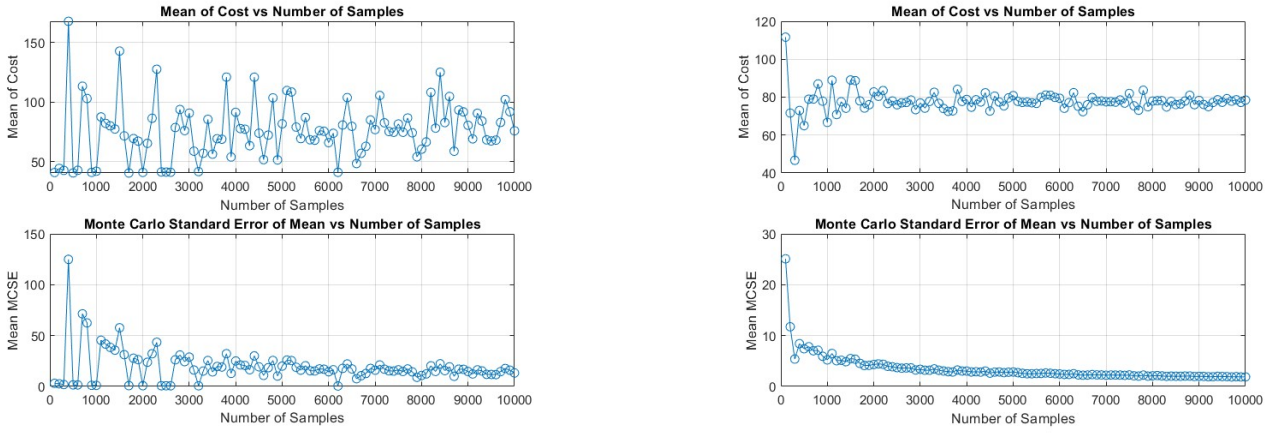


Figure 10: improvement with the number of samples

1.3 Surrogate modeling

Surrogate modeling for the Digital Twin of the pressure vessel described in *subsection* 1.1.2 is carried out by defining an Artificial Neural Network (ANN) regression model, which represents the expected value of the system's measurement.

Multiple simulations are performed to collect sensor data from the model, as various input leakage dimensions (D_{leak}) result in different cumulative leakage volumes $V_{leak}(t)$.

The simulations are performed under the following conditions:

1. D_{leak} ranges from 10^{-4} to $3 \cdot 10^{-4}$ m over 100 simulations, leading to leakage areas on the order of 10^{-9} to $3 \cdot 10^{-8}$ m^2 (assuming circular leakage).

2. Each simulation runs for 1000 s, with a 1 s time step, resulting in 1001 time instants per simulation.

In *figure* 11, the relationship between volume and time is shown, along with the relationship between the leakage volume at the middle and at the end of the simulation versus the leakage area. As expected, both relationships are linear. The cumulative volume increases over time, and the gradient is higher for larger leakage sizes. In fact, as the area increases, the volume increases linearly both at the end and at the middle of the simulation.

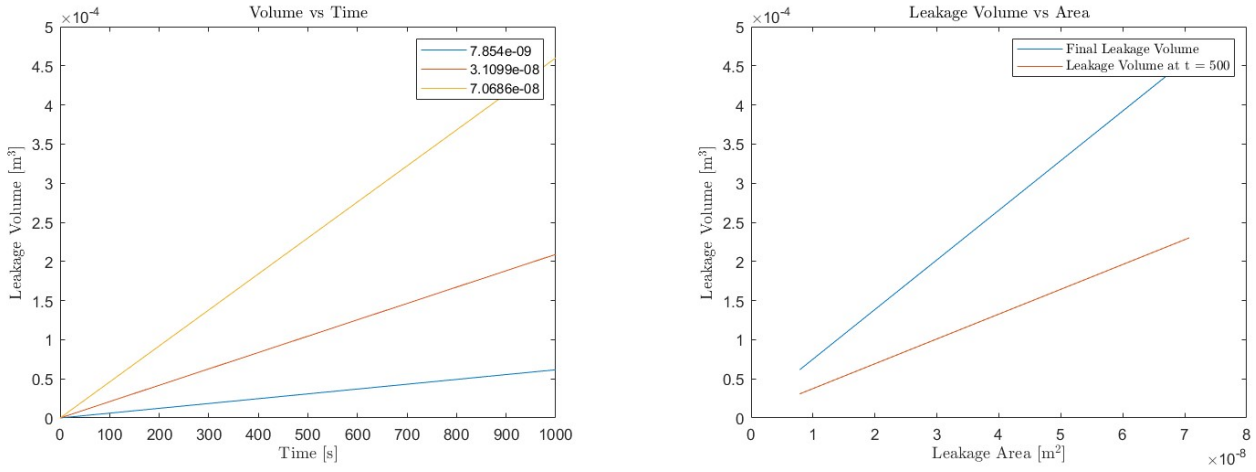


Figure 11: relationship between volume, time, area

The regression must approximate a relatively simple function, which the Neural Network should reconstruct as follows:

$$V_{leak} = f(t, A_{leak})$$

The training dataset consists of a matrix (X) containing two features: the simulation time, ranging from 0 to 1000 for each simulation, and the leakage size. The target values are represented by a vector (y), where each entry corresponds to the leakage volume associated with a specific simulation at a given time. Dataset is shown in *figure* 12.

This implies that $X \in \mathbb{R}^{(1001 \cdot 10) \times 2}$, while $y \in \mathbb{R}^{1001 \cdot 10}$. Since the function is linear, the training is expected to converge quickly. However, to further enhance the training performance and stability, the data are normalized, and the network is trained using these transformed inputs and targets.

$$\bar{X} = \text{mean}(X), \quad \sigma_X = \text{std}(X), \quad X_{\text{norm}} = \frac{X - \bar{X}}{\sigma_X}$$

$$\bar{y} = \text{mean}(y), \quad \sigma_y = \text{std}(y), \quad y_{\text{norm}} = \frac{y - \bar{y}}{\sigma_y}$$

This naturally implies that, when using this surrogate model to perform inference, the output must be denormalized using the parameters computed during the preprocessing phase:

$$y = \bar{y} + \sigma_y \cdot \hat{y}_{\text{norm}}$$

$$X = \begin{bmatrix} 0 & A_{leak,sim1} \\ 1 & A_{leak,sim1} \\ 2 & A_{leak,sim1} \\ \vdots & \vdots \\ 0 & A_{leak,simj} \\ 1 & A_{leak,simj} \\ \vdots & \vdots \end{bmatrix}, \quad y = \begin{bmatrix} V_{leak,sim1}(0) \\ V_{leak,sim1}(1) \\ V_{leak,sim1}(2) \\ \vdots \\ V_{leak,simj}(0) \\ V_{leak,simj}(1) \\ \vdots \end{bmatrix}$$

Figure 12: data

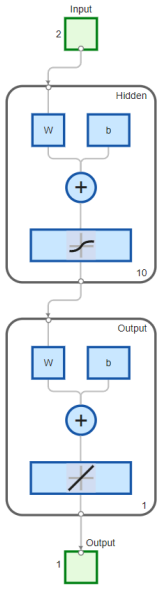


Figure 13: network architecture

The chosen network architecture is shown in *figure* 13. It is a feedforward neural network (FFNN) with two input neurons (time and leakage area), and one output neuron which regresses the leakage volume. A single hidden layer with ten neurons, using S-shaped activation functions, is fully connected to the output layer, which has a linear activation function. The network therefore has very few parameters ($2 \cdot 10 + 10 + 10 \cdot 1 + 1 = 41$), which enables a low training time per epoch.

Moreover, the dataset is split into *train* (70%), *validation* (15%), and *test* (15%) subsets.

To train the network, backpropagation employs the Mean Squared Error (MSE) function:

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

where y_i represents the true values, $\hat{y}_i$ represents the predicted values, and N is the total number of samples (the whole training set for MatLab, since it uses *batch training*).

Its trend with respect to the training epochs is shown in *figure* 14, together with other parameters representing the training behavior. The MSE decreases rapidly over the course of 1000 epochs, converging to a very low error due to the simplicity of $f(t, A_{leak})$. For the same reason, there is almost no difference between training, validation, and test MSE.

The second plot shows the gradient, used to search for the global minimum of the error. The gradient is not monotonic, indicating that some local minima are encountered before reaching final convergence.

The parameter μ is related to the Levenberg-Marquardt training algorithm, while validation fails represent instances where the Mean Squared Error on the validation dataset increases compared to the previous epoch (almost zero everywhere in this case).

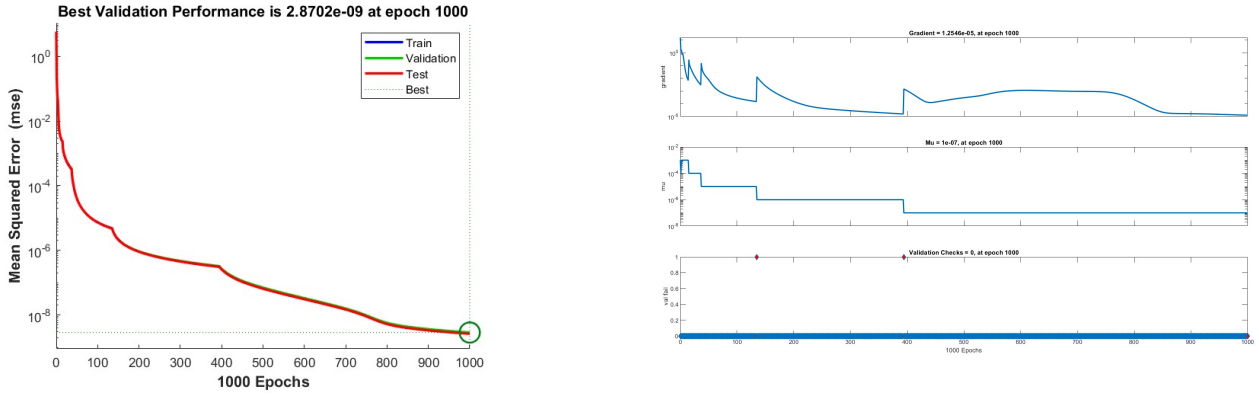


Figure 14: MSE, gradient, μ, and validation checks

In *figure* 15, the errors (targets − outputs) are plotted as a histogram (presenting very low values), along with the regression performance represented by the R^2 parameter. Since R^2 equals 1 for all cases, this indicates outstanding performance.

However, the results suggest that an Artificial Neural Network might not even be necessary in this case. This observation aligns with the fact that the simulation data are noise-free and completely deterministic.

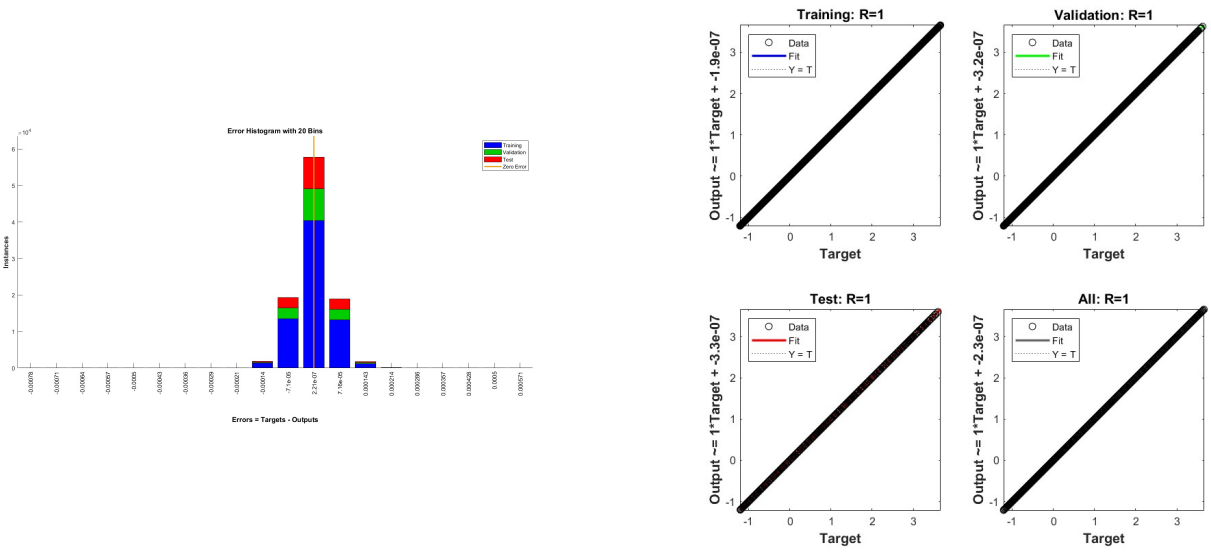


Figure 15: error distribution and performances

Moreover, an additional comparison has been performed between the previous network and one with just two hidden neurons (*figure* 16). While the MSE remains very small, and the values

for training, validation, and testing are close, it is 100,000 times larger than that of the previous network. Additionally, learning stagnates after just a few epochs. However, the R^2 value remains very close to 1: this model still performs well.

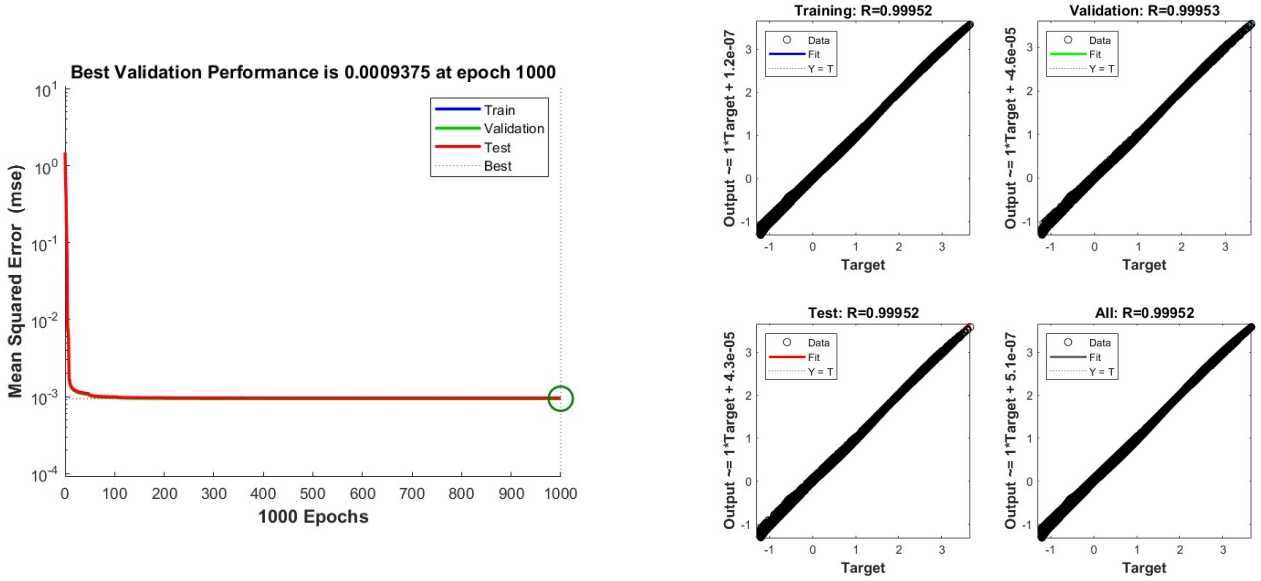


Figure 16: ANN's performances with just 2 hidden neurons

1.4 Model updating

Using the Artificial Neural Network surrogate model of the pressure vessel's Digital Twin (*section* 1.3), the Markov Chain Monte Carlo algorithm is used to update the model parameter (leakage area) based on the corresponding measurement (leakage cumulative volume). For the sake of simplicity, $A \leftarrow A_{leak}$ and $V \leftarrow V_{leak}$ from now on.

Measurement noise. Experimental data are generated by running a simulation with $D_{leak} = 1.5 \cdot 10^{-4}\ m$ — a value in the diameter interval used to train the surrogate model. This corresponds to a *true* (imposed) leakage area of $A = 1.77 \cdot 10^{-8}\,\mathrm{m}^2$. The resulting leakage cumulative volume V, as a function of time, is shown in *figure* 17.

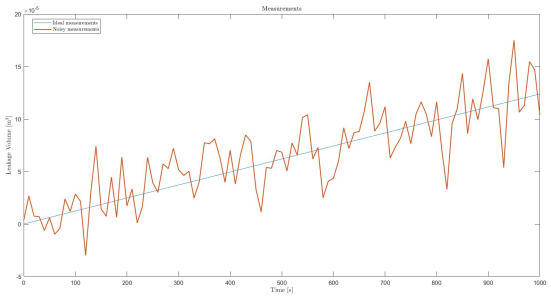


Figure 17: measurements

Real sensors would not ideally measure V, as some measurement noise is always present. The measurement is thus assumed to follow a Gaussian distribution, with the simulated leakage volume from the Simulink model as its expected value and a standard deviation of $\sigma_\eta = 2.5 \cdot 10^{-5}$ m:

$$V(t) \sim \mathcal{N}(V_{sim}(t), \sigma_\eta^2)$$

Measurement samples can be assumed to be independent. Notice that, in this Bayesian approach (utilized by the Markov algorithm), the leakage area is the parameter to estimate, given the measurement outcomes and the corresponding time steps. Thus, the likelihood function becomes the product of independent Gaussian distributions:

$$\mathcal{L}(\mathbf{V} \mid A_k) = p(\mathbf{V} \mid A_k) = \prod_{t=0}^{T} \frac{1}{\sqrt{2\pi\sigma_\eta^2}} e^{-\frac{[V(t)-V_{ANN}(t,A_k)]^2}{2\sigma_\eta^2}},$$

where:

- $V_{ANN}(t, A_k)$ approximates the simulation output for a given A_k.
- A_k denotes the state at iteration k.
- $\mathbf{V}$ is the measurement vector, in which each element is a time-sample from the measurements.

Since the likelihood is a product of values between 0 and 1, being $T = 1000$ this could lead to numerical precision issues. Hence, it is advisable to work with the log-likelihood function:

$$\log \mathcal{L}(\mathbf{V} \mid A_k) = \sum_{t=0}^{T} \left\{ \log \frac{1}{\sqrt{2\pi\sigma_\eta^2}} - \frac{[V(t) - V_{ANN}(t, A_k)]^2}{2\sigma_\eta^2} \right\}$$

MCMC method. The Monte Carlo Markov Chain (MCMC) algorithm is used to sample from a probability distribution when direct sampling is challenging. By combining Bayesian inference with Monte Carlo sampling, it iteratively converges to the posterior probability distribution $p(A \mid \mathbf{V})$. The corresponding pseudocode, adapted to the specific case, is shown in *algorithm* 2.

Algorithm 2 Updating Leakage Size - Monte Carlo Markov Chain

1. Initialization:
   - Select prior pdf $p(A_k)$ s.t. $A_k \cdot 10^9 \sim \mathrm{W}(40, 1.5)$
   - Select proposal pdf $q(A_{k+1} \mid A_k) \sim \mathcal{N}(A_k, \sigma_{prop}^2)$
   - Select starting point A_1 for the Markov Chain and compute $p(A_1)$, $\mathcal{L}(\mathbf{V} \mid A_1)$

repeat

2. Sample $\hat{A}_{k+1}$ from proposal $q(A_{k+1} \mid A_k)$
3. Compute $\log \alpha(\hat{A}_{k+1} \mid A_k) \leftarrow \log p(\hat{A}_{k+1})$, $\log \mathcal{L}(\mathbf{V} \mid \hat{A}_{k+1})$
4. Set $A_{k+1} \leftarrow \hat{A}_{k+1}$ in the chain with probability α, otherwise set $A_{k+1} = A_k$

until desired L_{chain} is reached

5. Erase the burn-in period and proceed with thinning

return $\hat{p}(A \mid \mathbf{V})$

Prior knowledge $p(A_k)$ is that the size is positive, and it is modeled as a Weibull distribution on $A_k \cdot 10^9$, with parameters chosen to make it unlikely for the leakage diameter to exceed 0.4 mm (see *figure* 18).

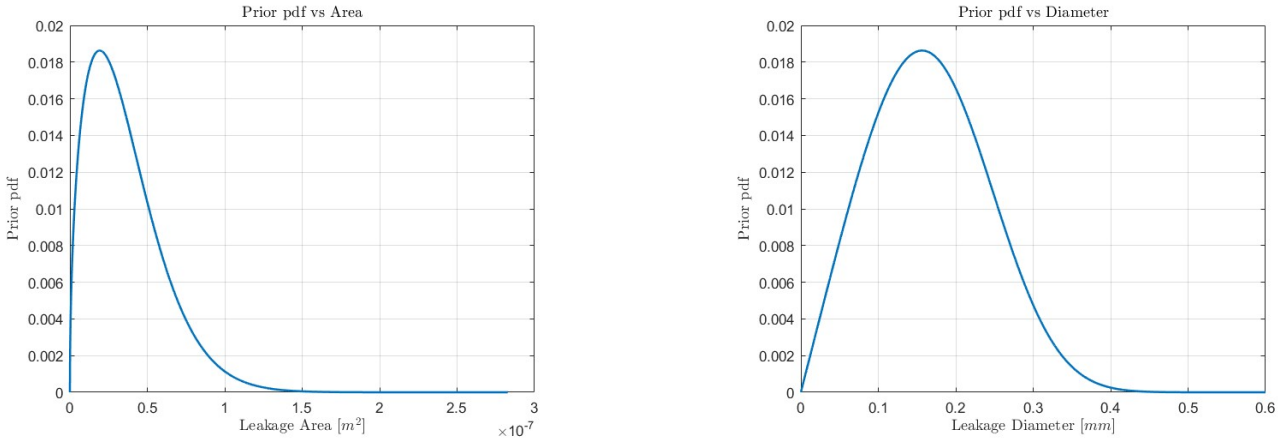


Figure 18: prior $p(A_k)$, $p(D_k)$

Proposal density $q(A_{k+1} \mid A_k)$ is a user-defined normal distribution with mean A_k and a specified proposal standard deviation σ_{prop}. Initially, σ_{prop} is set to $3 \cdot 10^{-9}$.

The starting point A_1 is chosen arbitrarily, and initially D_1 is set to 2 mm. Chain length L_{chain} is instead set to 5000.

Some considerations:

- *ANN output*: simulation time and $\hat{A}_{k+1}$ are normalized with respect to the parameters $\bar{X}$ and σ_X, as computed in *section* 1.3. For the outputs $V_{ANN}(t, \hat{A}_{k+1})$, the inverse linear transformation is applied using $\bar{y}$ and σ_y. The result is then used to evaluate the log-likelihood.

- α *computation*: acceptance ratio, i.e. the probability to accept $\hat{A}_{k+1} \mid A_k$. Computed as:

$$\alpha = \min\left(\frac{p(\hat{A}_{k+1}) \cdot \mathcal{L}(\mathbf{V} \mid \hat{A}_{k+1})}{p(A_k) \cdot \mathcal{L}\left(\mathbf{V} \mid A_k\right)},\ 1\right)$$

  The corresponding logarithmic form is:

$$\log \alpha = \min\Big(\log \mathcal{L}(\mathbf{V} \mid \hat{A}_{k+1}) - \log \mathcal{L}\left(\mathbf{V} \mid A_k\right) + \log p(\hat{A}_{k+1}) - \log p(A_k),\ 0\Big)$$

  The acceptance ratio increases as the values of $\hat{A}_{k+1}$ move towards regions with higher prior probability than the previous A_k, or higher likelihood. This indicates that the chain is progressing towards regions of higher posterior probability. Ideally, an acceptance ratio of $\alpha \approx 0.3$ should be targeted.

- *Accept* $\hat{A}_{k+1}$ *with* α *probability*: this means sampling from a $\mathbf{Be}(\alpha)$ distribution. It is equivalent to sampling $r \sim \mathcal{U}(0, 1)$ and comparing it to α (or, equivalently, their logarithms). If $r < \alpha$, then $\hat{A}_{k+1}$ is accepted; otherwise, it is rejected.

The Markov Chain obtained with the previously mentioned parameters is shown in *figure* 19, along with the corresponding acceptance ratios, which are evaluated as their means every 50 samples to avoid overcrowding the plots. The proposal variance has been chosen *ad hoc* to achieve optimal acceptance ratios. The Chain slightly overestimates the true leakage area; however, the error is negligible.

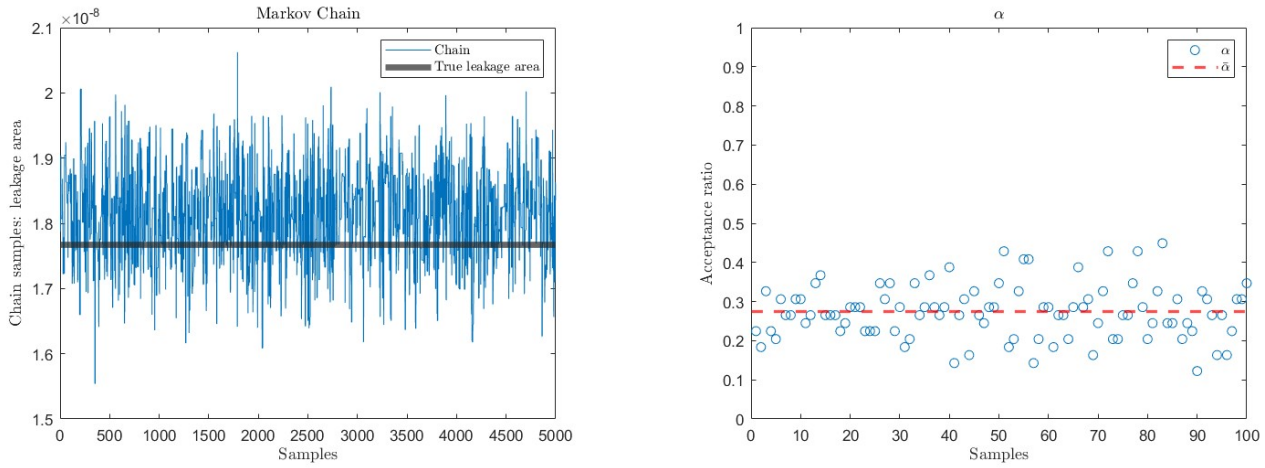


Figure 19: Markov Chain and acceptance ratios

Thinning and posterior distribution. Thinning is performed by retaining one sample every 10 to reduce autocorrelation, i.e. dependencies among consecutive samples. The burn-in period (transient phase, corresponding to the first 100 samples) is discarded to ensure the analysis focuses on the stabilized results. The posterior distribution is then estimated from the resulting A_k, and the outcome is shown in *figure* 20.

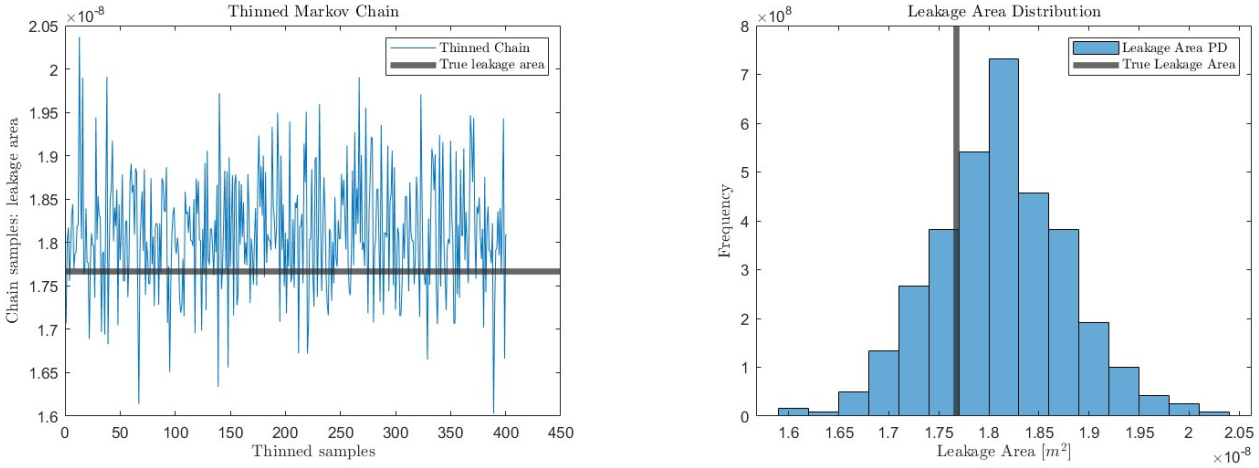


Figure 20: Thinned Markov Chain and posterior distribution $\hat{p}(A \mid \mathbf{V})$

The expected value of the distribution slightly overestimates the true value, as mentioned earlier, but the estimation remains accurate. The evaluated mean area is $1.82 \cdot 10^{-8}\ m^2$ (true one is $1.7 \cdot 10^{-8}\ m^2$), with an estimated standard deviation of $6.42 \cdot 10^{-10}\ m^2$ (approximately 10 times lower than the proposal variance).

Proposal variance and initialization. It should be noted that the proposal pdf q is user-defined: even if results would not depend on its selection if an infinite number of samples were drawn, in practice the *variance* of the proposal is a crucial aspect for obtaining an accurate $\hat{p}(A \mid \mathbf{V})$. The following plots (*figures* 21, 22, 23) show Markov Chains, acceptance ratios, and estimated posteriors for different-magnitudes variances. Finally, by varying D_1 an instability phenomenon of the Markov Chain is shown in *figure* 24.

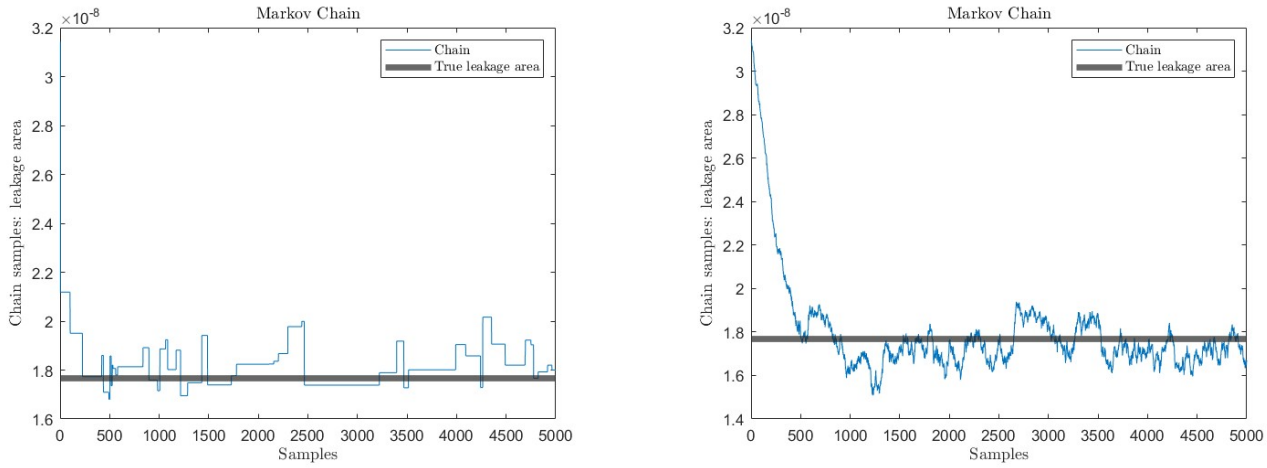


Figure 21: Markov Chains, variances: 10^{-7}, 10^{-10}

In the left plot, the proposal variance is too large: many samples are rejected before a new one is accepted. This occurs because the algorithm samples from a very wide distribution, often moving far from the center of mass of $\hat{p}(A \mid \mathbf{V})$. The autocorrelation between consecutive samples is too low.

In the right plot, the proposal variance is instead too small: almost all samples are accepted, as the algorithm samples from a narrow distribution. In this case, the autocorrelation between consecutive samples is too high, making it difficult to explore regions far from the current state.

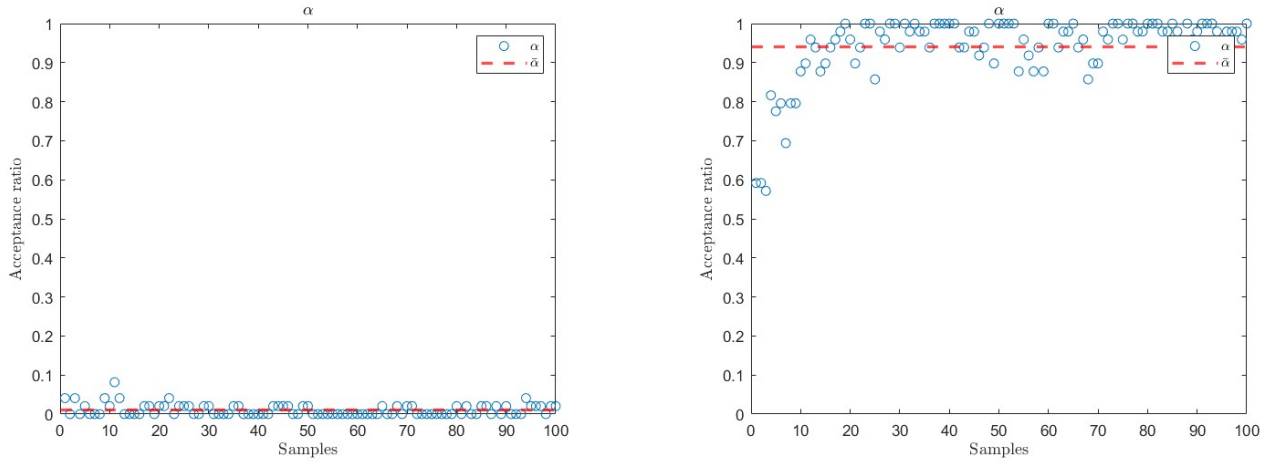


Figure 22: Acceptance ratios, variances: 10^{-7}, 10^{-10}

When the proposal variance is too large, the acceptance ratio drops; on the other hand, when it is too small, α approaches 1.

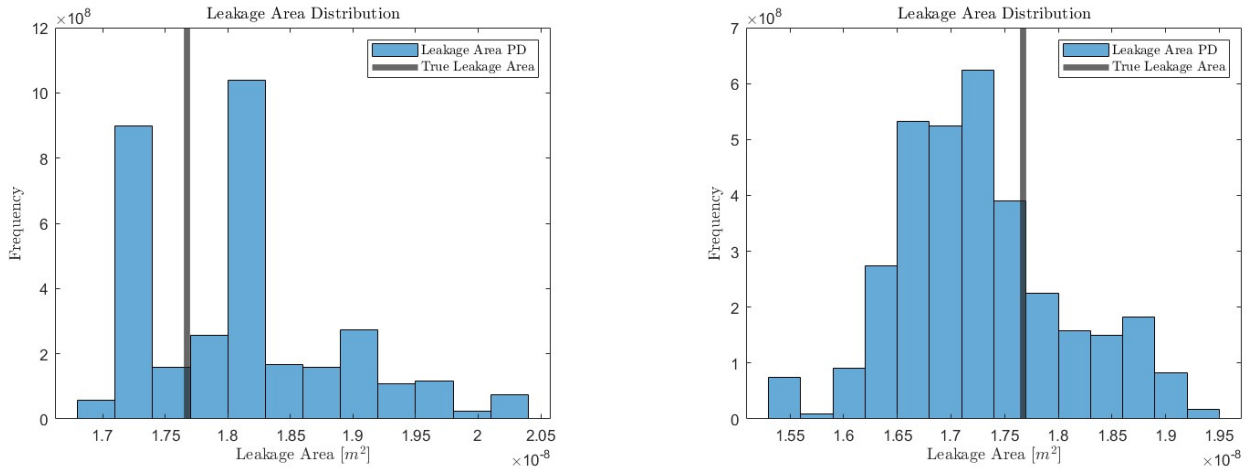


Figure 23: Posterior distributions, variances: 10^{-7}, 10^{-10}

The distributions reflect the behavior of the (thinned) Markov Chains. The first, oscillating between distant values without stabilizing, exhibits two peaks. The second, which rejects only a few samples, is wider than the reference estimate of the posterior distribution, resulting in a higher variance.

Summary statistics:

- Higher variance: mean $1.81 \cdot 10^{-8}$, standard deviation $7.52 \cdot 10^{-10}$.
- Smaller variance: mean $1.73 \cdot 10^{-8}$, standard deviation $7.82 \cdot 10^{-10}$.

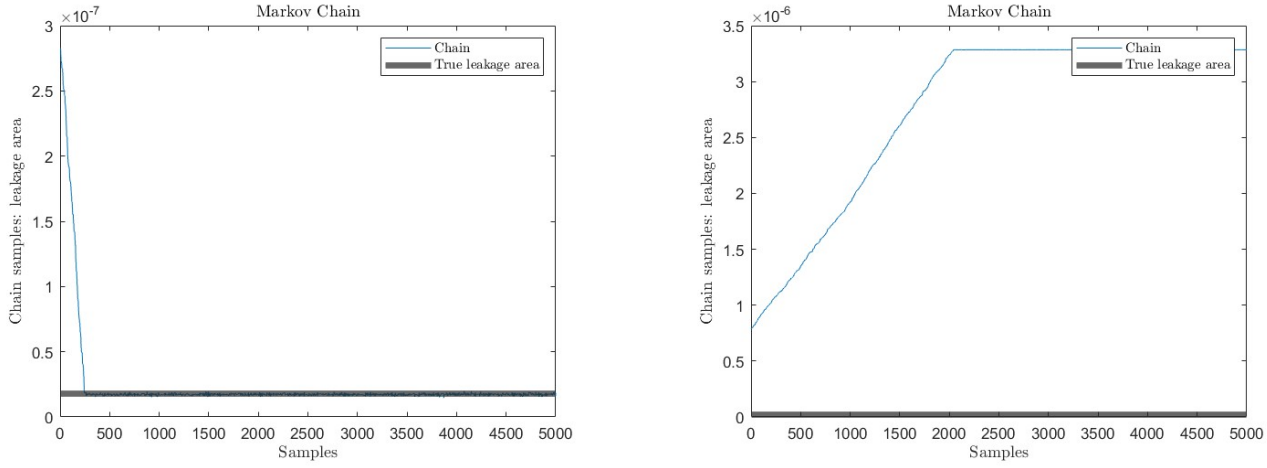


Figure 24: Markov Chains, $D_1 = 6 \cdot 10^{-4}$ *and* $D_1 = 10^{-3}$ *(*$\sigma_{prop} = 3 \cdot 10^{-9}$*)*

In both cases (*figure* 24), the diameter lies far outside the training data of the surrogate model and within a region where the value of the Weibull prior distribution is considerably small. The reason for instability is twofold:

1. **Surrogate model not trained for** $\boldsymbol{D_1 > 0.3\,mm}$: the network produces completely unreliable outputs, resulting in incorrect likelihoods.

2. **Extremely small prior**: the Weibull distribution approaches zero in these cases (see *figure* 18), with its logarithm tending towards $-\infty$. Consequently, beyond a certain leakage diameter, the prior fails to balance the mistakes of the ANN, causing the chain to stabilize at incorrect values. Interestingly, increasing the number of samples or changing the proposal variances has no effect here.

   For $D_1 = 0.6\ mm$, the prior is strong enough to guide the network to a region where its predictions are accurate. For $D_1 = 1\ mm$, the prior becomes so *weak* that it cannot counterbalance the network's errors.

The plots in *figure* 25 illustrate this phenomenon: for $D_1 = 1\ mm$, the cumulative leakage volume absurdly decreases, and for $D_1 = 0.5\ mm$ a non-monotonic behavior is observed, indicating a loss of credibility in the network's predictions.

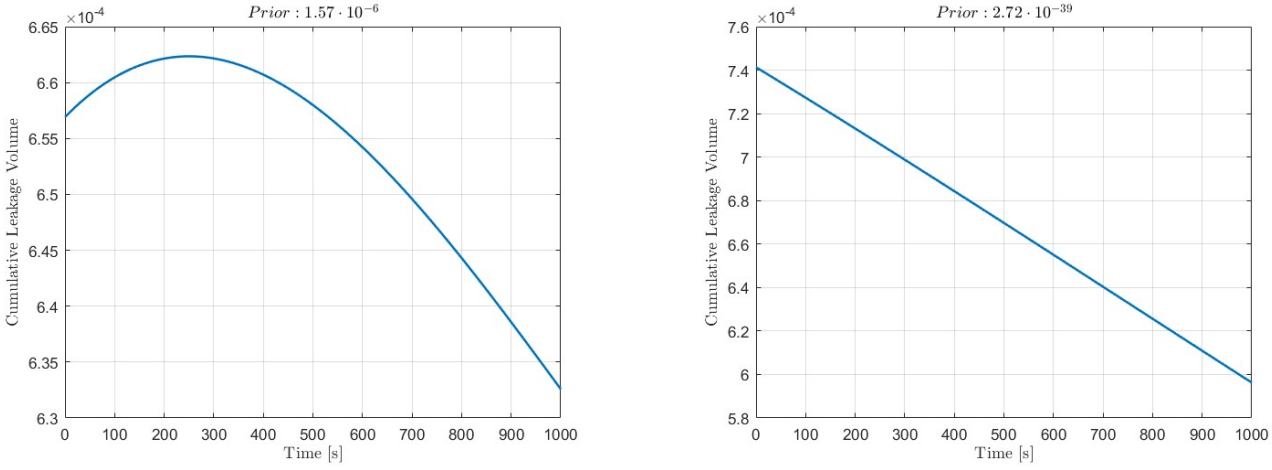


Figure 25: ANN outputs, $D_1 = 5 \cdot 10^{-4}$ *and* $D_1 = 10^{-3}$

It is noticeable how the prior changes, from $1.57 \cdot 10^{-6}$ to $2.72 \cdot 10^{-39}$, becoming dominated by the (incorrect) surrogate model output.

2 Battery Degradation

The Digital Twin of a battery, which degrades over time, has been developed along with the corresponding surrogate Artificial Neural Network. Subsequently, the Residual Useful Life is estimated through MCMC-MH and Particle Filter algorithms.

2.1 Modeling and surrogate modeling activity

Modeling. The battery model in *figure* 26 describes its macroscopic behavior, without accounting for electrochemical equations. Represented as a whole by the battery contained in the Simulink model in *figure* 27, it is made of:

- **Fundamental Battery Model**: a voltage source with finite battery charge capacity. This means that the voltage depends on the State of Charge (SOC), defined as the ratio of the current charge to the rated battery capacity:

$$SOC = \frac{AH}{AH_{max}}$$

  An approximate relationship between the voltage and the remaining charge is:

$$V = V_{nom} \cdot \frac{SOC}{1 - \beta \cdot (1 - SOC)},$$

  where V_{nom} is the nominal voltage, when the battery is fully charged at no load ($V_{nom} = 4.2\ V$ in this case). The constant β is calculated such that at V_1 (the open-circuit voltage, imposed and lower than the nominal one), the charge is AH_1.

- $\mathbf{R_{SD}}$: self-discharge resistance, modeling the phenomenon by which some charge dissipation occurs even when the voltage terminals are disconnected (i.e. the battery partially loses its charge due to internal electrochemical processes): neglected here.

- $\mathbf{R_0}$: internal resistance, models dissipation during usage.

- **Charge Dynamics**: an RC circuit modeling the behavior during charging, represented in Simulink by a polarization resistance R_1 and a time constant T_1.

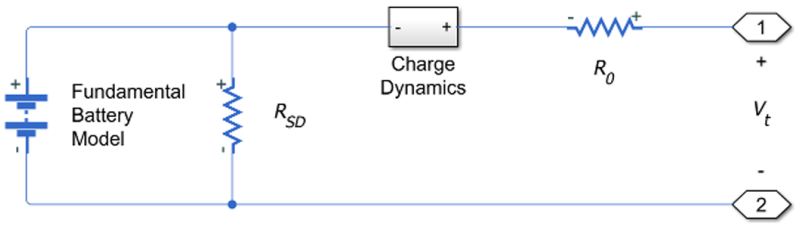


Figure 26: battery model

Battery degradation is generally caused by both calendar aging and fade. The former refers to long-term performance deterioration when the battery is not in use and is neglected in this model. The latter represents the degradation due to cycling and depends on SOC.

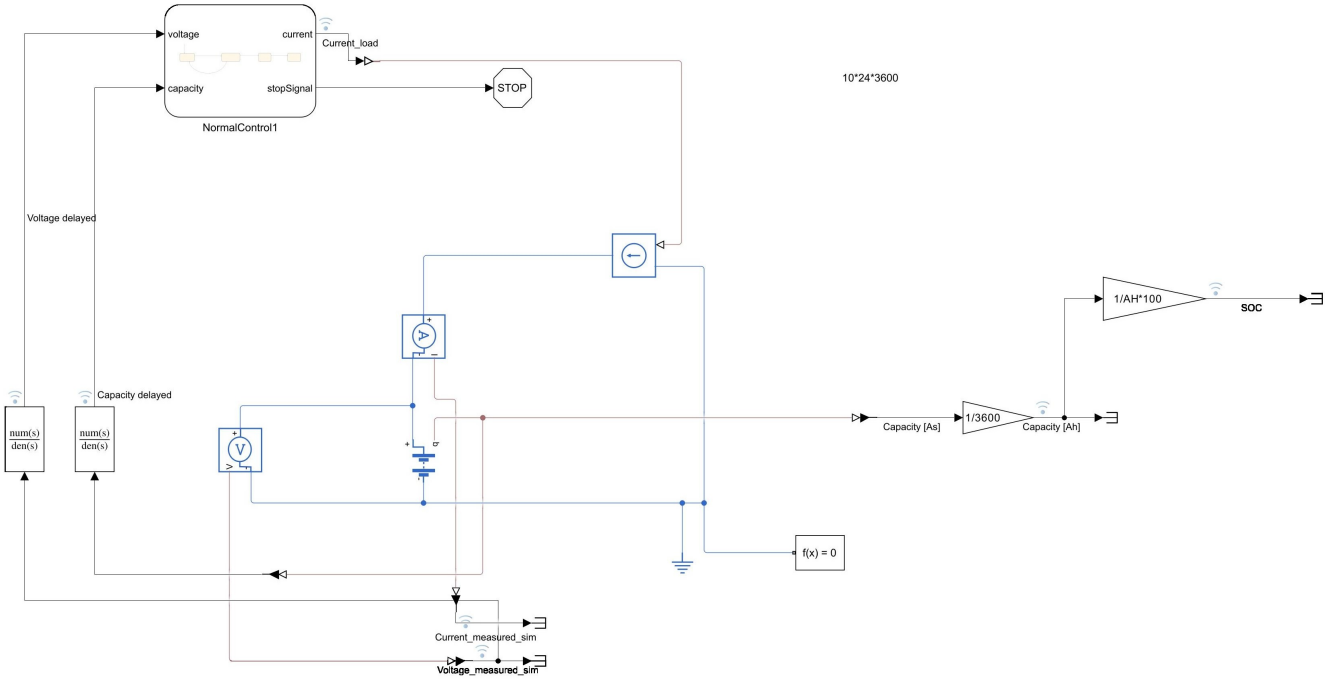


Figure 27: battery degradation

Some additional hypotheses are considered: beyond not accounting for calendar aging, the influence of temperature on all parameters is omitted, as well as the dependence on SOC of R_0, R_1, T_1, and $\eta_{discharge}$ (introduced later).

In *figure* 27, the other components of the Digital Twin are shown. Beyond the battery, a controlled current source maintains the load defined by the control system, detailed in *figure* 28.

This represents a general charge-discharge process: a 2 A current is applied during the charging phase and maintained until AH exceeds 1.75 Ah. Once this condition is met, the discharge phase imposes a current of -2 A until the voltage drops to 2.7 V, at which point a new cycle begins. These battery cycles are ideally repeated up to 50 times, when a resting phase would begin, followed by the termination of the simulation. However, it should be noted that, for a simulation time of 6000 s (as set in the model), the system undergoes only one complete charge-discharge cycle, plus one partial charge.

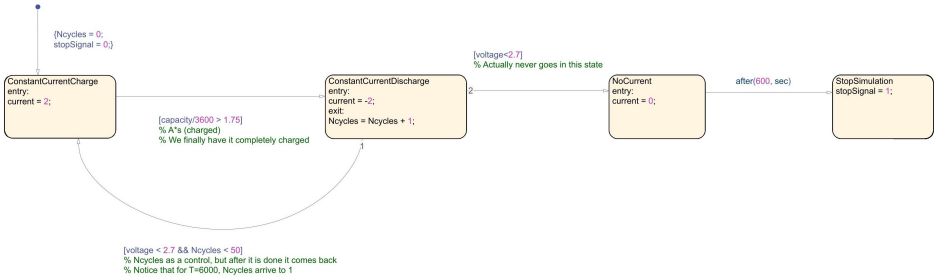


Figure 28: charge and discharge

Finally, a voltmeter and an ammeter are placed in parallel and in series with the battery to measure the voltage and current, respectively.

The fade model scales the following parameters as a function of the number of cycles N:

$$R_0^{(N \text{ cycles})} = R_0^{(0 \text{ cycles})}(1 + k_2\sqrt{N})$$
$$V_1^{(N \text{ cycles})} = V_1^{(0 \text{ cycles})}(1 - k_3 N)$$
$$AH^{(N \text{ cycles})} = AH^{(0 \text{ cycles})}(1 - k_1\sqrt{N})$$

k_i are the degradation parameters computed according to the parameters η_{R_0}, η_{V_1}, and $\eta_{discharge}$: they represent the ratio between the related variables at $N = 100$ cycles and at the beginning ($N = 0$ cycles). In particular:

$$\eta_{discharge} = \frac{AH^{(100 \text{ cycles})}}{AH^{(0 \text{ cycles})}} \quad \Longrightarrow \quad k_1 = \frac{1 - \eta_{discharge}}{10}$$

It should be observed that, while $\eta_{discharge}$ and η_{V_1} are less than one, degradation causes an increase in η_{R_0}, indicating higher internal resistance.

The data obtained from the simulated test will have $\eta_{discharge} = 0.9$ as the true parameter to be inferred from the measurements. Additionally, the voltage measures acquired by the voltmeter are affected by noise and are distributed according to a normal distribution:

$$V_{meas}(t) \sim \mathcal{N}(V_{sim}(t), \sigma^2),$$

where $\sigma = 0.05$ V.

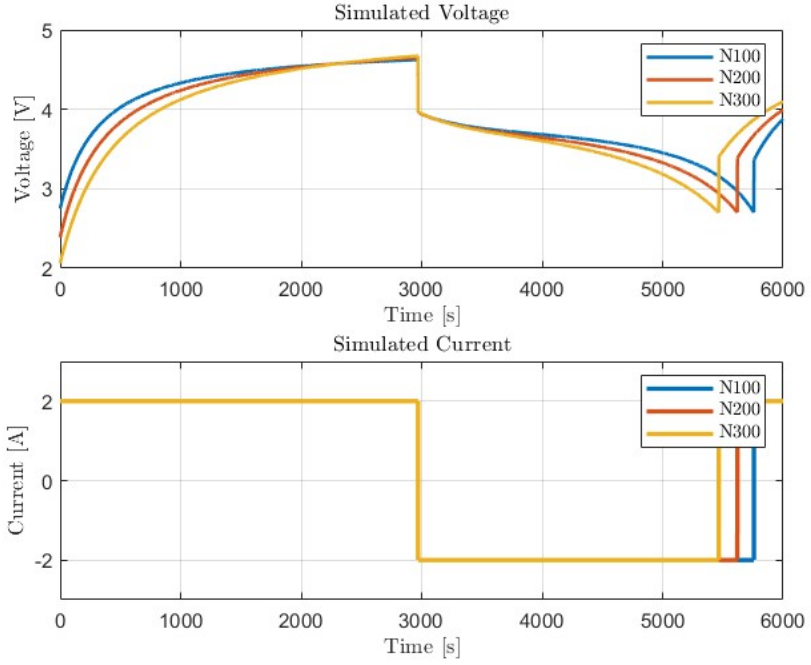


Figure 29: simulated voltage and current for different cycles

In *figure* 29, the simulated voltage and current are plotted with respect to time. The duration of a single cycle increases when the battery is younger. Additionally, at the beginning, the battery cumulates charge under a constant current, always reaching the same capacity 1.75 Ah: all cases require the same charging time.

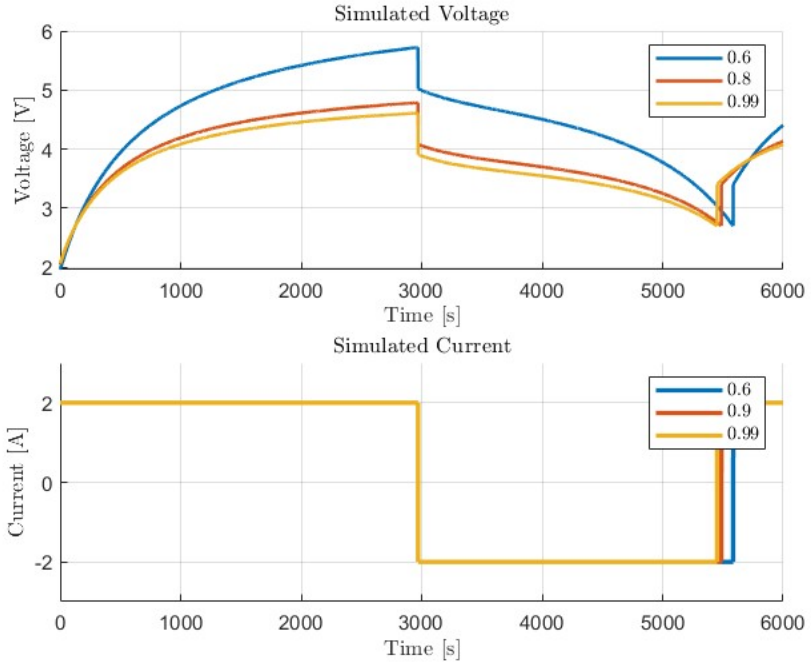


Figure 30: simulated voltage and current for different $\eta_{discharge}$ (at $N = 300$)

In *figure* 30, different values of $\eta_{discharge}$ are analyzed. Lower values result in higher voltage when the target capacity (1.75 Ah) is reached. A healthier battery (with higher $\eta_{discharge}$) completes a cycle in less time.

In *figure* 31 the voltmeter's output, affected by measurement noise, is shown.

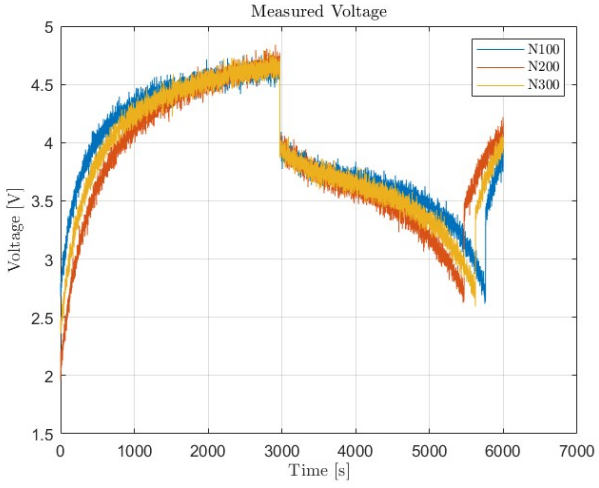


Figure 31: measured voltage

Surrogate modeling. Surrogate modeling for battery degradation is carried out using a regression model. This approach saves time by approximating the simulation through an Artificial Neural Network (ANN). A dataset of simulations with $\eta_{discharge}$ spanning from 0.8 to 0.99, at $N = 100$, 200, and 300 is generated.

The ANN models the function:

$$V = f(t, \eta_{discharge})$$

Since the dataset comprises one charge-discharge cycle plus one partial charge, it is discontinuous; thus, a single ANN cannot capture such discontinuities. The regression is therefore split into three different regions, with separate networks handling each region (charge, discharge, partial charge).

However, as already noted in *figures* 29, 30, and 31, the three regions have different lengths. Some time intervals are thus discarded, selecting only the regions where every battery undergoes charging and discharging.

The networks' architecture is similar to the one discussed in *section* 1.3, and training is performed in the same way. The only difference is the chosen number of hidden neurons (5).

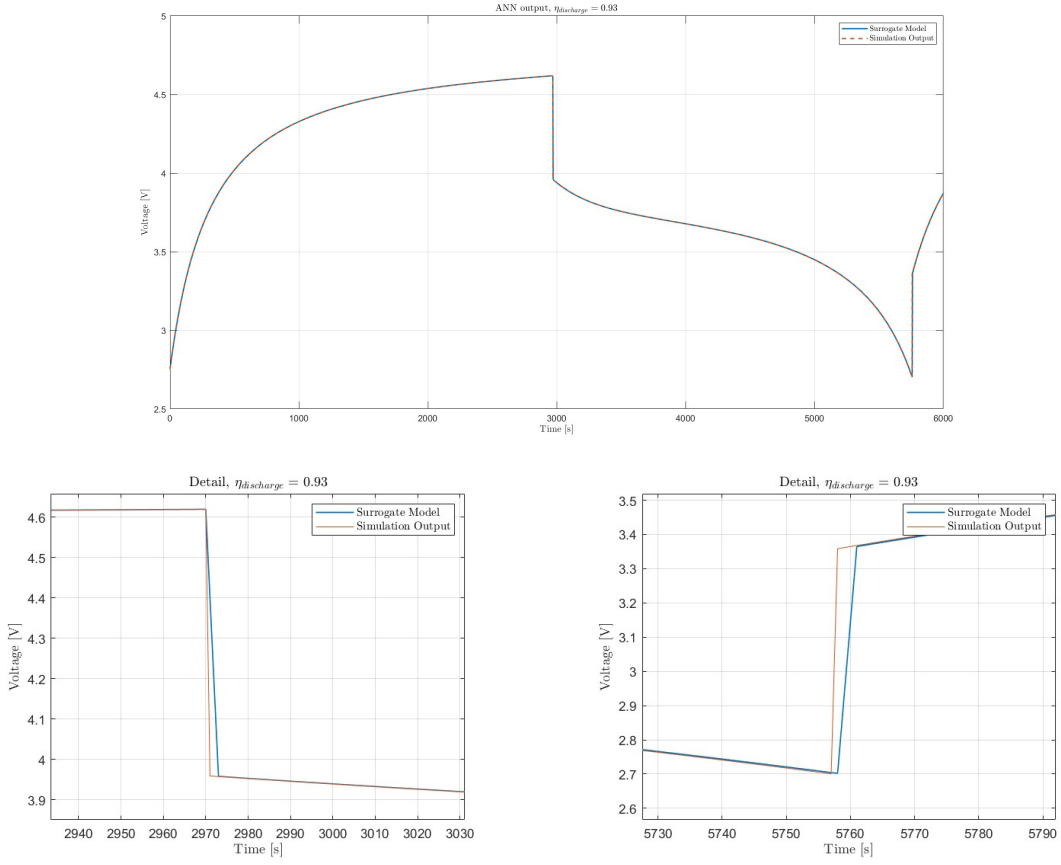


Figure 32: ANN output

In *figure* 32, inference is performed for $\eta_{discharge} = 0.93$ ($N = 100$). The approximation is quite good, except near the discontinuities.

2.2 Residual life prediction by MCMC-MH

Using the Artificial Neural Network surrogate model of the battery's Digital Twin (*section* 2.1), the Markov Chain Monte Carlo algorithm is used to update the model parameter ($\eta_{discharge}$) based on the corresponding voltage measurement (V_{meas}). For simplicity, let $\eta \leftarrow \eta_{discharge}$ and $V \leftarrow V_{meas}$

from now on. Measurement data are obtained by adding noise (*figure* 31) to the simulation's output, which is run with the *true* value $\eta = 0.9$.

Prior distribution. Given the parameter:

$$\eta = \frac{AH^{(100 \text{ cycles})}}{AH^{(0 \text{ cycles})}},$$

its values can only lie in the interval $[0, 1]$. Additionally, based on data from similar batteries, η is more likely to be close to 1 than to 0: the prior distribution $p(\eta_k)$ should be skewed towards 1. A β distribution is chosen (*figure* 33).

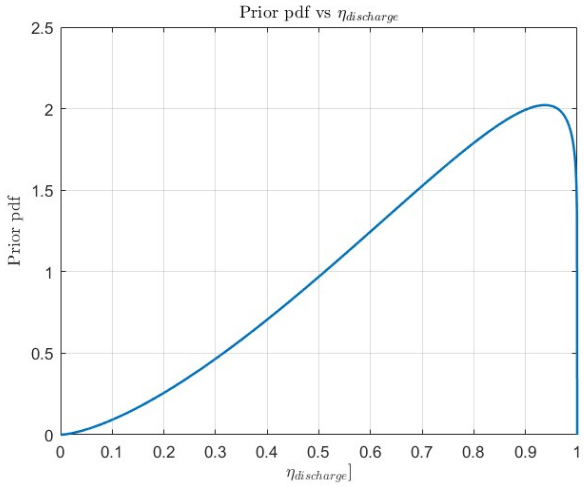


Figure 33: prior distribution

MCMC method. The proposal density $q(\eta_{k+1} \mid \eta_k)$ is user-defined as a Gaussian distribution with proposal standard deviation σ_{prop}. Its symmetry guarantees the *ergodicity* of the chain. For $N = 100$, σ_{prop} is set to $1.5 \cdot 10^{-2}$, but it is adjusted for different values of N to ensure $\alpha \approx 0.3$.

In the MCMC algorithm η is the parameter to estimate, based on the measurement outcomes and the corresponding time steps. Consequently, the likelihood function is expressed as the product of independent Gaussian distributions:

$$\mathcal{L}(\mathbf{V} \mid \eta_k) = p(\mathbf{V} \mid \eta_k) = \prod_{t=0}^{T} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{[V(t) - V_{ANN}(t, \eta_k)]^2}{2\sigma^2}},$$

where $\mathbf{V}$ represents the voltage vector (over time), and η_k the state at iteration k.

Similarly to what is described in *section* 1.4, the log-likelihood is employed:

$$\log \mathcal{L}(\mathbf{V} \mid \eta_k) = \sum_{t=0}^{T} \left\{ \log \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{[V(t) - V_{ANN}(t, \eta_k)]^2}{2\sigma^2} \right\}$$

The MCMC pseudocode, adapted to the specific case, is presented in *algorithm* 3. Starting point is reasonably set to $\eta = 0.85$. For more detailed insights, refer to *section* 1.4.

Algorithm 3 Updating η - Monte Carlo Markov Chain

1. Initialization:
   - Select prior pdf $p(\eta_k)$ s.t. $\eta_k \sim \beta(2.5, 1.1)$
   - Select proposal pdf $q(\eta_{k+1} \mid \eta_k) \sim \mathcal{N}(\eta_k, \sigma^2_{prop})$
   - Select starting point η_1 for the Markov Chain and compute $p(\eta_1)$, $\mathcal{L}(\mathbf{V} \mid \eta_1)$

repeat

2. Sample $\hat{\eta}_{k+1}$ from proposal $q(\eta_{k+1} \mid \eta_k)$
3. Compute $\log \alpha(\hat{\eta}_{k+1} \mid \eta_k) \leftarrow \log p(\hat{\eta}_{k+1})$, $\log \mathcal{L}(\mathbf{V} \mid \hat{\eta}_{k+1})$
4. Set $\eta_{k+1} \leftarrow \hat{\eta}_{k+1}$ in the chain with probability α, otherwise set $\eta_{k+1} = \eta_k$

until desired L_{chain} is reached

5. Erase the burn-in period and proceed with thinning

return $\hat{p}(\eta \mid \mathbf{V})$

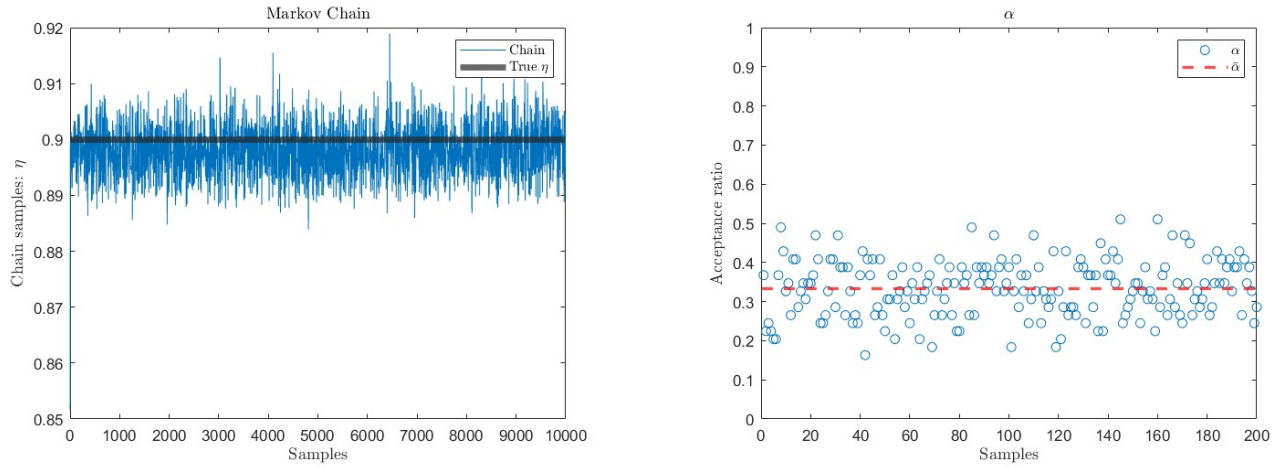


Figure 34: Markov Chain and acceptance ratios (@ $N = 100$)

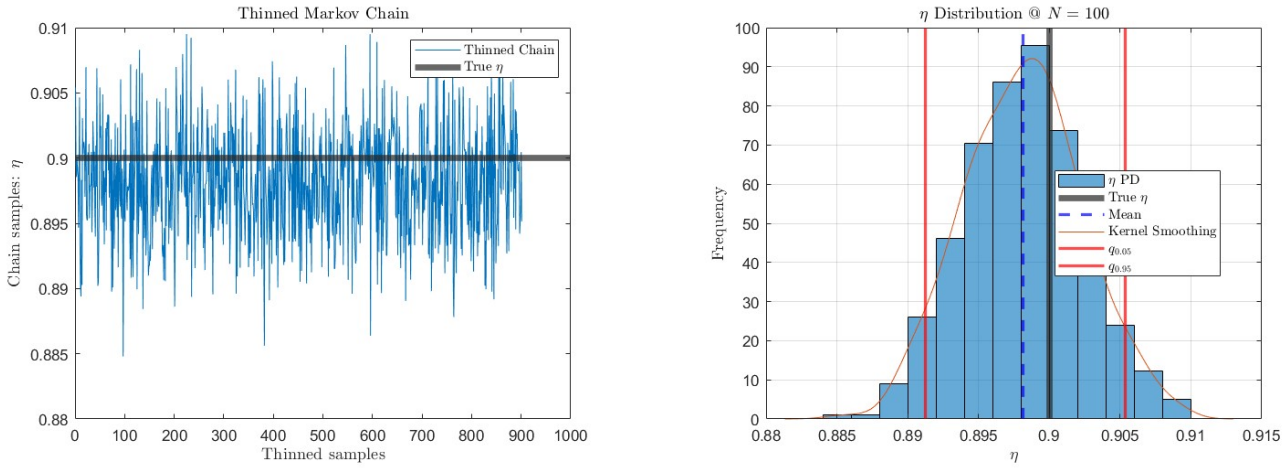


Figure 35: Thinned Markov Chain and posterior distribution $\hat{p}(\eta \mid \mathbf{V})$ (@ $N = 100$)

In *figure* 34, the obtained Markov Chain (@ $N = 100$) is shown, along with the corresponding acceptance ratios, averaged every 50 samples (as done also in *chapter* 1). The chosen proposal

variance stabilizes the mean around 0.3.

In *figure* 35, the posterior distribution $\eta \mid \mathbf{V}$ illustrates the interval between the 5% and 95% quantiles, defined as the values corresponding to cumulative probabilities of 5% and 95%. The true value falls within this interval, which is centered around the estimated expected value (i.e. the mean).

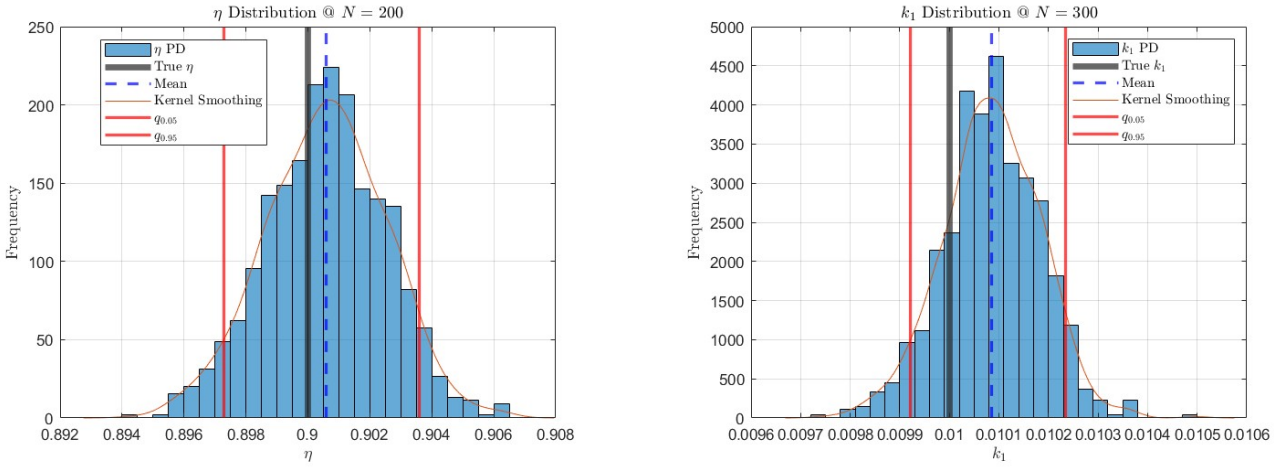


Figure 36: $\hat{p}(\eta \mid \mathbf{V})$ @ $N = 200$ *and* @ $N = 300$

Finally, *figure* 36 presents the posterior distributions evaluated at 200 and 300 cycles. Note that these histograms are obtained by adjusting the proposal variance to ensure the acceptance ratio oscillates around 0.3. In this case, the confidence interval is narrower, as the effect of degradation is higher, thus reducing the uncertainty in the parameter estimation. This is more evident when the number of cycles increases.

RUL estimation. From the η distribution, the corresponding k_1 posterior can be evaluated, leveraging its definition:

$$k_1 = \frac{1-\eta}{10}$$

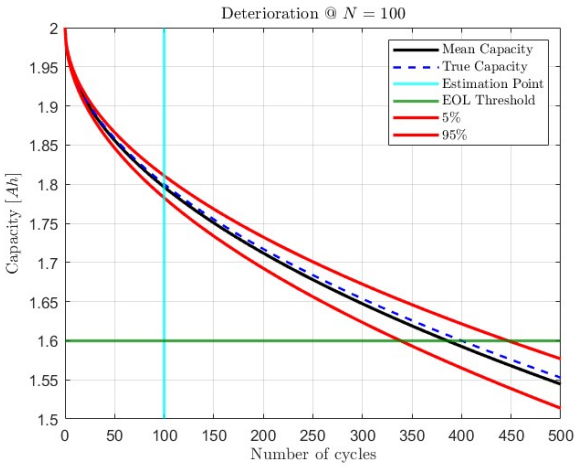
The *EOL* is defined as the number of cycles at which the capacity of the battery crosses the 80% threshold (with respect to the rated battery capacity AH_0: $AH_{EOL} = 0.8 \cdot 2\ Ah = 1.6\ Ah$). Since:

$$AH(N) = AH_0(1 - k_1\sqrt{N})$$

It follows immediately that:

$$\frac{AH_{EOL}}{AH_0} = 0.8 = 1 - k_1\sqrt{N_{EOL}} \implies N_{EOL} = \left(\frac{1-0.8}{k_1}\right)^2, \quad RUL = N_{EOL} - N.$$

In *figures* 37, 38, and 39, the estimated distribution of k_1 (in each case: $N = 100$, 200, and 300) is used to estimate the posterior capacity behavior, along with the *EOL* distribution. Note that the right-hand distribution statistically represents the interval projected onto the 1.6 Ah line on the left.



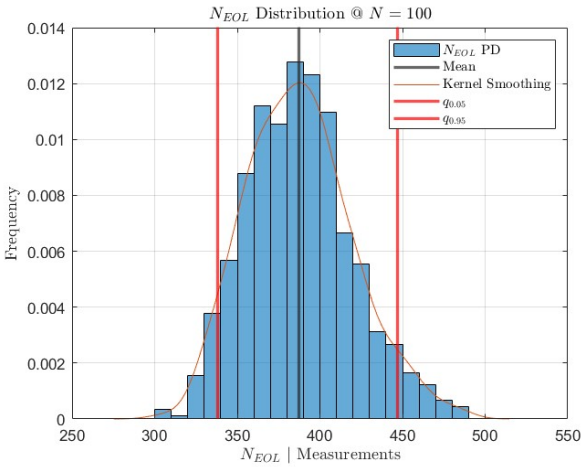
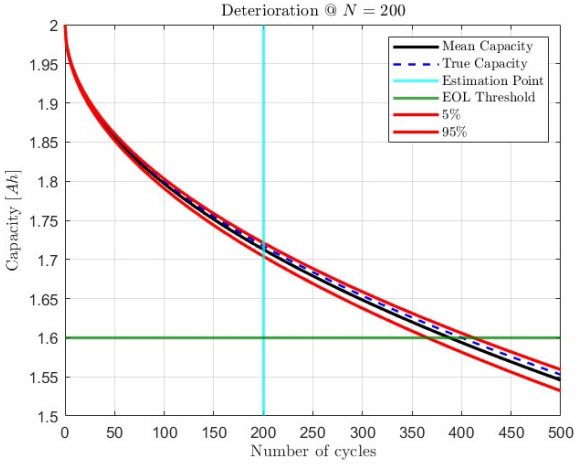


Figure 37: $AH(t)$ and EOL posterior distribution @ $N = 100$



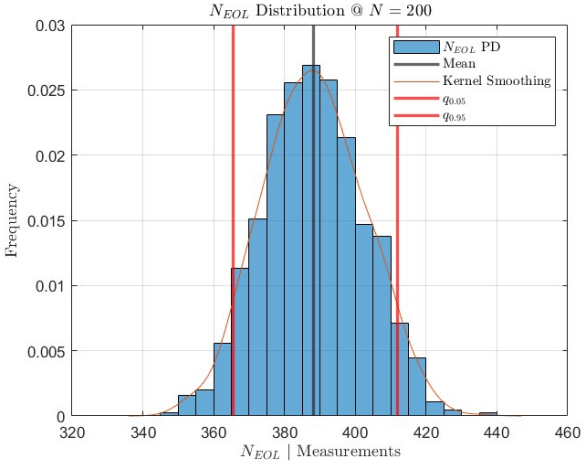
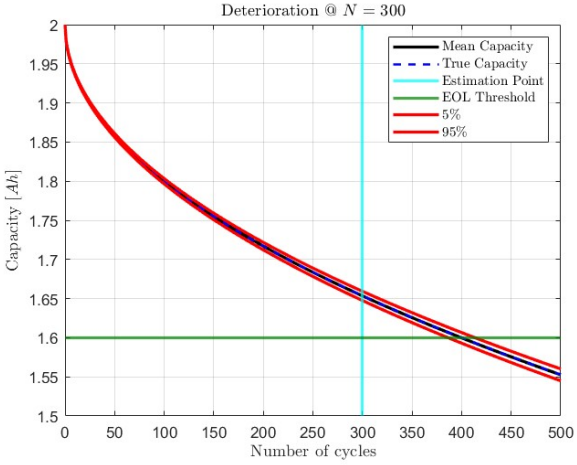


Figure 38: $AH(t)$ and EOL posterior distribution @ $N = 200$



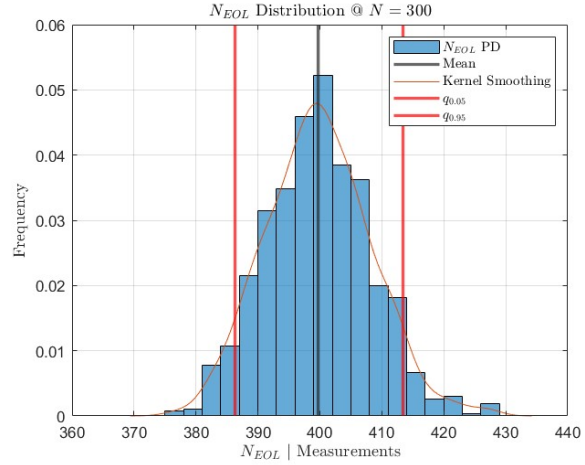


Figure 39: $AH(t)$ and EOL posterior distribution @ $N = 300$

As stated before, the higher the N at which k_1 is estimated, the lower the uncertainty on the End of Life. This means that the interval (and the space between quantiles) becomes narrower.

Interestingly, in *figure* 40, the measurement sampling frequency is reduced from 1 Hz to 0.01 Hz at $N = 300$: having fewer samples reduces the amount of information available to estimate k_1, thus increasing the posterior uncertainty (to the point where the battery could fail at any moment!).

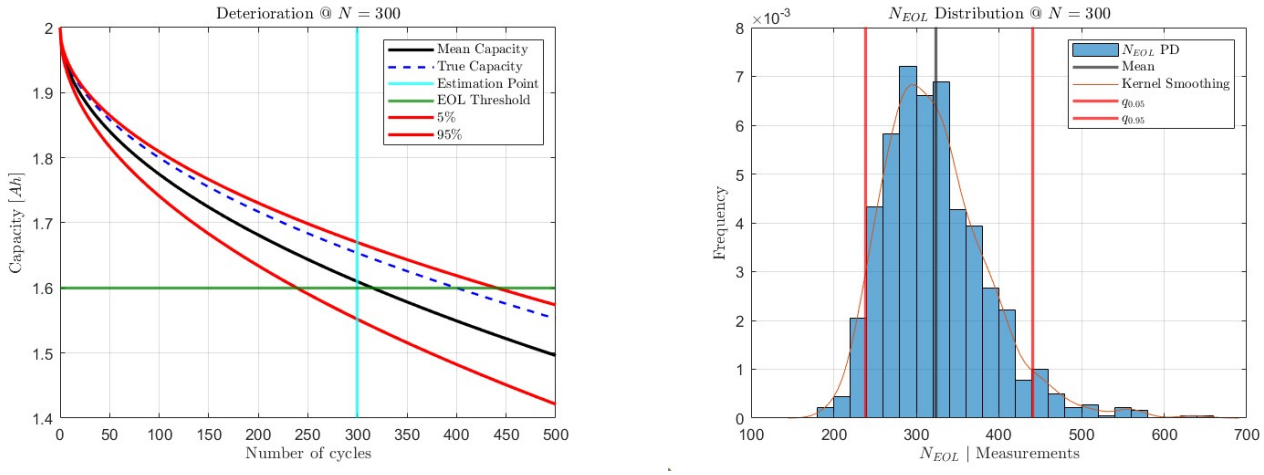


Figure 40: sampling measurements every 100 s @ $N = 300$

2.3 Residual life prediction by Particle Filter

Measurements of the capacity loss due to battery fade degradation from a real lithium battery are provided over 837 charge-discharge cycles.

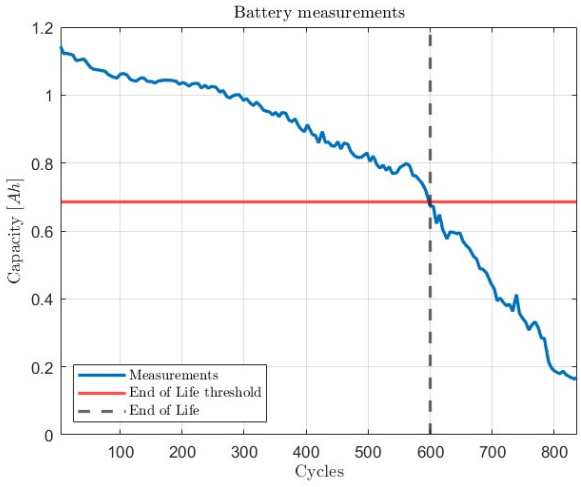


Figure 41: measurements of lithium battery capacity

The goal is to infer the Residual Useful Life (RUL) of the battery: the number of cycles remaining before the battery is considered dead. In *figure* 41, the measurements are shown along with the End of Life (EOL) threshold, set at 60% of the initial capacity.

Particle Filter, a *sequential importance sampling* algorithm, is used for the battery's diagnosis and prognosis. The chosen model for battery deterioration is:

$$Q(n) = a + b \cdot (1 - e^{c \cdot n}),$$

where n is the number of full charge-discharge cycles, $Q(n)$ the battery capacity (in *section* 2.1 referred to as AH), and a, b, c form the state of the system, i.e.:

$$\mathbf{x} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}.$$

Particle Filter allows the estimation of the posterior probability $p(\mathbf{x}_k \mid Q_{0:k}, n_{0:k})$ of the state $\mathbf{x}_k$ given the sequential measurements $(Q_{0:k}, n_{0:k})$, thereby enabling the estimation of $p(\mathbf{RUL}_k \mid Q_{0:k}, n_{0:k})$. This approach relies on a set of two equations:

1. **Process equation**: state transition function, describing how the state vector $\mathbf{x}$ changes over time (subsequent steps k). The assumed process equation is the following:

$$\begin{cases} a_k = a_{k-1} + \mathcal{N}(0, \sigma_a^2) \\ b_k = b_{k-1} + \mathcal{N}(0, \sigma_b^2) \\ c_k = c_{k-1} + \mathcal{N}(0, \sigma_c^2) \end{cases} \implies \mathbf{x}_k = \mathbf{x}_{k-1} + \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \sigma_a^2 & 0 & 0 \\ 0 & \sigma_b^2 & 0 \\ 0 & 0 & \sigma_c^2 \end{bmatrix}\right)$$

   Standard deviations are set to $\sigma_a = 10^{-3}$, $\sigma_b = 10^{-4}$, and $\sigma_c = 10^{-5}$.

2. **Measurement equation**: function linking the observable capacity Q_k with the state $\mathbf{x}_k$ and the input n_k. Assuming Gaussian measurement noise:

$$Q_k = a_k + b_k (1 - e^{c_k n_k}) + \mathcal{N}(0, \sigma_Q^2),$$

   where σ_Q is the measurement uncertainty, set a priori to $\sigma_Q = 0.05\ Ah$. This equation is used into a Gaussian likelihood function to assess the likelihood of each estimated particle (i.e. x_k^i).

Pseudocode for the Particle Filter is shown in *algorithm* 4. The number of iterations is chosen to not exceed the EOL computed from the actual measurements. An important hyperparameter to be tuned is the number of particles, $n_{particles}$, which is initially set to 2500.

These particles need to be initialized, and the most straightforward choice is to use a multivariate Gaussian distribution, with $\mathbf{\Sigma}$ initially set as a diagonal covariance matrix and an initial mean state $\bar{\mathbf{x}}_0 = (1.054, 0.021, 0.004823)$.

It should be noted that the likelihood is evaluated by comparing the measured capacity with the expected value of the measurement function, based on the Gaussian modeling of the measures. The log-likelihood is not required (unlike in other cases) since there is only a single value, rather than a product of multiple ones.

The results of the simulation are shown in *figures* 42, 43, 44, and 45. States a, b, c, and capacity Q are evaluated with respect to the number of cycles.

As the prior is continuously adjusted based on measurements and process equation, the estimated uncertainty decreases: filtering allows to concentrate the particles on the most likely region,

Algorithm 4 State estimation - Particle Filter

1. Initialization:
   - Select $n_{particles}$
   - Initialize the state particles according to prior knowledge $\mathbf{x_0} = \mathcal{N}(\bar{\mathbf{x}}_\mathbf{0}, \mathbf{\Sigma}_{\mathbf{x}_0})$

repeat

2. Evaluate measurement function $\forall$ particle i:

$$Q_k^i = a_k^i + b_k^i \left(1 - e^{c_k^i n_k}\right)$$

3. Evaluate likelihood function $\forall i$:

$$\mathcal{L}_i = \mathcal{L}(Q_{meas,k} \mid \mathbf{x}_k^i) = \frac{1}{\sqrt{2\pi\sigma_Q^2}} e^{-\frac{(Q_{meas,k} - Q_k^i)^2}{2\sigma_Q^2}}$$

4. Normalize likelihoods as weights:

$$w_i = \frac{\mathcal{L}_i}{\sum_i \mathcal{L}_i}$$

5. Resample particles from **Multinomial**$(n_{particles}, \mathbf{w})$ (i.e. posterior distribution)
6. Predict the next particles with the process equation (i.e. next prior distribution)

until $EOL_{measures}$

return $\hat{p}(\mathbf{x}_k \mid Q_{meas,0:k})$

since the less probable ones are filtered out. However, the distribution does not always become narrower, as both the variance of the measurements and of the process must be taken into account.

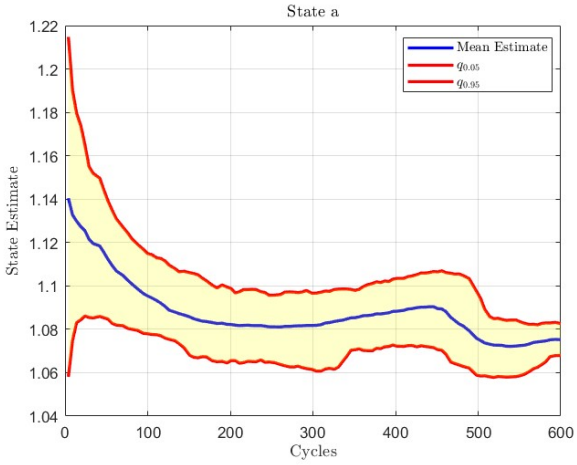


Figure 42: a

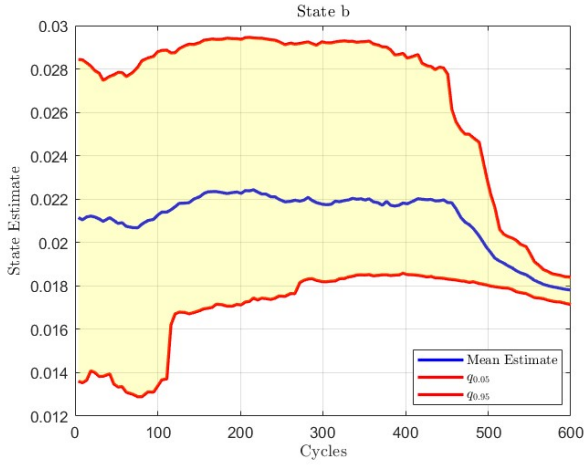


Figure 43: b

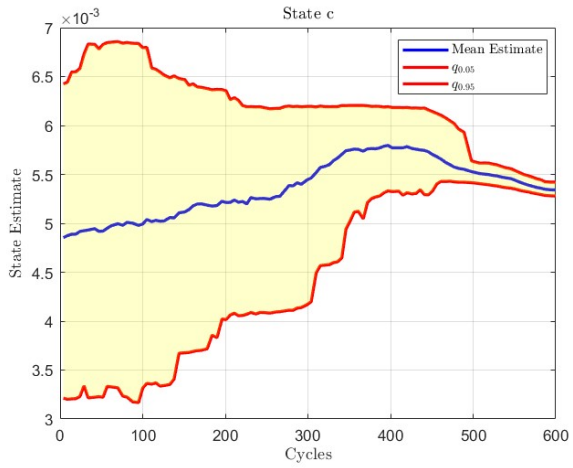


Figure 44: c

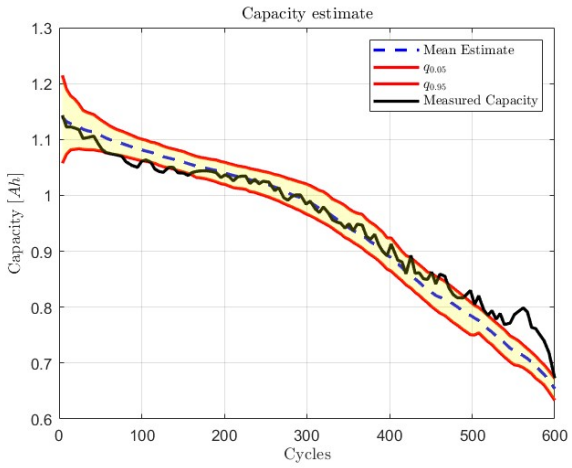


Figure 45: Q

In *figure* 46, the estimated Remaining Useful Life (RUL) is plotted. The *true* value is assumed to be linear, while the estimated one is forecasted using the process equation for each particle within the same iteration.

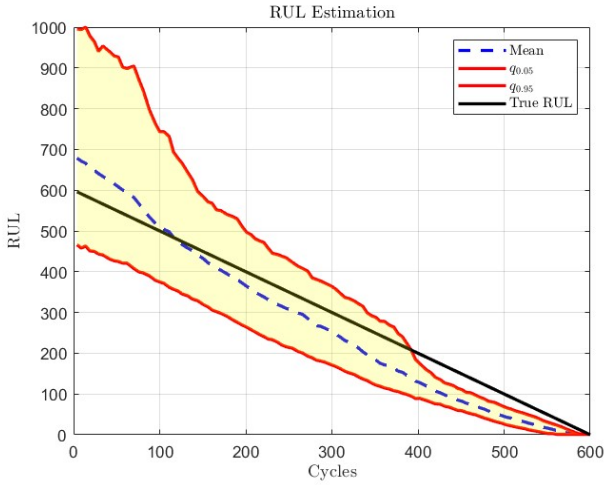


Figure 46: RUL estimation

Figure 47 illustrates why the RUL uncertainty decreases over time. The estimated capacity is forecasted starting from the current state using the process equation, resulting in a narrower confidence interval as more cycles are completed.

In *figure* 48, it can be observed how the prior distribution evolves, as the relationships between the states change from cycle 4 (no correlation) to cycle 546 (correlation and lower variance).

Lastly, it is important to understand the effects of changing certain parameters:

1. **Number of Particles**: comparing *figure* 49 and *figure* 46, fewer particles reduce uncertainty but compromise the ability to represent regions below and above the mean, while also decreasing computational cost. Conversely, increasing $n_{particles}$ improves the representation of the distribution and of the uncertainty. This improvement becomes particularly significant with non-linear RUL behaviors.

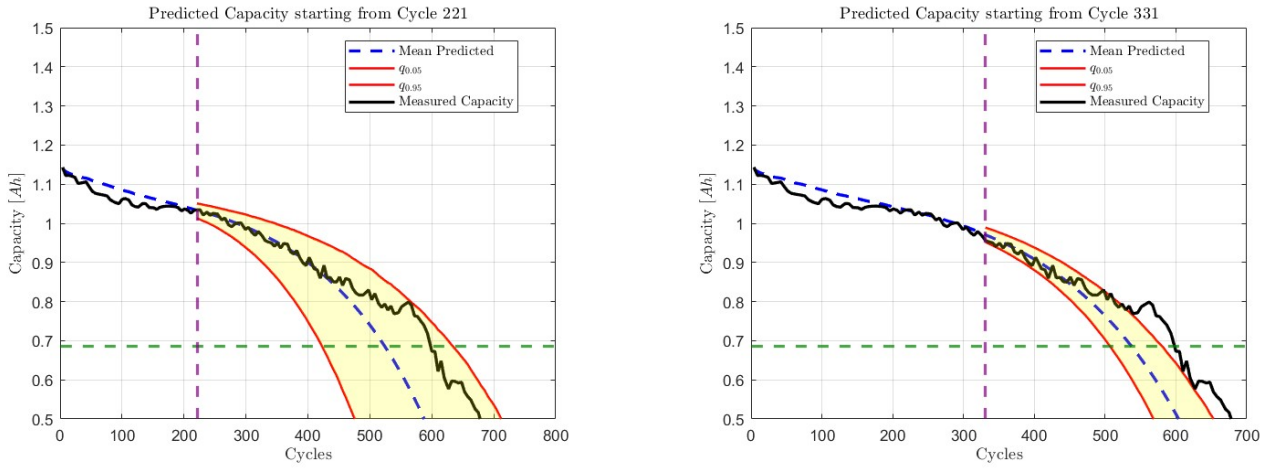


Figure 47: predicted capacity, from different times

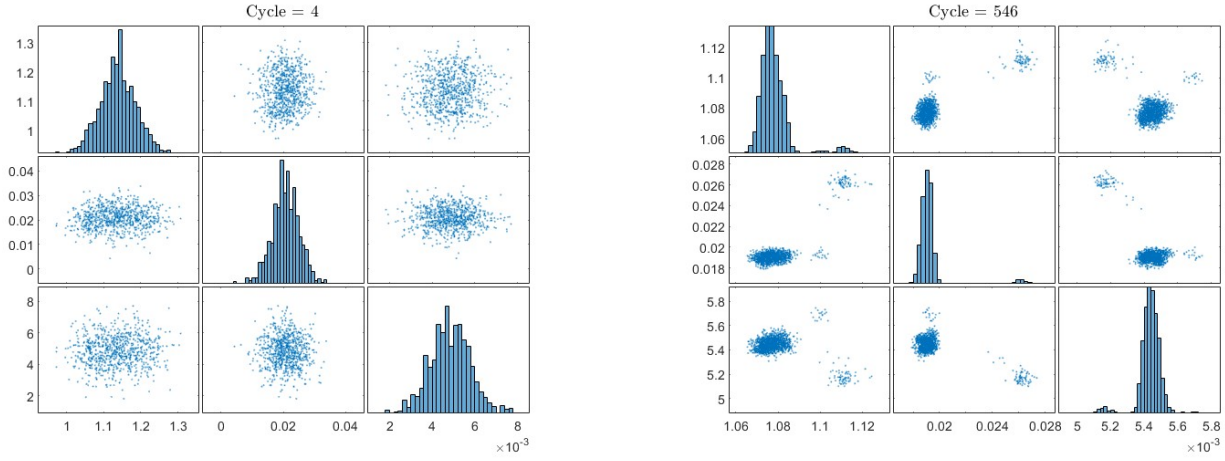


Figure 48: states at $N = 4$ and $N = 546$

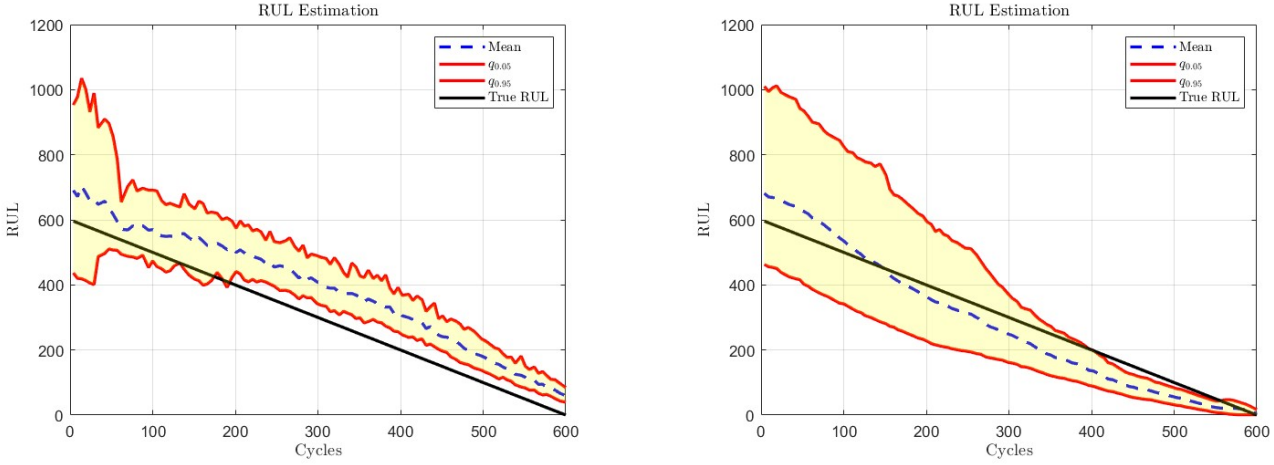


Figure 49: RUL estimation at $n_{particles} = 100$ and $n_{particles} = 5000$

2. **Initial State Covariance**: the initial state covariance $\mathbf{\Sigma_{x_0}}$ is modified by increasing the standard deviations by a 10 factor. The results, shown in *figure* 50, reveal a broader range

of uncertainty at the beginning, while gradually converging to the correct distribution over time. This situation occurs when the prior knowledge of the state is highly imprecise.

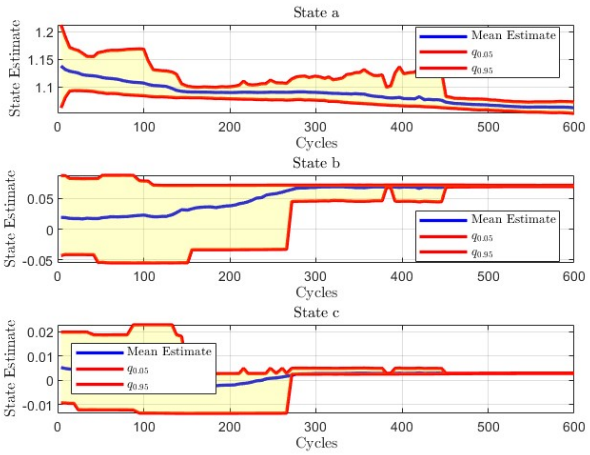


Figure 50: changing $\boldsymbol{\Sigma}_{\mathbf{x_0}}$

3. **Initial State Mean**: when $\mathbf{x}_0$ remains within the same order of magnitude, the algorithm successfully converges to the correct distribution. However, if $\mathbf{x}_0$ is altered by several orders of magnitude, the results become unsatisfactory.

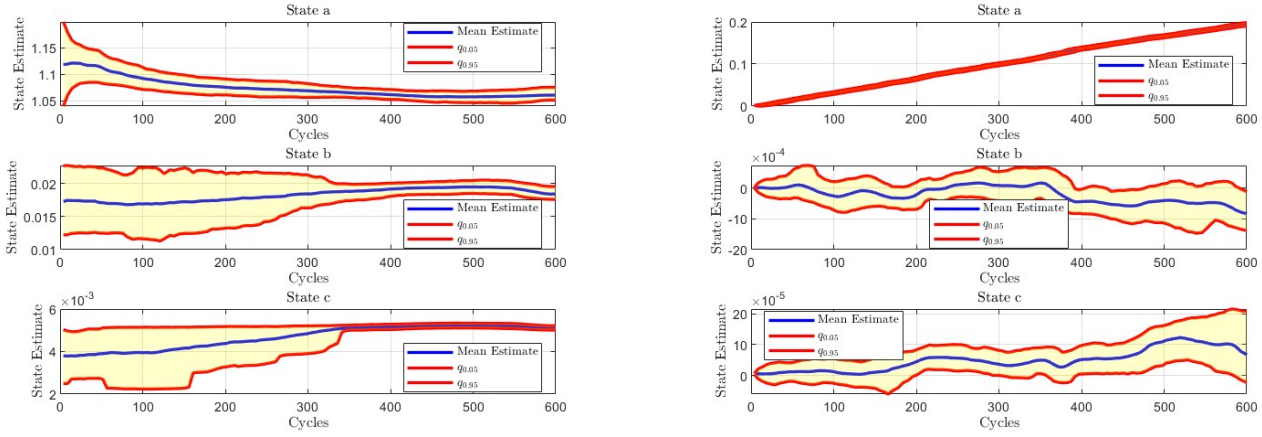


Figure 51: $\mathbf{x}_0 \cdot 0.8$ *and* $\mathbf{x}_0 \cdot 0.001$

4. **Measurement Noise**: if it increases (*figure* 52), the uncertainty grows, as all measurements are affected by higher noise (reducing the *impact* of likelihood). This is why proper sensor design is essential.

5. **Process Standard Deviations**: when they increase, uncertainty grows. It is worth noting that, in this case, a random walk process equation is used, which adds noise to the previous particles, instead of adopting a deterministic one (e.g. Paris law).

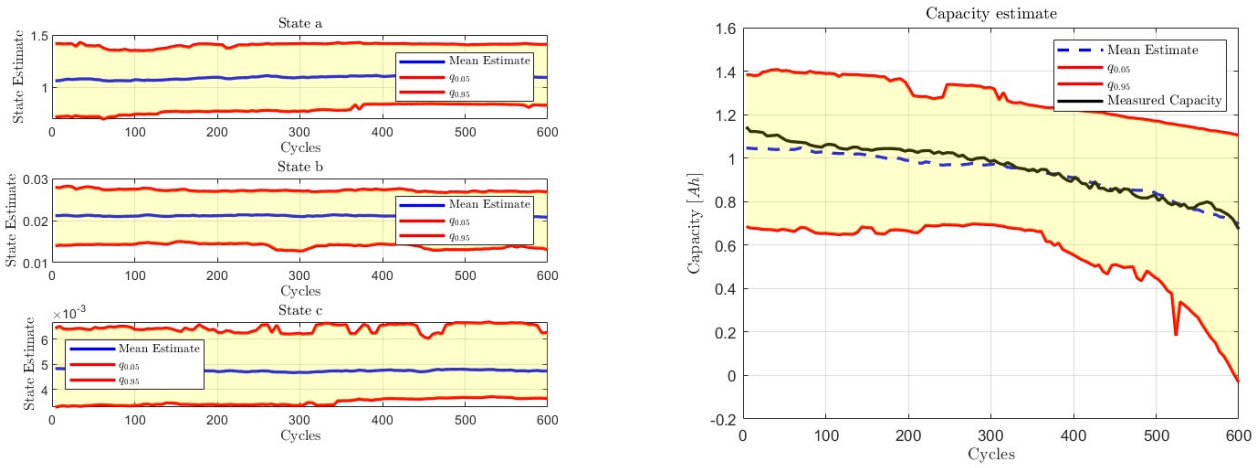


Figure 52: measurement noise multiplied by 100

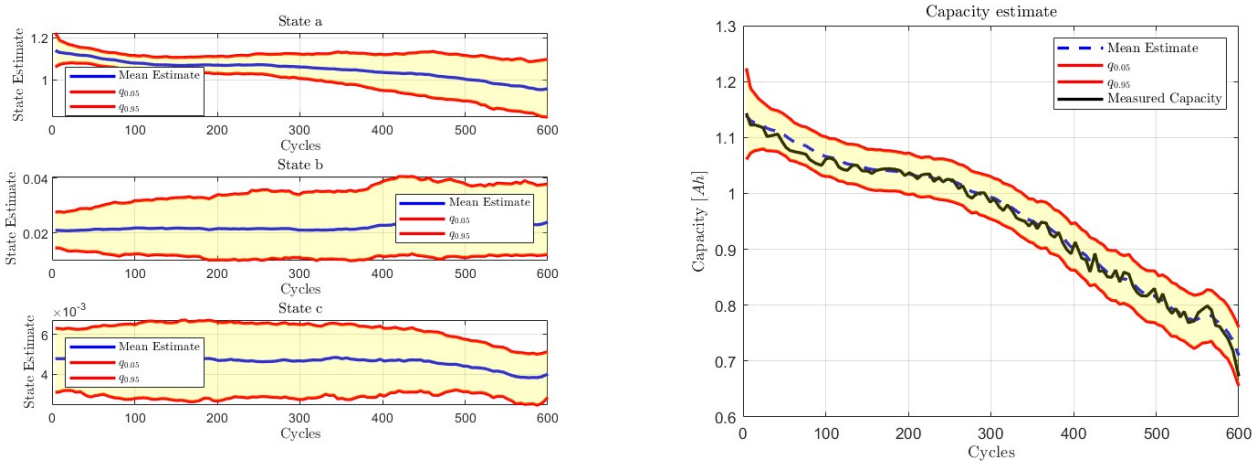


Figure 53: process standard deviations multiplied by 10