UNIVERSIDADE FEDERAL DE PERNAMBUCO

DEPARTAMENTO DE ENGENHARIA MECÂNICA

CURSO DE GRADUAÇÃO EM ENGENHARIA NAVAL

ANÁLISE DE CONFIABILIDADE DE DUTOS SUBMARINOS CORROÍDOS UTILIZANDO REDES NEURAIS

Lucas Gabriel Borges dos Santos

RECIFE, SETEMBRO / 2024

LUCAS GABRIEL BORGES DOS SANTOS

ANÁLISE DE CONFIABILIDADE DE DUTOS SUBMARINOS CORROÍDOS UTILIZANDO REDES NEURAIS

Trabalho de conclusão de curso apresentado à Universidade Federal de Pernambuco, em Recife/PE, como requisito parcial para a obtenção do título de Bacharel em Engenharia Naval.

ORIENTADOR: Adriano Dayvson Marques Ferreira

RECIFE, SETEMBRO / 2024

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Santos, Lucas Gabriel Borges dos.
Análise de confiabilidade de dutos submarinos corroídos utilizando redes neurais / Lucas Gabriel Borges dos Santos. - Recife, 2024.
65 p.

Orientador(a): Adriano Dayvson Marques Ferreira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Naval - Bacharelado, 2024.
Inclui referências, apêndices.

1. Dutos submarinos corroídos. 2. GRASP. 3. Optuna. 4. PSO. 5. Redes neurais artificiais. I. Ferreira, Adriano Dayvson Marques. (Orientação). II. Título.

620 CDD (22.ed.)

LUCAS GABRIEL BORGES DOS SANTOS

ANÁLISE DE CONFIABILIDADE DE DUTOS SUBMARINOS CORROÍDOS UTILIZANDO REDES NEURAIS

Trabalho de Conclusão de Curso apresentado à Universidade Federal de Pernambuco como requisito parcial para obtenção do título de Bacharel em Engenharia Naval.

Aprovado em: 25/10/2024

BANCA EXAMINADORA

_____

Prof. Dr. Adriano Dayvson Marques Ferreira (Orientador)
Universidade Federal de Pernambuco

_____

Profa. Dra. Luciete Alves Bezerra (Examinador Interno)
Universidade Federal de Pernambuco

_____

Prof. Dr. Renato de Siqueira Motta (Examinador Externo)
Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço ao meu orientador, Professor Adriano Davyson Marques Ferreira, pela paciência, dedicação e confiança em me guiar durante a minha jornada no curso.

Agradeço aos professores do curso de Engenharia Naval da Universidade Federal de Pernambuco, pelos ensinamentos recebidos, tanto dentro quanto fora da sala de aula.

Agradeço aos colegas que fiz ao longo da minha trajetória, que me apoiaram em momentos difíceis e compartilharam risadas em momentos alegres.

Agradeço à minha família, por todo o apoio, esforço e amor ao longo de toda a minha vida, me guiando até este momento. Este momento é graças a vocês.

Agradeço também aos meus cachorros, Luna, Nick e Floquinho (in memoriam), e ao meu gato, Tom. O carinho e afeto de vocês me incentivaram a superar meus desafios.

“The only world I’ve ever know
Sleeps beneath the waves
But I remember”
Evanescence

RESUMO

O transporte de óleo e gás através de dutos, principalmente para dutos submarinos, é amplamente utilizado devido a sua eficiência e eficácia operacional. Nesse contexto, a análise de confiabilidade de dutos submarinos se apresenta essencial para garantir o funcionamento contínuo deste sistema e evitar falhas e acidentes. Os métodos experimentais e numéricos frequentemente utilizados apresentam resultados conservadores, além de demandarem alto tempo e complexidade na análise. No estudo de (FERREIRA et al., 2021), foi proposto o uso de Redes Neurais Artificias (RNA) na previsão da pressão de falha de dutos enterrados corroídos. Brandão (BRANDÃO, 2023) e Silva (SILVA, 2024) avançam nessa pesquisa, investigando, respectivamente, a capacidade das meta-heurísticas *Particle Swarm Optimization* (PSO) e *Greedy Randomized Adaptive Search Procedure* (GRASP) na otimização dos parâmetros das RNAs. Este trabalho tem como objetivo desenvolver uma RNA capaz de prever a pressão de falha em dutos submarinos corroídos, avaliando sua capacidade de substituir a equação proposta por (NETTO, 2009) e oferecendo uma alternativa precisa e rápida na análise de confiabilidade. Foram implementadas quatro RNAs: uma sem otimização e outras três utilizando os métodos PSO, GRASP e Optuna, com o intuito de identificar qual RNA se adequa melhor ao objetivo proposto. Todas as análises foram realizadas em Python (PYTHON SOFTWARE FOUNDATION, 2024). Os resultados indicaram que as RNAs otimizadas apresentaram melhor desempenho em termos de precisão, com destaque para a otimização via Optuna, que se mostrou eficiente em termos de tempo computacional e qualidade das predições. A análise sugere que as RNAs otimizadas são uma alternativa viável para a análise de dutos submarinos corroídos, apresentando vantagens em precisão, velocidade e complexidade das previsões.

Palavras-chave: dutos submarinos corroídos; GRASP; Optuna; PSO; redes neurais artificial.

ABSTRACT

The Transportation of oil and gas through pipelines, particularly subsea pipelines, is widely utilized due to its efficiency and operational effectiveness. In this context, reliability analysis is essential to ensure the continuous operation of this system and to prevent failures and accidents. Experimental and numerical methods commonly employed yield conservative results and require significant time and complexity for analysis. In the study by (FERREIRA et al., 2021), the use of Artificial Neural Networks (ANN) for predicting the failure pressure of corroded buried pipelines was proposed. Brandão (BRANDÃO, 2023) and Silva (SILVA, 2024) advances this research by investigating, respectively, the capability of the metaheuristics *Particle Swarm Optimization* (PSO) and *Greedy Randomized Adaptive Search Procedure* (GRASP) meta-heuristics to optimize the parameters of the ANNs. This work aims to develop an ANN capable of predicting the failure pressure in corroded subsea pipelines, assessing its ability to replace the equation proposed by (NETTO, 2009) and providing a precise and rapid alternative for reliability analysis. Four ANNs were implemented: one without optimization and three utilizing PSO, GRASP, and Optuna methods to identify which ANN best meets the proposed objective. All analyses were conducted using Python (PYTHON SOFTWARE FOUNDATION, 2024). The results indicated that the optimized ANNs demonstrated better performance in terms of accuracy, particularly with Optuna optimization, which proved efficient in both computation time and prediction quality. The analysis suggests that the optimized ANNs are a viable alternative for evaluating corroded subsea pipelines, offering advantages in accuracy, speed, and complexity of predictions.

Keywords: artificial neural networks; corroded subsea pipelines, GRASP, Optuna, Particle Swarm Optimization.

Lista de Figuras

Lista de Tabelas

SUMÁRIO

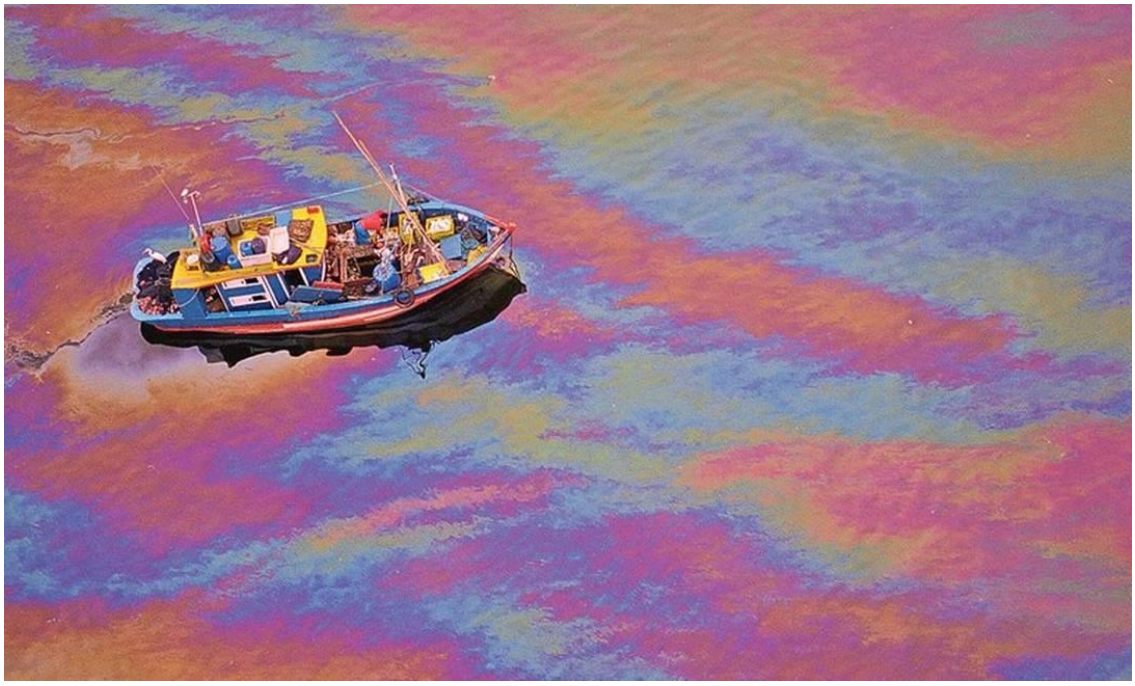
1. INTRODUÇÃO

Os dutos são bastantes utilizados no transporte de óleo e gás a grande distância, devido às suas vantagens na segurança, eficiência e nos custos em comparação com outros tipos de transporte. Porém, com o passar do tempo, os dutos podem apresentar diversos tipos de defeitos (e.g. corrosão e falhas por fadiga) (XIE; TIAN, 2018).

Entre esses defeitos, a corrosão se destaca como um dos principais alvos de estudo. Trata-se de um processo eletroquímico onde ocorre deterioração e perda de material devido a interação com o meio ambiente (FERREIRA et al., 2021). Este processo pode acarretar em falhas e acidentes nos dutos, resultando em impactos econômicos, sociais e ambientais significativos.

Como exemplo de acidentes na área, temos o vazamento de óleo na Baía de Guanabara em 2000, ilustrado na Figura 1. Um duto que conectava a Refinaria Duque de Caxias ao terminal Ilha d'Água, na ilha do Governador, rompeu vazando cerca de 1,3 milhões de litros de óleo combustível que se espalhou por uma área de 40 km², afetando a população, flora e fauna local (GRANDELLE, 2020). Estudos realizados pela Companhia Ambiental do Estado de São Paulo (CETESB) indicaram que o rompimento se deve a exposição do duto a movimentos tortuosos e a corrosão (CETESB, 2018).

Figura 1 - Vazamento de óleo na Baía de Guanabara



Fonte: (GRANDELLE, 2020)

Existem diversas formas de avaliar a corrosão em dutos, entre elas os métodos analíticos descritos em normas técnicas, como a ASME B31G, DNV RF 101, ABS e API 579, bem como os métodos experimentais. Entretanto, essas abordagens apresentam limitações. Os métodos experimentais, por exemplo, envolvem custos elevados e podem ser demorados, enquanto as normas

tendem a fornecer resultados conservadores (i.e., ao evitar subestimar os danos, frequentemente exigem a troca ou reparo do duto antes que seja estritamente necessário).

Uma alternativa que vem sendo cada vez mais adotada é o uso da metodologia de Elementos Finitos (MEF) nas análises, pois essa abordagem gera resultados mais próximos dos obtidos experimentalmente (SAKAKIBARA; KYRIAKIDES; CORONA, 2008). No entanto, os métodos baseados em MEF ainda demandam um tempo considerável de implementação e são complexos, exigindo conhecimento técnico especializado.

Com isso em mente, (NETTO, 2009) realizou uma análise experimental e numérica da corrosão em dutos, com o objetivo de desenvolver uma equação simples e de fácil aplicação para o cálculo da relação das pressões externas de falha de dutos submarinos intactos e corroídos. A equação proposta apresentou resultados comparáveis aos obtidos por análises utilizando Elementos Finitos (EFs), com um erro de apenas 4,4% (NETTO, 2010).

Outro aspecto importante que vem ganhando destaque nos últimos anos é o uso de Redes Neurais Artificias (RNA) para realizar análises e previsões. As RNAs podem ser treinadas para realizar cálculos complexos, proporcionando resultados muito próximos dos valores reais. (FERREIRA et al., 2021) desenvolveu um trabalho utilizando RNAs na área de corrosão de dutos enterrados, obtendo resultados com erro médio de apenas 1,64%.

Além disso, é fundamental considerar as incertezas atribuídas aos parâmetros que impactam a falha dos dutos, como a própria corrosão. Estudos que adotam uma abordagem probabilística para a avaliar a probabilidade de falha de dutos submarinos sofrendo efeitos de fadiga (LI et al., 2021) e de corrosão (BEN SEGHIER; MUSTAFFA; ZAYED, 2022) têm sido desenvolvidos com o objetivo de aumentar a confiabilidade das análises.

O objetivo deste Trabalho de Conclusão de Curso (TCC) é desenvolver uma RNA e avaliar a sua capacidade em substituir a equação desenvolvida por (NETTO, 2009) na análise de pressão de falha de dutos submarinos corroídos. Para este objetivo, este trabalho está estruturado da seguinte forma: No **Capítulo 1**, apresenta-se a introdução, que contém o objetivo da pesquisa. No **Capítulo 2**, desenvolve-se a revisão bibliográfica, abordando os estudos atuais na área, bem como a justificativa e relevância do tema. O **Capítulo 3** trata da fundamentação teórica, onde serão explorados os conceitos necessários para o desenvolvimento do trabalho. No **Capítulo 4**, descreve-se a metodologia aplicada. No **Capítulo 5**, apresentam-se os resultados obtidos. O **Capítulo 6** traz a conclusão, seguido pelas referências e pelo **Anexo A**, que contém o código desenvolvido para a análise.

1.1 OBJETIVOS

O objetivo geral deste Trabalho de Conclusão de Curso (TCC) é desenvolver e treinar uma RNA capaz de calcular a pressão de falha para dutos submarinos corroídos, adotando a equação

desenvolvida por (NETTO, 2009) e avaliar a capacidade desta RNA de substituir a equação na análise de dutos.

Afim de atingir este objetivo, são propostos os seguintes objetivos específicos:

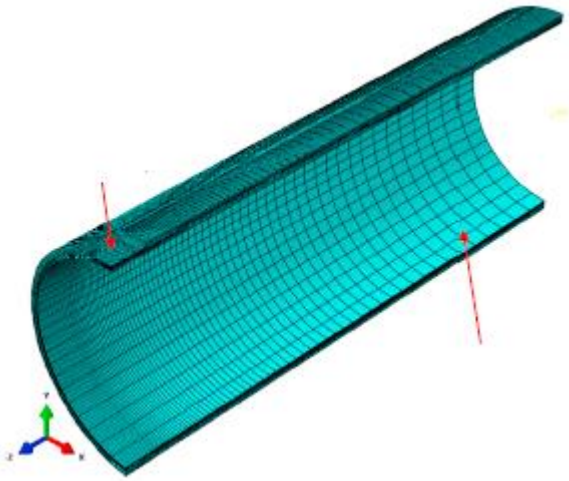
- Desenvolver um banco de dados para treino e validação da RNA adotando a equação proposta por (NETTO, 2009) como base.
- Gerar uma RNA para análise de pressão de falha de dutos submarinos.
- Implementar uma rotina computacional que integre o algoritmo da RNA com o algoritmo do GRASP, PSO e Optuna para otimização.
- Calcular a probabilidade de falha utilizando o método de Monte Carlo de um grupo de amostras gerado pelo método Latin Hypercube Sampling.
- Comparar os resultados obtidos pelas RNAs desenvolvidas com os obtidos adotando a equação desenvolvida por (NETTO, 2009).
- Investigar a aplicabilidade dos modelos desenvolvidos, em termo de qualidade dos resultados e tempo hábil das análises.

2. REVISÃO BIBLIOGRÁFICA

As análises na corrosão de dutos são tradicionalmente efetuadas utilizando métodos analíticos descritos em normas, como a ASME B31G, DNV RF 101, ABS e API 579(CHOI et al., 2016). Porém, os métodos analíticos apresentam certas limitações, como o método experimental que considera o comportamento não-linear dos dutos de forma indireta, em comparação aos métodos numéricos.

Portanto, atualmente são mais utilizados métodos computacionais nas análises, com destaque para o uso de Elementos Finitos (EFs), que conseguem gerar modelos com comportamento próximo aos casos reais com menor carga computacional, como mostrado na Figura 2.

Figura 2 - Exemplo de modelo de Elementos Finitos (EFs)



Adaptado de: (MONDAL; DHAR, 2019)

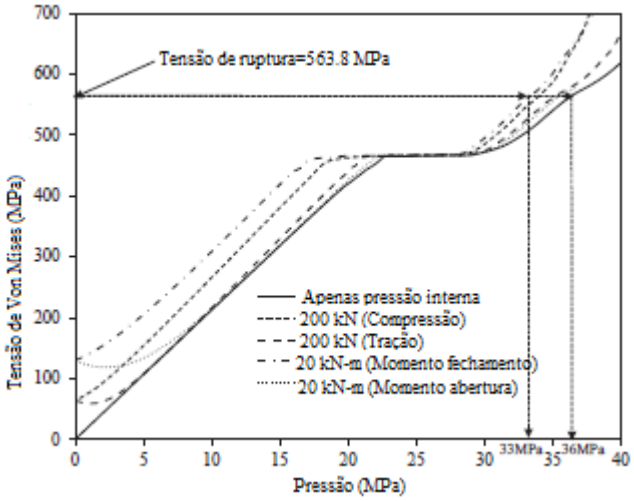
Os modelos EFs necessitam das dimensões dos defeitos obtidos em casos reais para sua geração, e as ferramentas mais utilizadas na literatura atual para esse fim são as In-line Inspection (ILI). As ferramentas ILI apresentam a vantagem de detectar diversos tipos de características e defeitos do duto em uma única inspeção, como o diâmetro interno do duto, a espessura da corrosão, danos mecânicos e perda de metal (Xie & Tian, 2018).

Com o uso dos modelos EFs, é possível desenvolver estudos mais específicos na área de corrosão, como o estudo recente (WANG; ZHOU, 2019) sobre o efeito da corrosão nos joelhos dos dutos, regiões curvas que conectam os dutos e que apresentam maior stress devido a sua curvatura. Apesar do estudo apresentar resultados satisfatórios sobre o tema, carece de uma comparação entre os dados obtidos para joelhos e para regiões retas.

Outro estudo realizado na área tem foco na influência de forças axiais e momentos fletores devido a cargas externas na pressão de ruptura de dutos corroídos (MONDAL; DHAR, 2019). O estudo demonstra que a aplicação de forças de compressão e momentos fletores de fechamentos em áreas corroídas do duto podem acarretar na diminuição na pressão de ruptura, como demonstrando na Figura 3. Pode-se perceber que as curvas de compressão e momento fechamento apresentam tensão

de Von Mises significativamente maior (eixo y) comparada as outras curvas, que apresentam comportamento similar.

Figura 3 - Efeitos da força axial e momento fletor na pressão de ruptura.

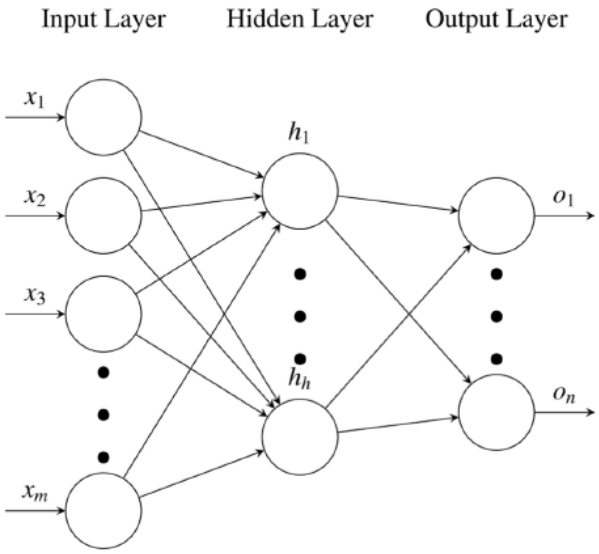


Adaptado de: (MONDAL; DHAR, 2019)

Com ambos os estudos (WANG; ZHOU, 2019) (MONDAL; DHAR, 2019) apresentando informações importantes a serem considerados em estudos futuros, o foco na indústria atual é na automação do processo de análise de dutos, de modo a diminuir a demanda de engenheiros especializados na área de avaliação de integridade. Ferreira et al. (2021) apresenta dois artigos com este foco, sendo o primeiro com objetivo de demonstrar o PIPEFLAW, um programa desenvolvido pelo PADMEC, com a capacidade de calcular a pressão de ruptura para modelos EFs com 95% de precisão (FERREIRA et al., 2021).

O segundo artigo demonstra o desenvolvimento de um sistema de análise multiresolução utilizando redes neurais para a automação da análise de dutos enterrados (FERREIRA et al., 2021). O estudo apresenta a vantagem da utilização de perfis river-bottoms, capaz de representar corrosões 3D em formato 2D, diminuindo assim a carga computacional da análise. Porém, o ponto principal do estudo é o uso de Redes Neurais Profundas (RNPs) na automatização da análise. O funcionamento das RNP consiste em cada unidade de processamento aplicarem uma função ativa com valores de peso e bias afim de se obter um valor real. A Figura 4 representa a arquitetura de uma rede neural.

Figura 4 - Arquitetura de uma Rede Neural



Fonte: (FERREIRA et al., 2021)

As RNPs precisam ser treinadas com dados reais a fim de desenvolver sua capacidade analítica. (FERREIRA et al., 2021) utiliza dados estatísticos para geração de modelos EFs com corrosões aleatórias para a fase de treinamento. As análises utilizando RNPs apresentaram então resultados com uma diferença de apenas 5% em comparação a modelos EFs.

Brandão (BRANDÃO, 2023) e Silva (SILVA, 2024) continuam o trabalho desenvolvido por Ferreira (FERREIRA et al., 2021) ao implementar meta-heurísticas para otimizar os seguintes hiperparâmetros da RNA: (i) número de camadas; (ii) número de neurônios e; (iii) função de ativação, visto que Ferreira (FERREIRA et al., 2021) otimizou estes através de tentativa e erro.

Brandão (BRANDÃO, 2023) adota a meta-heurística *Particle Swarm Optimization* (PSO), que simula o movimento de grupos de animais na busca de soluções ótimas em um espaço de busca. Já Silva (SILVA, 2024) adota a meta-heurística *Greedy Randomized Adaptive Search Procedure* (GRASP), que é uma meta-heurística com um algoritmo de busca de vizinhança, capaz de fugir de ótimos locais.

Netto (NETTO, 2009) realizou um estudo em uma alternativa na análise de dutos submarinos corroídos, desenvolvendo uma equação simples de obter a relação entre a pressão externa de falha de dutos intactos e corroídos. Embora o estudo tenha sido inicialmente voltado para falhas internas, devido a corrosão interna, foi observado que os resultados para falhas externas apresentaram correlação similar, validando a equação para ambas as situações. A equação apresentou boa correlação com os dados obtidos pela análise experimental realizada por (SAKAKIBARA; KYRIAKIDES; CORONA, 2008), com erro de 4,4% (NETTO, 2010).

Um ponto importante, levantando por (MOTTA; FERREIRA; AFONSO, 2024) está na consideração das incertezas nos parâmetros nas análises de falhas de dutos. O estudo utiliza o método

de Monte Carlo e de confiabilidade de primeira ordem, baseando no Método da Área Efetiva (MAE), com o objetivo de otimizar a análise, mas mantendo a confiabilidade do processo.

2.1 JUSTIFICATIVA E RELEVÂNCIA

A revisão da literatura mostrou que a análise da corrosão de dutos é essencial para a indústria de óleo e gás, sendo capaz de prevenir prejuízos e atrasos, bem como acidentes. No entanto, as normas geralmente fornecem resultados conservadores, enquanto as análises numéricas e experimentais apresentam procedimentos complexos e demorados.

A indústria apresenta uma necessidade de resultados precisos, rápidos e simplificados, com o objetivo de otimizar o processo e reduzir a necessidade de mão de obra especializada nas análises. (FERREIRA et al., 2021) demonstrou que a análise para dutos enterrados utilizando Redes Neurais Artificiais apresenta resultados satisfatórios, precisos e rápidos, mas o estudo careceu de uma análise abordando dutos submarinos. Diante disso, este Trabalho de Conclusão de Curso (TCC) do aluno tem como objetivo avaliar a capacidade das redes neurais artificiais de realizar a análise de falha de dutos submarinos corroídos, avaliando sua precisão e o tempo hábil, e sua capacidade de substituir a equação desenvolvida por (NETTO, 2009) nas análises de dutos submarinos.

3. FUNDAMENTAÇÃO TEÓRICA

A análise de falha em dutos corroídos é um processo complexo, que envolve diversos conhecimentos nas áreas de estatísticas, análise estrutural, etc. O mesmo pode ser dito para o desenvolvimento e treinamento de uma Rede Neural Artificial (RNA), onde pode ser necessário adotar meta-heurísticas, isto é, estratégias de busca.

Esta seção tem como objetivo apresentar os conceitos necessários para o compreendimento deste estudo:

3.1 Monte Carlo

O método de Monte Carlo é uma técnica estatística capaz de prever resultados baseado em estimativas probabilísticas e valores de entradas específicos, com limites bem adotados (KISSELL, 2021). É frequentemente utilizada em análises de confiabilidade afim de considerar as incertezas presentes em variáveis críticas nas análises e que podem gerar incerteza no estudo.

A metodologia consiste em gerar um número N de amostras aleatórias, onde cada variável é distribuída estatisticamente de acordo com a distribuição que melhor se adapta a ela. Para cada amostra, obtém-se um número N de soluções, assumindo que a amostra simulada é similar a uma amostra obtida experimentalmente, o que permite considerar o método como uma análise estatística (MOTTA; FERREIRA; AFONSO, 2024). O número de falhas (N_F) é contado para calcular a probabilidade de falha (P_F), que é obtida pela relação (Eq. 1).

$$P_F = \frac{N_F}{N} \qquad (Eq.\ 1)$$

A seguir, é apresentando o pseudo-código do método de Monte Carlo:

Algoritmo 1 – Pseudo-código do Método de Monte Carlo.

Hiperparâmetros: N (número de amostras), distribuição_variáveis:

Enquanto i ≤ N **faça**

Gerar amostra aleatória de variáveis

Aplicar distribuição nas amostras geradas

Calcular a solução para a distribuição de cada amostra N

Incrementar contador de falha (NF) se solução falhar

i = i+1

Fim-do-enquanto

PF = NF/N

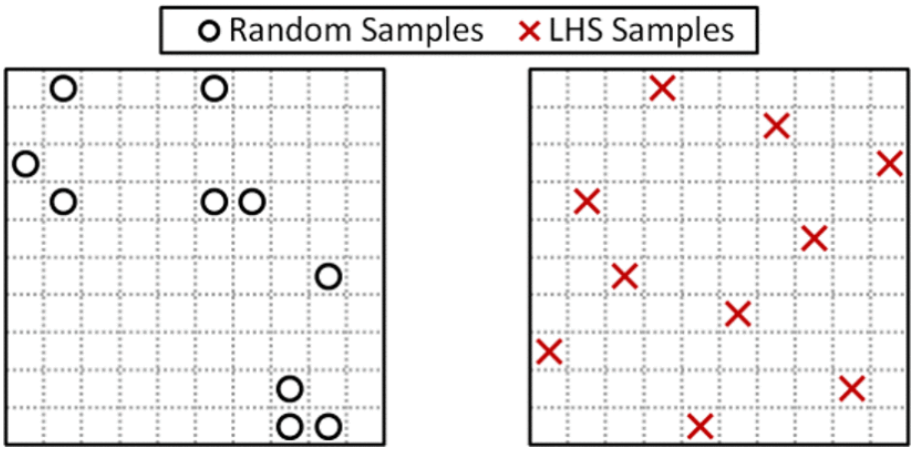
Fonte: Autor, 2024.

3.1 Latin Hypercube Sampling

O *Latin Hypercube Sampling* (LHS) é um método estatístico de amostragem amplamente utilizado em simulações computacionais para gerar variáveis aleatórias que buscam cobrir todo o espaço amostral (HADIGOL; DOOSTAN, 2018). Desenvolvido inicialmente por (MCKAY; BECKMAN; CONOVER, 1979), o LHS é uma técnica que visa melhorar a qualidade das estimativas, especialmente em situações em que a avaliações de funções complexas ou pesadas computacionalmente se faz necessária.

A metodologia do LHS baseia-se no conceito de divisão do espaço de entrada em subintervalos igualmente prováveis, garantindo que cada intervalo de cada variável seja amostrado apenas uma vez. Diferentemente da amostragem aleatória, o LHS reduz a possibilidade de clusters de pontos de amostragem, melhorando a representação da variabilidade das entradas e proporcionando resultados mais robustos e precisos. A Figura 5 a seguir apresenta uma ilustração da comparação da amostragem aleatória com a LHS.

Figura 5 - Representação visual da comparação entre a amostragem aleatória e LHS.



Fonte: (PREECE; MILANOVIC, 2016).

3.2 Formulação numérica

A análise desenvolvida pelo presente trabalho usa como base no estudo realizado por (NETTO, 2009), onde ele obteve uma equação (Eq. 2) capaz de calcular a relação entre a pressão de falha de dutos submarinos corroídos e intacto, com resultados similares aos obtidos em análise desenvolvidas utilizando EFs (SAKAKIBARA; KYRIAKIDES; CORONA, 2008).

$$\frac{\bar{P}_{COR}}{\bar{P}_{CO}} = \left[\frac{1-\frac{d}{t}}{1-\frac{d}{t}\left(1-\left(\frac{c}{\pi D}\right)^{0.4}\left(\frac{l}{10D}\right)^{0.4}\right)}\right]^{2.675} \qquad (Eq.\ 2)$$

Essa equação foi desenvolvida para atender falhas rasas, intermediárias e profundas que atendam as seguintes especificações:

- Falhas rasas
    - A relação d/t deve estar dentro da faixa 0,1 e 0,2;
    - A relação $c/\pi D$ deve ser menor ou igual a 0,1;
- Falhas intermediárias
    - A relação d/t deve ser maior que 0,2 e menor igual a 0,4;
    - A relação $c/\pi D$ deve ser menor ou igual a 0,1;
        - Nessa faixa, se se $\frac{c}{\pi D} \geq 0.15 - 0.25\frac{d}{t}$, setar $\frac{c}{\pi D} = 0.15 - 0.25\frac{d}{t}$.
- Falhas profundas
    - A relação d/t deve ser maior que 0,4 e menor igual a 0,6;
    - A relação $c/\pi D$ deve ser menor ou igual a relação $0{,}2 - 0{,}25\frac{d}{t}$;
    - Nessa faixa, se $\frac{c}{\pi D} \geq 0.1 - 0.125\frac{d}{t}$, setar $\frac{c}{\pi D} = 0.1 - 0.125\frac{d}{t}$.

Para todos os casos, (NETTO, 2009) recomenda que *l/D* deve ser fixado em 10 caso a razão ultrapasse 10, visto que a pressão de falha tende a diminuir abruptamente para esses valores. Afim de determinar a pressão de falha para dutos intactos, foi adotada a equação de Barlow (Eq. 3) que utiliza a tensão de escoamento do material (S) em seu cálculo. A pressão operacional foi definida utilizando a Maximum Allowable Working Pressure (MAWP), apresentada em (AMERICAN SOCIETY OF MECHANICAL ENGINEERS, 2019), adotando o coeficiente de segurança de 72% (Eq. 4).

$$\bar{P}_{CO} = 2 * S * \frac{t}{D} \qquad (Eq.\ 3)$$

$$\bar{P}_{OP} = 0{,}72 * \bar{P}_{CO} \qquad (Eq.\ 4)$$

3.3 Redes Neurais Artificiais

As RNAs são um modelo de machine learning inspirados no funcionamento do cérebro humano e no seu processo de aprendizado (HAYKIN, 2001). Elas são compostas por camadas de neurônios artificiais interconectados, onde cada neurônio recebe entradas, as processa e gera saídas que alimentam a próxima camada, formando uma estrutura densa e hierárquica.

Um dos modelos mais simples de RNA é o *perceptron*, que consiste em uma única camada (neurônio simples) e é capaz de resolver problemas de classificação binária. Cada entrada (x_1, x_2, x_3, ..., x_n) é ponderada por pesos (w_1, w_2, w_3, ..., w_n) que determinam a sua importância relativa para a geração da saída (ROSENBLATT, 1958), como exibido em (Eq. 5).

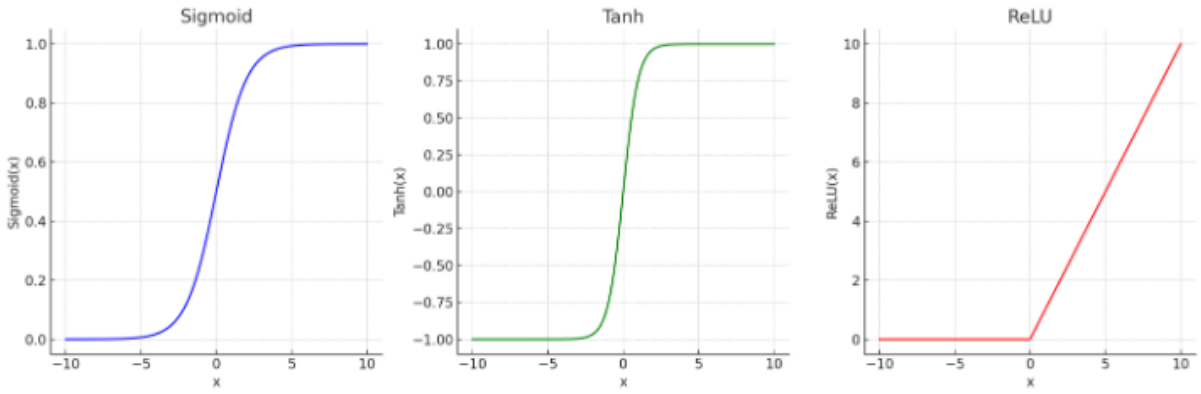
$$\sum w_{kj} * x_j + b_k \qquad (Eq.\ 5)$$

Existem três tipos de camadas adotadas numa RNA: a camada de entrada, a camada oculta e a camada de saída. A camada de entrada recebe os dados principais (features) necessários para os cálculos das saídas. A camada oculta realiza o processamento dos dados, através de uma soma ponderada linear das entradas e aplica as funções de ativação para transformar os valores antes de enviá-los para a próxima camada. Para redes com várias camadas ocultas, a RNA é classificada como uma rede profunda (*deep neural network*) (GOODFELLOW; BENGIO; COURVILLE, 2016).

As funções de ativação são aplicadas nas saídas de cada neurônio e introduzem não-linearidades na RNA, permitindo que a rede aprenda e capture padrões mais complexos. Essas funções controlam a importância de cada neurônio na rede, determinando se ele deve ser "ativado" ou não, ou seja, se sua saída será considerada no processo de decisão (HAYKIN, 1999). Geralmente, as camadas ocultas em uma RNA utilizam a mesma função de ativação, enquanto a função de ativação final pode variar de acordo com o tipo de problema sendo resolvido. A escolha de ativação depende do objetivo da RNA, mas as três mais utilizadas são: Sigmoid, Tanh e ReLU.

A função Sigmoid converte valores entre 0 e 1, sendo útil em problemas binários, embora possa sofrer com o problema de gradientes "desvanecentes", dificultando o aprendizado em redes profundas (LECUN et al., 1998). A função Tanh mapeia entradas para um intervalo entre -1 e 1, tornando-se mais robusta que a Sigmoid, pois permite que os neurônios tenham valores negativos, o que amplia a gama de representações (LECUN et al., 1998). Já a ReLU (*Rectified Linear Unit*) é a mais utilizada em redes profundas por sua simplicidade e eficiência, pois não sofre tanto com o problema de gradientes "desvanecentes" e facilita o treinamento de redes mais complexas (GLOROT et al., 2011). A Figura 6 apresenta uma representação gráfica de cada função de ativação.

Figura 6 - Representação visual das funções de ativação: Sigmoid, Tanh e ReLU



Fonte: Autor, 2024.

Os pesos e vieses adotados na análise são calculados por um processo chamado de retropropagação, que consiste em propagar o erro de predição, podendo ser dividido em duas etapas (BRANDÃO, 2023). Na primeira etapa, conhecida como propagação direta, os dados de entradas são alimentados na RNA, sendo processados através de conexões ponderadas por pesos fixos (HAYKIN,

1999). Na segunda etapa, os pesos são ajustados a partir de uma equação de minimização do erro, onde o valor calculado é subtraído do desejado para obter o erro (RUMELHART; HINTON; WILLIAMS, 1986). Esse erro é minimizado consecutivamente por um algoritmo de otimização (e.g Adam, Softmax, Sigmoid) para ajustar os pesos e otimizar o desempenho da rede (BRANDÃO, 2023).

Além dos pesos, outros parâmetros devem ser considerados no uso de RNA, como o número de épocas e o tamanho do lote de dados. As épocas referem-se ao número de ciclos que o conjunto de dados de treinamento é repetidamente apresentado à RNA, representando o número de iterações na quais a rede é treinada (HAYKIN, 1999). Já o tamanho do lote refere-se à quantidade de amostras que são utilizadas a cada atualização dos pesos durante uma época. Esse fator ajuda a definir a qualidade e o peso computacional da análise, com tamanhos menores apresentando maior qualidade, porém maior peso computacional. Para os casos onde o tamanho do lote é considerado 1, chamamos de gradiente estocásticos e a cada amostra, os pesos são atualizados (GOODFELLOW et al., 2016).

3.4 Greedy Randomized Adaptive Search Procedure

A meta-heurística Greedy Randomized Adaptive Search Procedure (GRASP) é uma técnica capaz de gerar soluções que escapam de ótimos locais após o procedimento da busca local (SILVA, 2024). Partindo de uma solução inicial vazia, o algoritmo escolhe iterativamente um elemento para compor sua solução, formando o que é chamado de Lista Restrita de Candidatos (LRC). O GRASP é conhecido por ser um algoritmo guloso, pois, a cada passo realizado, ele escolhe o melhor candidato de acordo com uma função de avaliação. Essa característica permite que o algoritmo encontre boas soluções em um tempo relativamente curto, embora nem sempre garanta uma solução ótima global (RESENDE et al., 2016).

O GRASP utiliza dois hiperparâmetros na sua análise: o número de iterações e α que controla a cardinalidade da LRC. O parâmetro α varia entre 0 e 1, com zero representando uma busca aleatória e 1 representando uma busca gulosa, isso é, escolhe apenas os melhores valores para compor a LRC. A calibração do parâmetro α pode ser realizada utilizado o método do GRASP reativo, onde através das formulações a seguir, é possível redefinir os valores do parâmetro (SILVA, 2024).

$$q_i = \left(\frac{f^*}{A_i}\right)^{\delta} \qquad (Eq.\ 6)$$

$$p_i = \frac{q_i}{\sum_{j=1}^{j=k} q_j} \qquad (Eq.\ 7)$$

Onde *f** representa a melhor solução obtida até o momento, δ é um parâmetro de amplificação e A_i é a média das soluções. A probabilidade p_i passa a ser o novo valor do parâmetro α. O algoritmo

2 apresenta a estrutura do GRASP, onde a fase de construção corresponde à criação da LRC composta por elementos selecionados aleatoriamente. Em seguida, o algoritmo de melhoria aplica uma busca local sobre esses resultados iniciais, com o objetivo de refinar e aprimorar a melhor solução encontrada.

Algoritmo 2 – Pseudo-código do GRASP Padrão.

Hiperparâmetros: α, número_iterações:
Enquanto i ≤ número_iterações **faça**
Fase de construção
Algoritmo de melhoria
Atualiza melhor solução encontrada
i = i+1
Fim-do-enquanto

Fonte: Adaptado de (SILVA, 2024).

3.5 Particle Swarm Optimization

A Particle Swarm Operation (PSO) é uma meta-heurística baseada no comportamento social do coletivo de animais (e.g., bando de passáros, cardume de peixes) que trabalham em conjunto na busca por uma área ideal que atinja seus objetivos (BRANDÃO, 2023). A técnica é utilizada em problemas de otimização contínua e combina elementos de exploração no espaço de soluções, permitindo que o enxame encontre ótimos globais ou locais com eficiência (KENNEDY et al., 1995).

O algoritmo PSO modela um conjunto de partículas (potenciais soluções) que "voam" pelo espaço amostral, ajustando suas posições com base em duas informações principais: a melhor solução já encontrada por cada partícula individual (memória individual) e a melhor solução descoberta pelo conjunto inteiro de partículas (memória global). Essa dinâmica de cooperação e competição entre as partículas permite que elas explorem diversas regiões do espaço de soluções ao longo das iterações (CLERC; KENNEDY, 2002).

No PSO, cada partícula *i* representa uma solução candidata para o problema de otimização, sendo representado por três vetores: sua posição atual x_i, a melhor posição encontrada pela partícula até o momento p_i e a velocidade v_i associada a ela. A posição representa à solução atual da partícula, enquanto a velocidade define o quanto e em qual direção essa partícula irá mover no espaço de soluções. O processo de buscar é governado pelas seguintes equações:

$$v_i(t+1) = v_i + c_1 r_1 \big(p_i - x_i(t)\big) + c_2 r_2 \big(g - x_i(t)\big) \qquad (Eq.\ 8)$$

$$x_i(t+1) = x_i(t) + v_i(t+1) \qquad (Eq.\ 9)$$

Onde g é a melhor posição global encontrada pelo enxame; c1 e c2 são coeficientes de aceleração que determinam a importância da exploração individual e global, também chamados respectivamente de parâmetro cognitivo e social; e r1 e r2 são variáveis aleatórias distribuídas uniformemente no intervalo [0, 1], que introduzem um componente estocástico no movimento das partículas (TRELEA, 2003).

Há duas variações principais do PSO, que se distinguem pela forma como as partículas compartilham informações: o gbest (global best) e o lbest (local best). Na variação gbest, todas as partículas do enxame têm acesso à melhor solução encontrada globalmente pelo enxame inteiro, o que significa que cada partícula é influenciada pela mesma solução global g. Esse método é geralmente mais rápido para convergir, mas pode ser mais propenso a convergência prematura em mínimos locais devido à falta de diversidade nas soluções (ENGELBRECHT, 2005). Já na variação lbest, cada partícula só se comunica com um subconjunto de partículas vizinhas. Isso significa que cada partícula é influencia pela melhor solução encontrada dentro do seu grupo local, em vez da melhor solução global. Essa abordagem introduz mais diversidades no enxame e pode ajudar a evitar a convergência prematura, embora a convergência global possa ser mais lenta (CLERC; KENNEDY, 2002).

É preciso ter cuidado ao adotar a (Eq. 8), visto que a depender dos valores assumidos para r_1 e r_2, a velocidade e a posição da partícula podem aumentar rapidamente de modo que tenda a infinito. Esse cenário resultaria em partículas que fogem do espaço de busca e não atendem às restrições do problema (BRANDÃO, 2023). Afim de ter um controle da explosão de velocidade, pode ser adotados os parâmetros como o fator de inércia (w) e o fator de constrição (χ).

O fator de inércia varia entre 0 e 1 e tem como objetivo ajustar a influência da velocidade anterior da partícula na velocidade atual, onde o valor 0 representa que não há influência da velocidade anterior e a partícula adota um comportamento mais exploratório, sem considerar a direção previamente calculada enquanto o valor 1 representa uma análise fortemente influenciada pela sua velocidade anterior, resultando em uma busca maior em uma região específica e evitando cenários de explosão de velocidade, onde a partícula se afasta demais do espaço de busca no momento (t+1). Na equação abaixo, pode-se ver a influência do fator de inércia na análise.

$$v_i(t+1) = wv_i + c_1 r_1\big(p_i - x_i(t)\big) + c_2 r_2\big(g - x_i(t)\big) \qquad (Eq.\ 10)$$

Já o fator de constrição é um parâmetro introduzido para garantir a estabilidade e a convergência do algoritmo ao longo das iterações, restringindo a magnitude das atualizações de velocidades das partículas. Esse fator é calculado considerando c1 e c2, adotando a formulação a seguir.

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \quad (Eq.\ 11)$$

$$\varphi = c_1 + c_2 \quad (Eq.\ 12)$$

O algoritmo lógico do PSO é representado a seguir:

Algoritmo 3– Etapas de atualização do PSO

Para cada etapa no tempo t **faça**

Para cada partícula *i* dentro do enxame faça

Atualize a posição x_i (t) usando as Eq. 9 e 10

Calcule o desempenho da partícula f(x_i(t))

Atualize p_i, g

Fim

Fim

Fonte: Adaptado de (BRANDÃO, 2023).

3.6 OPTUNA

O Optuna é uma biblioteca de otimização de hiperparâmetros implementada na linguagem de programação *Python*, projetada para ser eficiente, flexível e fácil de usar. Ele adota duas abordagens no seu funcionamento: *search space sampling* e *pruning*. O *search space sampling* diz respeito ao processo de exploração do espaço de busca de hiperparâmetros, isso é, o conjunto de todas as combinações possíveis de valores para os hiperparâmetros.

De forma a diminuir a capacidade computacional e otimizar a análise, o Optuna escolhe as combinações de maneira a explorar as áreas mais promissores do espaço de busca, utilizando o método Tree-structured Parzen Estimator (TPE) que desenvolve uma distribuição de probabilidade condicional para avaliar quais áreas do espaço de busca contém boas soluções (AKIBA et al., 2019). Para isso, o TPE define um limiar (τ) em relação ao MSE e divide o conjunto de amostras em dois subconjuntos apresentados a seguir.

$$l(x) = P(x|f(x) < \tau) \quad (Eq.\ 13)$$

$$g(x) = P(x|f(x) \geq \tau) \quad (Eq.\ 14)$$

O algoritmo tenta selecionar amostras x que maximizem a relação $\frac{l(x)}{g(x)}$, favorecendo valores que são mais favoráveis de melhorar a qualidade da RNA.

Já o *pruning* diz respeito a uma técnica que interrompe tentativas que, após um certo número de iterações, não apresenta resultados promissores. A técnica compara a média do desempenho de

um experimento (M_t) até o momento t, com a média de desempenho de outros já executados ($f(x_i)$), e se o desempenho estiver abaixo do esperado, essa execução é interrompida, evitando o desperdício de recursos computacionais em treinos não promissores (BERNBARDT et al., 2020). A formulação que descreve o pruning é apresentada a seguir.

$$M_t = \frac{1}{t}\sum_{i=1}^{t} f(x_i) \qquad (Eq.\ 15)$$

O Optuna se diferencia por introduzir o conceito de "estudo" (*study*), que envolve a criação de experimentos com definições claras de objetivos a serem minimizados ou maximizados. Cada estudo consiste em várias "tentativas" (*trials*), onde uma tentativa representa uma combinação específica de hiperparâmetros testados (AKIBA et al., 2019). Seu algoritmo é apresentado a seguir:

Algoritmo 4 – Etapas de atualização do Optuna

Para cada teste (trial) **faça**

Selecione os hiperparâmetros usando o TPE pela Eqs. 13 e 14

Treine o modelo com os hiperparâmetros selecionados

Atualize o pruning pela Eq. 15

Atualize o melhor desempenho obtido

Fim

Fonte: Autor, 2024.

4. Metodologia

No presente trabalho, foi desenvolvido uma RNA para análise da pressão de falha para dutos submarinos corroídos, bem como seu banco de dados para treinamento. Foram adotadas 4 metodologias no desenvolvimento da RNA: sem otimização de parâmetros e com a otimização adotando as técnicas de GRASP, PSO e OPTUNA.

Nesta seção, cada uma dessas abordagens será detalhada, bem como a a metodologia adotada para a comparação dos resultados.

4.1 Dados para treinamento

Os dados para treinamento da RNA foram desenvolvidos pelo autor por meio de um código computacional construído na linguagem *Python*. Utilizando a técnica estatística LHS de amostragem, foram gerados casos aleatórios para as variáveis adotadas na equação semiempírica desenvolvida por (NETTO, 2009), estabelecendo limites mínimos e máximos considerando os tipos de defeitos especificados em (NETTO, 2009). Essa metodologia foi escolhida para garantir a geração de casos aleatórios de forma que não haja concentração excessiva em um espaço amostral específico e que todos os tipos de defeitos apresentem a mesma quantidade de amostras.

As variáveis adotadas foram as relações d/t; $C/\pi D$ e $l/10D$, utilizadas para obter a relação das pressões (Eq. 2), e a relação t/D, empregada no cálculo da pressão intacta (Eq.3). Optou-se por usar essas relações em vez das variáveis individuais, pois estas permitem uma abordagem mais ampla no estudo, avaliando dutos com diferentes tamanhos e configurações de defeitos. A Tabela 1 a seguir, apresenta os valores mínimos e máximos definidos para cada variável gerada pelo LHS, que foram definidos com base em valores usualmente adotados em análises na área de dutos (GONG et al., 2020) para as relações. Importante notar que os limites para t/D foram considerados os mesmos para todos os defeitos, visto que dizem respeito a geometria do duto.

Tabela 1 - Limites mínimo e máximo das relações.

Variáveis	**Defeitos Rasos**		**Defeitos Intermediários**		**Defeitos Profundos**	
	Mínimo	**Máximo**	**Mínimo**	**Máximo**	**Mínimo**	**Máximo**
d/t	0,1	0,2	0,2	0,4	0,4	0,6
c/πD	0,01	0,1	0,03	0,1	0,05	0,1
l/10D	0,1	0,5	0,3	0,7	0,5	1
t/D	0,02	0,1	0,02	0,1	0,02	0,1

Fonte: Autor, 2024.

Com as variáveis geradas, foi realizado um filtro para garantir que apenas os dados que se adequassem às limitações apresentadas na formulação de (NETTO, 2009) fossem aceitos,

descartando os demais. Nessa etapa, foram gerados 10.000 casos de cada tipo de defeito, dos quais todos foram aceitos no filtro, totalizando 30.000 casos para treinamento da RNA.

A partir desses dados, foi calculada a relação entre a pressão de falha para dutos corroídos e dutos intactos, com base na formulação de (NETTO, 2009). Utilizando a equação de Barlow, foi possível obter a pressão de falha para o duto intacto, que é usada em conjunto com a relação calculada para determinar a pressão de falha para o duto corroído, conforme apresentado na (Eq. 16). É importante pontuar que a tensão de escoamento (S) adotada foi de 490 Mpa, visto que está na faixa para aços normalmente adotados na construção de dutos submarinos, como o API 5L X70 (

$$\bar{P}_{COR} = \bar{P}_{CO} * \frac{\bar{P}_{COR}}{\bar{P}_{CO}} \qquad (Eq.\ 16)$$

Com os resultados obtidos, foi criado um banco de dados com os dados obtidos das amostras e as suas respectivas pressões de falha para duto corroído. O banco de dados foi gerado no formato pickle (.pkl) que é um formato de compressão específico para linguagem Python. Esse banco de dados será adotado posteriormente para treinar a RNA desenvolvida, onde 70% será utilizado para treino e 30% utilizado para teste da RNA.

4.2 RNA Básica

Uma das abordagens adotada foi do treinamento da RNA sem utilizar meta-heurística para otimizar os hiperparâmetros da rede, com o objetivo de comparar com os outros resultados utilizando técnicas meta-heurísticas. Essa RNA será referida como RNA Básica durante o trabalho.

Essa análise adotou duas camadas ocultas, onde a primeira possui 64 neurônios e a segunda possui 32 neurônios, e ambas adotam a função de ativação Relu. A função de otimização dos pesos adotada foi a Adam, enquanto o número de épocas e tamanho do lote, foram respectivamente 50 e 32.

A função de erro utilizada para avaliar a qualidade da RNA é o Erro Quadrático Médio (EQM), visto que é o mais adotado nas análises de RNA (HAYKIN, 2001). Ao final do treinamento, essa RNA e as outras foram salvas na extensão "*.keras*", que é o formato nativo recomendado para a biblioteca *keras* do *Python*. É importante notar que alguns parâmetros como número de épocas, tamanho do lote, função de otimização e função de erro será adotado para as outras análises também, de modo que a comparação dos resultados se torne mais coerente.

4.3 RNA + GRASP

Nesta etapa, será explicada a implementação da meta-heurística GRASP na otimização da RNA. Foram adotadas as funções de ativação Sigmoid, Tanh e Relu para essa análise e para as

próximas. O número de camadas varia entre 1 e 5, enquanto o número de neurônios por camada varia entre 5 e 128. Esses parâmetros serão replicados nas próximas análises com utilização de meta-heurísticas.

O número de iterações escolhido foram 10, por tentativa e erro, visto que atingia uma análise satisfatória, e este é o único critério de parada da otimização. O parâmetro α é escolhido aleatoriamente entre a faixa de valores 0,1; 0,3; 0,5; 0,7 e ajustado para cada iteração com sua performance considerada. O valor da variável δ foi definido como 10, visto que é o mais utilizado para acelerar respostas (GASPAR CUNHA; TAKAHASHI; HENGGELER ANTUNES, 2013). O algoritmo deve retornar a melhor solução encontrada nessa faixa de iterações.

4.4 RNA + PSO

Nesta etapa, para aplicar a meta-heurística PSO na otimização da RNA, foi utilizada a biblioteca *pyswarms* da linguagem computacional *Python*, que implementa a técnica de forma simples. Para o estudo, foi escolhido a abordagem utilizando gbest, adotando as funções de ativação ReLU (0), Tanh (1) e Sigmoid (2) com os respectivos números inteiros associados a elas.

A faixa de otimização do número de camadas e neurônios foi a mesma adotada na análise RNA + GRASP. Nessa análise, foram consideradas 10 iterações e tamanho do enxame de 100, visto que por tentativa e erro, esses valores apresentaram os melhores resultados com menor carga computacional.

Os parâmetros cognitivo e social, utilizando para regular a influência dos resultados individuais e coletivos, foram considerados igual à 2,05, segundo os valores padrões propostos por (BRATTON; KENNEDY, 2007). Já o peso da inércia (w) adotado foi de 0,5, com o objetivo de reduzir a carga computacional do código, visto que o código consegue convergir para um resultado adequado com este parâmetro. O algoritmo deve retornar a melhor solução encontrada nessa faixa de iterações.

4.5 RNA + Optuna

A biblioteca Optuna também foi implementada para otimizar os seguintes parâmetros da RNA: número de neurônios, número de camadas e função de ativação. A faixa de valores desses parâmetros é a mesma adotada para as outras análises, bem como seu número de épocas e tamanho do lote.

Foi considerado, após tentativa e erro, um tamanho de teste (*trials*) de 10, visto que a RNA converge nessa faixa com um EQM adequado, sendo este o único parâmetro específico do algoritmo e o único critério de parada. Ao final dos *trials*, será retornado a melhor solução encontrada.

4.6 Comparação entre resultados

Após o treinamento das RNAs, foi realizada a comparação dos resultados obtidos pelos modelos RNAs em função da equação desenvolvida por (NETTO, 2009), em que o trabalho foi baseado. Nesta etapa, foi utilizado a técnica LHS novamente para gerar as amostras aleatórias usadas na comparação dos resultados obtidos pelos modelos RNA e pela formulação, adotando os limites apresentados na Tabela 2.

Tabela 2 - Variáveis para comparação.

Variáveis	**Amostras**	
	Mínimo	**Máximo**
d/t	0,1	0,6
c/πD	0,01	0,1
l/10D	0,1	1
t/D	0,02	0,1

Fonte: Autor, 2024.

Essas amostras serão filtradas, de maneira similar a forma que foram filtrados na geração do banco de dados. Após o filtro, essas amostras serão usadas para calcular a pressão de falha para dutos submarinos corroídos de duas maneiras: através dos modelos RNAs gerados e através da formulação desenvolvida por (NETTO, 2009) e a (Eq. 13).

A comparação entre as pressões de falhas obtidas pelos modelos ($\bar{P}_{COM}$) e obtida pela formulação ($\bar{P}_{COF}$) será realizada de modo a obter o Erro Percentual Médio Absoluto (EPMA) apresentado em (Eq. 17).

$$EPMA = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{\bar{P}_{COF} - \bar{P}_{COM}}{\bar{P}_{COF}}\right| x100 \qquad (Eq.\ 17)$$

4.7 Análise Probabilística

A última análise realizada é a análise probabilística realizada utilizando o método de Monte Carlo, onde alguns dos parâmetros de entrada serão variados com o objetivo de avaliar como a pressão de falha se comporta diante dessas variações. Foi selecionada uma amostra que será avaliada na análise, em que cada um dos seus parâmetros será variado em 100.000 amostras variadas distintas.

O critério de escolha da amostra levou em consideração a diferença entre a pressão operacional e a pressão de falha, de modo que esta não ultrapasse 10%, utilizando a (Eq. 2)

desenvolvida por (NETTO, 2009) como base, evitando assim selecionar uma amostra onde todas as amostras variáveis falhem ou sejam aprovadas. Cada parâmetro será variado seguindo uma distribuição adotada em (DE SIQUEIRA MOTTA; MARQUES FERREIRA; AFONSO, 2024), conforme especificado na Tabela 3 a seguir.

Tabela 3 - Distribuições adotadas para parâmetros.

Parâmetro	**Distribuição Adotada**	**Desvio Padrão em relação à média**
D	Normal	3%
t	Log-normal	5%
l	Normal	5%
d	Normal	5%
c	Normal	5%
Pressão Operacional	Gumbel	3%

Adaptado de: (DE SIQUEIRA MOTTA; MARQUES FERREIRA; AFONSO, 2024).

É importante lembrar que as variáveis adotadas foram as relações d/t; $C/\pi D$, $l/10D$ e t/D. Portanto, para aplicar as distribuições, foi necessário calcular *t*, *d*, *C* e *l* por meio das relações apresentadas abaixo. O valor de D adotado foi baseado nos valores do diâmetro adotado no trabalho desenvolvido por (SAKAKIBARA; KYRIAKIDES; CORONA, 2008) com dutos experimentais, que realiza a análise com dutos que variam na faixa de 2 polegadas. Visto que a distribuição na Tabela 3 considera o uso da SI, D foi convertido para 0,0508 metros. Após a distribuição e geração das amostras variáveis, foram realizados os cálculos das relações ($\frac{d}{t}$, $\frac{C}{\pi * D}$, $\frac{l}{10 * D}$ e $\frac{t}{D}$) para que pudessem ser aplicados na formulação e nos modelos RNAs como entradas.

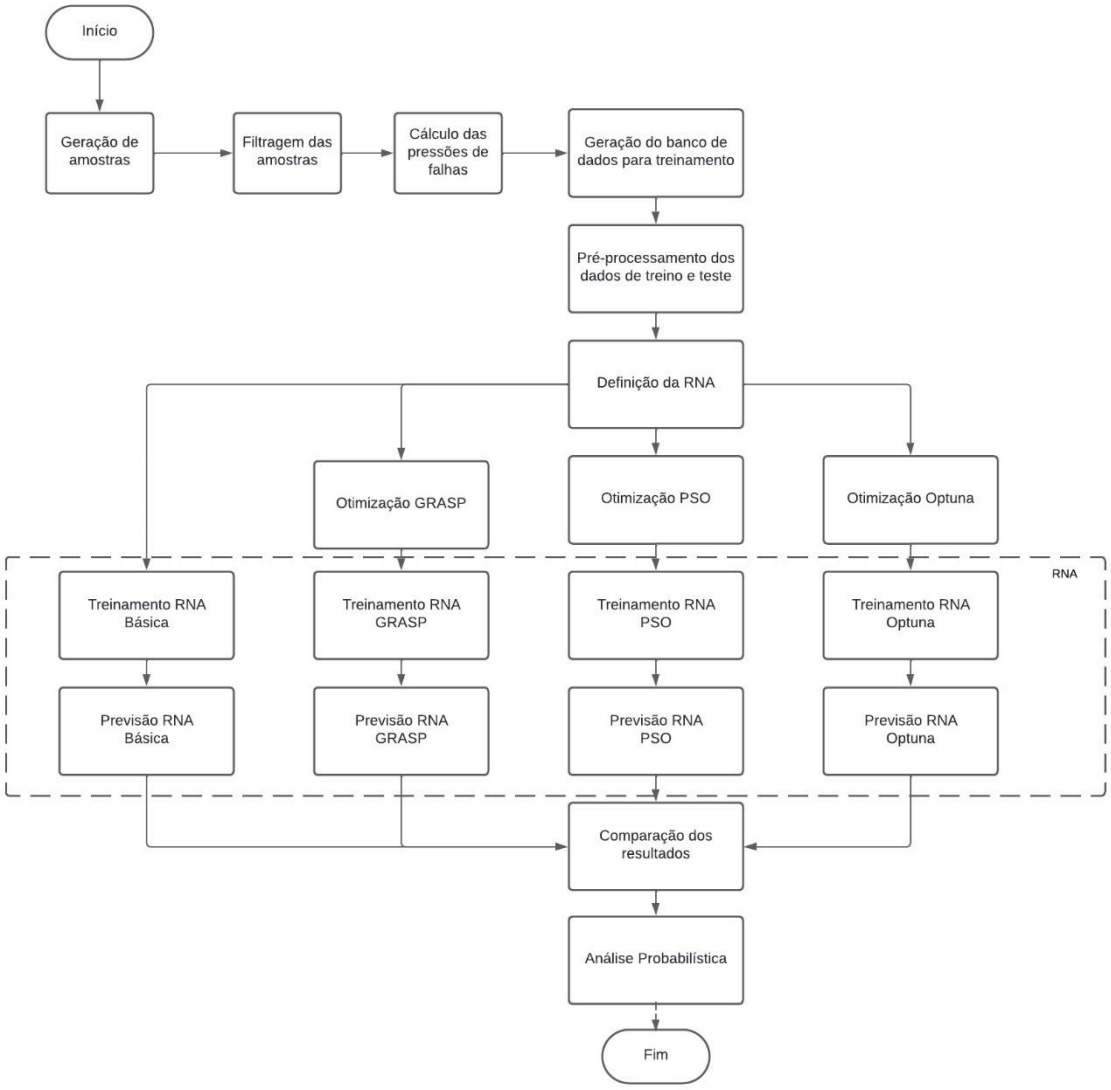
Após a aplicação das entradas e a obtenção das pressões de falhas para cada amostra variada, foi avaliado se cada caso atende ao critério de falha, que consiste em verificar se a pressão de falha calculada ultrapassa a pressão operacional. A relação entre o total de amostras (N) e o total de amostras que falharam (N_F) foi computada com o objetivo de calcular a probabilidade de falha (P_F) utilizando a (Eq. 1).

4.8 Google Colaboratory

O código foi desenvolvido, adotando os procedimentos descritos anteriormente, na linguagem de programação *Python (PYTHON SOFTWARE FOUNDATION, 2024)* e no ambiente de programação Google Colaboratory. O Collab é um serviço de computação em nuvem, baseado em notebooks *Jupyter*, que oferece ao usuário acesso a recursos computacionais avançados, como unidades de processamento gráficos (GPUs) (COLAB, 2024).

Além disso, a plataforma oferece um compartilhamento rápido com outros usuários e permite a organização do código em células, afim de compartimentalizar e otimizar o desempenho computacional do código. A Figura 7 a seguir, apresenta o fluxograma do código desenvolvido.

Figura 7 – Fluxograma do código desenvolvido.



Fonte: Autor, 2024.

5 RESULTADOS

Nesta seção, serão apresentados os resultados obtidos em cada RNA, bem como a comparação dos resultados obtidos.

5.1 Simulação RNA

A simulação foi realizada para cada RNA treinada adotando os parâmetros descritos anteriormente. Os critérios avaliados foram a melhor EQM obtida no treinamento de cada método, o seu tempo de treinamento em segundos e a função de ativação usada. Vale notar que para o método básico, a função de ativação escolhida foi ReLU, pela sua maior eficiência computacional e ser melhor adequada para redes neurais profundas, em comparação a Sigmoid a TanH. É importante pontuar que ao usar o Google Colab, é possível utilizar GPUs paralelas, o que reduz significativamente o tempo de treino de modelos. A Tabela 4 a seguir apresenta os resultados obtidos.

Tabela 4 - Resultados obtidos para RNAs.

Método	**EQM**	**Tempo Treino (s)**	**Função de ativação**
Básico	5,7381	69,28	reLU
PSO	0,0164	816,84	reLU
GRASP	0,1116	773,39	reLU
Optuna	0,0404	744,13	reLU

Fonte: Autor, 2024.

Para a comparação da execução dos modelos, foram gerados 10.000 casos para a comparação entre os resultados das análises dos modelos e da equação. Os parâmetros avaliados são: O Erro Percentual Médio Absoluto entre os resultados obtidos pelos modelos e pela equação (EPMA) e a probabilidade de falha da amostra calculado pelo método de Monte Carlo (P_F). O comparativo dos resultados obtido é apresentado na Tabela 5 a seguir.

Tabela 5 - Comparativo dos resultados obtidos pelos modelos e pela equação.

Método	**EPMA**	**P_F**	**Diferença P_F**
Básico	9,26%	99,82%	36,55%
PSO	3,19%	67,15%	3,88%
GRASP	2,19%	66,71%	3,44%
Optuna	3,69%	57,69%	5,58%
Equação	–	63,27%	–

Fonte: Autor, 2024.

É possível notar que na Tabela 4, a EQM se comporta de maneira similar para todos os casos otimizados, porém é consideravelmente maior para o método básico, como era esperado visto a falta de otimização. Na Tabela 5, notamos que existem certa variação na EPMA e na P_F, porém todos apresentarem bons resultados, com a EPMA menor que 4% e a diferença de P_F menor que 6%.

Com o objetivo de verificar como alguns parâmetros influenciam na qualidade das RNAs, foi realizada a análise de outros casos, adotando novos parâmetros nos treinamentos das RNAs. Os parâmetros variados para esses casos são apresentados na Tabela 6 a seguir.

Tabela 6 - Parâmetros adotados para os casos.

Caso	Número máximo de Neurônios	Épocas	Iterações/*Trials*
Caso original	128	50	10
Caso 1	128	30	10
Caso 2	128	50	5
Caso 3	64	50	10

Fonte: Autor, 2024.

5.1.1 Primeiro Caso (C01): Interações e parâmetros originais e 30 épocas

Nesse caso, foi adotado menos épocas que o problema original, adotando um valor de 30 épocas, porém foi mantido os parâmetros e número de iterações/*trials* originais. O objetivo é avaliar a influência das épocas na qualidade das análises e do seu tempo computacional. Os resultados obtidos são apresentados nas Tabelas 7 e 8 a seguir.

Tabela 7 – Resultados obtidos para RNAs para o Caso 1.

Método	EQM	Tempo Treino (s)	Função de ativação
Básico	8,4227	51,71	reLU
PSO	0,0294	491,90	reLU
GRASP	0,1146	482,73	reLU
Optuna	0,4453	519,84	tanH

Fonte: Autor, 2024.

Tabela 8 – Comparativo dos resultados obtidos pelos modelos e pela formulação para o Caso 1.

Método	EPMA	P_F	Diferença P_F
Básico	10,61%	78,50%	24,30%
PSO	2,76%	59,96%	5,76%
GRASP	2,96%	54,55%	0,35%
Optuna	4,91%	47,43%	6,77%
Equação	–	54,20%	–

Fonte: Autor, 2024.

Observa-se que a redução no número de épocas influencia positivamente o tempo computacional da análise. No entanto, também impacta negativamente a qualidade da análise, sendo mais notável no modelo básico que apresentou EQM e EPMA maiores que no caso original. Para os casos com otimização, porém, apesar de haver perca na qualidade, os resultados ainda são precisos, com EPMA menor que 5% e diferença P_F menor que 7% para todos os métodos.

5.1.2 Segundo Caso (C02): Parâmetros e épocas originais e metade das interações

Os modelos com otimização adotam um parâmetro denominado de: iterações (para PSO e GRASP) ou *trials* (para Optuna) em suas análises, representando o número de execuções do algoritmo para encontrar o melhor resultado. Neste caso, será considerada a metade dessas iterações, i.e., 5 iterações para os métodos de GRASP e PSO, e 5 *trials* para o método Optuna. Os demais parâmetros, bem como o número de épocas, serão mantidos iguais ao caso original. O modelo básico não será incluído nesta análise, uma vez que nenhum parâmetro foi alterado para ele.

A Tabela 9 e 10 apresentam os resultados obtidos a seguir.

Tabela 9 – Resultados obtidos para RNAs para o Caso 2.

Método	**EQM**	**Tempo Treino (s)**	**Função de ativação**
PSO	0,0121	393,03	reLU
GRASP	0,0424	362,10	reLU
Optuna	0,0231	338,41	reLU

Fonte: Autor, 2024.

Tabela 10 – Comparativo dos resultados obtidos pelos modelos e pela formulação para o Caso 2.

Método	**EPMA**	**P_F**	**Diferença P_F**
PSO	2,21%	26,06%	3,50%
GRASP	1,98%	24,71%	2,16%
Optuna	1,58%	21,38%	1,18%
Equação	–	22,55%	–

Fonte: Autor, 2024.

Neste caso, é possível notar que os resultados apresentaram comportamento bem similar ao do caso original para EQM, EPMA e Diferença P_F, porém com menor tempo computacional. Isso indica que as RNAs conseguem convergir em um resultado satisfatório adotando menos iterações/*trials*, mas é preciso tomar cuidado na escolha deste parâmetro, visto que para outros cenários, é possível que a mudança no parâmetro pode afetar a qualidade da RNA.

5.1.3 Terceiro Caso (C03): Iterações e épocas originais e alteração na faixa de neurônios

No desenvolvimento das RNAs com otimização, um dos parâmetros a ser ajustado está no número de neurônios, em que foi definido uma faixa entre 5 a 128. Como o objetivo da análise é minimizar o EQM, pode-se acabar utilizando um valor próximo ao limite superior, o que resultaria em uma maior carga computacional. Diante disso, será avaliada a qualidade das RNAs utilizando

uma faixa com valor máximo reduzido. Para este estudo, a nova faixa considerada será entre 5 e 64 neurônios. Os resultados obtidos são apresentados na Tabela 11 e 12 a seguir.

Tabela 11 – Resultados obtidos para RNAs para o Caso 3.

Método	**EQM**	**Tempo Treino (s)**	**Função de ativação**
PSO	0,2953	744,27	reLU
GRASP	0,0966	707,14	tanH
Optuna	0,0242	674,60	reLU

Fonte: Autor, 2024.

Tabela 12 – Comparativo dos resultados obtidos pelos modelos e pela formulação para o Caso 3.

Método	**EPMA**	**P_F**	**Diferença P_F**
PSO	4,07%	19,90%	8,28%
GRASP	2,95%	9,96%	1,66%
Optuna	2,37%	10,70%	0,92%
Equação	–	11,62%	–

Fonte: Autor, 2024.

É possível notar que em relação ao tempo computacional, houve uma leve redução comparado ao caso original. Quanto aos parâmetros que dizem respeito a qualidade da RNA (EQM, EPMA e Diferença P_F), os resultados flutuam para cada método, onde o GRASP e Optuna apresentaram resultados similares ao caso original, enquanto o PSO apresentou resultado com EPMA e diferença de P_F bem maior. Isso pode indicar que adotar uma faixa menor de número máximo de neurônios pode afetar a qualidade da análise, a depender do método de otimização adotado.

5.2 Análise Comparativa

Nessa etapa, será realizada a comparação dos resultados obtidos dos casos a fim de traçar conclusões a respeito dos resultados de cada método. Para isso, foi calculado a média geral dos parâmetros avaliados para todos os casos, que são apresentados na Tabela 13.

Tabela 13 – Média geral dos parâmetros para todos os casos.

Método	**EQM**	**Tempo Treino (s)**	**EPMA**	**Diferença P_F**
Básico	7,0804	60,50	9,94%	30,42%
PSO	0,0883	611,51	3,06%	5,36%
GRASP	0,0913	581,34	2,52%	1,90%
Optuna	0,1332	569,24	3,14%	3,61%

Fonte: Autor, 2024.

A partir das médias calculadas, é possível observar o comportamento de cada método e levantar as seguintes observações:

- O método básico apresentou o menor tempo de execução, porém os maiores valores de EQM, EPMA e diferença de $P_{F.}$ Esse resultado era esperado, já que a ausência de otimização compromete a precisão, mas acelera a análise.
- O RNA otimizado com PSO demonstrou ser o método mais lento, o que era esperado devido à complexidade inerente do algoritmo. Embora tenha apresentado EQM similar ao dos outros métodos, apresentou uma maior diferença P_F. Esse resultado, avaliado a sua flutuação nos casos proposto, pode indicar que é o método que mais é afetado na variação dos parâmetros.
- O método GRASP demonstrou desempenho intermediário tanto para EQM, quanto em tempo computacional, mas o melhor EPMA e diferença de P_F. Isso indica que o método se adequou bem a análise proposta, apesar do seu algoritmo possuir um comportamento aleatório.
- O método Optuna, foi o mais rápido, apresentando o menor tempo computacional. Apesar de apresentar os piores parâmetros para os critérios de qualidade como EQM e EPMA, a diferença entre os resultados não foi muita, indicando que o método é rápido e eficiente, sem sacrificar significativamente a qualidade da análise.
- É importante notar que a maioria utiliza a função de ativação reLU, o que sugere que essa função melhor se adequa ao estudo.

6 CONCLUSÃO

O presente trabalho teve como objetivo desenvolver uma RNA para a análise de confiabilidade de dutos submarinos corroídos e verificar sua capacidade de substituir a equação proposta por (NETTO, 2009) nessas análises. Para isso, foram desenvolvidas quatros RNAs: uma básica, sem a utilização de algoritmos ou meta-heurísticas de otimização; outra utilizando a meta-heurística GRASP; uma terceira com a meta-heurística PSO; e, por fim, uma quarta RNA otimizada com o algoritmo Optuna. O objetivo do desenvolvimento dessas RNAs está na comparação dos seus respectivos resultados e identificação de qual delas melhor se adequa ao propósito deste estudo.

A função que foi minimizada nas análises foi a EQM, que realiza a comparação entre os resultados previstos e os resultados reais, aumentando assim a precisão das RNAs. Outro aspecto que foi avaliado no estudo foi o tempo computacional das RNAs, de forma a obter resultados precisos e rápidos.

A análise indica que o modelo básico apresenta um erro significativamente maior que os outros, indicando que é necessário adotar meta-heurística ou algoritmo nas RNAs para obter resultados precisos. Os outros três modelos apresentaram resultados bastante similar e satisfatório, apresentando erro percentual de cerca de 3% comparado ao obtido pela formulação proposta por (NETTO, 2009).

Com isso, é possível concluir que as RNAs desenvolvidas adotando um método de otimização dos parâmetros podem substituir a formulação proposta por (NETTO, 2009), obtendo resultados extremamente próximos. Com relação ao tipo de método de otimização, foi demonstrado que os três métodos são viáveis, com cada um apresentando suas vantagens individuais.

O algoritmo Optuna se destaca por apresentar melhor velocidade, com erro ligeiramente maior em comparação às outras análises, porém não de forma significativa a ponto de comprometer a qualidade dos resultados. Outra vantagem observada pelo autor, é que o Optuna foi o método mais fácil de implementar, por ser uma biblioteca nativa da linguagem Python e exigindo apenas a otimização do seu parâmetro *trials*, em comparação ao GRASP e PSO que envolvem muito mais parâmetros para otimização.

Outro ponto importante observado é que os resultados obtidos pelo RNA + GRASP flutuavam a cada tentativa, visto que essa metodologia busca valores de forma aleatória, enquanto os resultados obtidos nas análises RNA + PSO e RNA + Optuna foram mais constantes.

6.1 Trabalhos Futuros

Como recomendação para trabalhos futuros, sugere-se realizar o treinamento das RNAs utilizando dados obtidos através de Elementos Finitos (EFs), ao invés de utilizar a formulação de (NETTO, 2009). Essa proposta é feita devido as limitações da formulação proposta por Netto, que restringem as especificações dos dutos e, consequentemente, o alcance da análise. Além disso, é importante destacar que o erro médio de 4,4% relatado por (NETTO, 2010), quando comparado a análise realizada por (SAKAKIBARA; KYRIAKIDES; CORONA, 2008) utilizando EFs, pode ser reproduzido na análise e até ampliado, afastando-se assim do comportamento real.

REFERÊNCIAS

AKIBA, T. et al. **Optuna: A Next-generation Hyperparameter Optimization Framework**. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining. **Anais**...2019.

AMERICAN SOCIETY OF MECHANICAL ENGINEERS. **Boiler and Pressure Vessel Code**. New York: ASME, 2019.

BEN SEGHIER, M. E. A.; MUSTAFFA, Z.; ZAYED, T. Reliability assessment of subsea pipelines under the effect of spanning load and corrosion degradation. **Journal of Natural Gas Science and Engineering**, v. 102, p. 104569, jun. 2022.

BERNBARDT, S.; KUHN, A.; WENZEL, C. Hyperparameter optimization in machine learning with Optuna. **Journal of Artificial Intelligence Research**, 2020.

BRANDÃO, B. C. **Otimização por enxame de partículas de rede neural artificial para previsão de pressão de falha em dutos corroídos**. [s.l.] Universidade Federal de Pernambuco, 2023.

BRATTON, D.; KENNEDY, J. **Defining a Standard for Particle Swarm Optimization**. 2007 IEEE Swarm Intelligence Symposium. **Anais**...IEEE, abr. 2007.

CETESB. **Principais acidentes | Emergências Químicas**. Disponível em: <https://cetesb.sp.gov.br/emergencias-quimicas/tipos-de-acidentes/dutos/principais-acidentes/>. Acesso em: 8 out. 2022.

CHOI, K.-H. et al. Comparison of computational and analytical methods for evaluation of failure pressure of subsea pipelines containing internal and external corrosions. **Journal of Marine Science and Technology**, v. 21, n. 3, p. 369–384, 15 set. 2016.

CLERC, M.; KENNEDY, J. The particle swarm-explosion, stability, and convergence in a multidimensional complex space. **IEEE transactions on Evolutionary Computation**, v. 6, n. 1, p. 58–73, 2002.

DE SIQUEIRA MOTTA, R.; MARQUES FERREIRA, A. D.; AFONSO, S. M. B. Probabilistic assessment of complex corrosion in pipelines considering River-Bottom Profile information. **Engineering Failure Analysis**, v. 165, p. 108801, nov. 2024.

ENGELBRECHT, A. P. **Fundamentals of Computational Swarm Intelligence**. [s.l.] John Wiley & Sons, 2005.

FERREIRA, A. D. M. et al. Multiresolution analysis and deep learning for corroded pipeline failure assessment. **Advances in Engineering Software**, v. 162–163, p. 103066, dez. 2021.

GASPAR CUNHA, A.; TAKAHASHI, R.; HENGGELER ANTUNES, C. **Manual de computação evolutiva e metaheurística**. [s.l.] [s.l.]: [s.n.], 2013.

GLOROT, X.; BORDES, A.; BENGIO, Y. Deep Sparse Rectifier Neural Networks. **Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics**, v. 15, p. 315–323, 2011.

GONG, S. et al. On the influence of interacting dual defects on the collapse pressure of pipes under external pressure. **Thin-Walled Structures**, v. 157, p. 107140, dez. 2020.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [s.l.] MIT Press, 2016.

Google Colaboratory. , [s.d.]. Disponível em: <https://research.google.com/colaboratory/intl/pt-BR/faq.html>. Acesso em: 11 out. 2024

GRANDELLE, RENATO. **Vazamento de mais de 1 milhão de litros de óleo na Baía de Guanabara completa 20 anos - Jornal O Globo**. Disponível em: <https://oglobo.globo.com/brasil/vazamento-de-mais-de-1-milhao-de-litros-de-oleo-na-baia-de-guanabara-completa-20-anos-1-24198470>. Acesso em: 8 out. 2022.

HADIGOL, M.; DOOSTAN, A. Least squares polynomial chaos expansion: A review of sampling strategies. **Computer Methods in Applied Mechanics and Engineering**, v. 332, p. 382–407, abr. 2018.

HAYKIN, S. **Neural Networks: A Comprehensive Foundation**. [s.l.] Prentice Hall, 1999.

HAYKIN, S. **Redes neurais: Princípios e Prática**. 2. ed. [s.l.] BookmanEditora, 2001.

KENNEDY, J.; EBERHART, R. **Particle swarm optimization**. [s.l.] IEEE International Conference on Neural Networks, 1995.

KISSELL, R. L. Nonlinear Regression Models. **Algorithmic Trading Methods**, p. 197–219, 2021.

LECUN, Y. et al. Gradient-Based Learning Applied to Document Recognition. **Proceedings of the IEEE**, v. 86, n. 11, p. 2278–2324, 1998.

LI, X. et al. Probabilistic fatigue failure assessment of free spanning subsea pipeline using dynamic Bayesian network. **Ocean Engineering**, v. 234, p. 109323, ago. 2021.

MARQUES FERREIRA, A. D. et al. Stochastic assessment of burst pressure for corroded pipelines. **Journal of the Brazilian Society of Mechanical Sciences and Engineering**, v. 43, n. 4, p. 193, 9 abr. 2021.

MCKAY, M. D.; BECKMAN, R. J.; CONOVER, W. J. Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code. **Technometrics**, v. 21, n. 2, p. 239–245, maio 1979.

MONDAL, B. C.; DHAR, A. S. Burst pressure of corroded pipelines considering combined axial forces and bending moments. **Engineering Structures**, v. 186, p. 43–51, maio 2019.

NETTO, T. A. On the effect of narrow and long corrosion defects on the collapse pressure of pipelines. **Applied Ocean Research**, v. 31, n. 2, p. 75–81, abr. 2009.

NETTO, T. A. A simple procedure for the prediction of the collapse pressure of pipelines with narrow and long corrosion defects — Correlation with new experimental data. **Applied Ocean Research**, v. 32, n. 1, p. 132–134, fev. 2010.

PREECE, R.; MILANOVIC, J. V. Efficient Estimation of the Probability of Small-Disturbance Instability of Large Uncertain Power Systems. **IEEE Transactions on Power Systems**, v. 31, n. 2, p. 1063–1072, mar. 2016.

PYTHON SOFTWARE FOUNDATION. **Python Language Reference, version 3.13**. , 2024. Disponível em: <https://www.python.org>. Acesso em: 13 out. 2024

RESENDE, M. G. C.; RIBEIRO, C. C. **Optimization by GRASP: Greedy Randomized Adaptive Search Procedures**. [s.l.] Springer, 2016.

ROSENBLATT, F. The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain. **Psychological Review**, v. 65, p. 386–408, 1958.

RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, p. 533–536, 1986.

SAKAKIBARA, N.; KYRIAKIDES, S.; CORONA, E. Collapse of partially corroded or worn pipe under external pressure. **International Journal of Mechanical Sciences**, v. 50, n. 12, p. 1586–1597, dez. 2008.

SILVA, S. M. DA. **Otimização de hiperparâmetro de uma rede neural artificial por GRASP reativo para previsão de falha em dutos corroídos**. **Universidade Federal de Pernambuco**, 2024.

TRELEA, I. C. The particle swarm optimization algorithm: convergence analysis and parameter selection. **Information processing letters**, v. 85, n. 6, p. 317–325, 2003.

WANG, Q.; ZHOU, W. A new burst pressure model for thin-walled pipe elbows containing metal-loss corrosion defects. **Engineering Structures**, v. 200, p. 109720, dez. 2019.

XIE, M.; TIAN, Z. A review on pipeline integrity management utilizing in-line inspection data. **Engineering Failure Analysis**, v. 92, p. 222–239, out. 2018.

ANEXO A

```
import pickle

import numpy as np

from pyDOE import lhs

import optuna

import pickle

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.optimizers import Adam

from tensorflow.keras import Input

from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error, mean_absolute_error

import random

import pyswarms as ps

from pyswarms.utils.functions import single_obj

import time

import numpy as np

from keras.models import load_model


# d - profundidade máxima defeito; t - espessura; l - comprimento máximo defeito, De - Diametro

externo, c - largura maximo defeito

# Formulação desenvolvida por Netto

def Rel(dt, c_PiD, l_10D):

  Raz=((1-dt)/(1-dt*(1-((c_PiD)**0.4)*((l_10D)**0.4))))**2.675
```

```
  return Raz
def generate_lhs_samples_for_defect_type(N, dt_range, tD_range, c_piD_range, l_10D_range):
    # Gera amostras LHS para o intervalo específico de cada tipo de defeito
    lhs_sample = lhs(4, samples=N)  # Ajustado para 4 variáveis: dt, tD, c_piD, l_10D

    dt_samples = dt_range[0] + (dt_range[1] - dt_range[0]) * lhs_sample[:, 0]  # Relação d/t
    tD_samples = tD_range[0] + (tD_range[1] - tD_range[0]) * lhs_sample[:, 1]  # Relação t/D
     c_piD_samples = c_piD_range[0] + (c_piD_range[1] - c_piD_range[0]) * lhs_sample[:, 2]  #
Relação C/(Pi*D)
     l_10D_samples = l_10D_range[0] + (l_10D_range[1] - l_10D_range[0]) * lhs_sample[:, 3]  #
Relação L/(10*D)
    # Combina os resultados em uma matriz de amostras
    samples = np.column_stack([dt_samples, tD_samples, c_piD_samples, l_10D_samples])
    return samples
# Define o número de amostras desejado para cada tipo de defeito
n_shallow = 10000
n_moderate = 10000
n_deep = 10000
# Gera amostras para defeitos rasos
shallow_samples = generate_lhs_samples_for_defect_type(n_shallow,
                                        dt_range=[0.1, 0.2],
                                        tD_range=[0.02, 0.1],
                                        c_piD_range=[0.01, 0.1],
                                        l_10D_range=[0.1, 0.5])
# Gera amostras para defeitos intermediários
moderate_samples = generate_lhs_samples_for_defect_type(n_moderate,
```

```
                                        dt_range=[0.2, 0.4],
                                        tD_range=[0.02, 0.1],
                                        c_piD_range=[0.03, 0.1],
                                        l_10D_range=[0.3, 0.7])
# Gera amostras para defeitos profundos
deep_samples = generate_lhs_samples_for_defect_type(n_deep,
                                        dt_range=[0.4, 0.6],
                                        tD_range=[0.02, 0.1],
                                        c_piD_range=[0.05, 0.1],
                                        l_10D_range=[0.5, 1])
# Combina todas as amostras em uma única matriz
all_samples = np.vstack([shallow_samples, moderate_samples, deep_samples])
# Classificar as amostras
def classify_defects(samples):
    shallow_defects = []
    moderate_defects = []
    deep_defects = []
    for sample in samples:
        dt_ratio, tD_ratio, c_piD_ratio, l_10D_ratio = sample
        # Defeitos rasos/superficiais
        if 0.1 <= dt_ratio <= 0.2 and tD_ratio <= 0.03 and c_piD_ratio <= 0.1:
            shallow_defects.append(sample)
        # Defeitos moderadamente profundos
        elif 0.2 < dt_ratio <= 0.4 and tD_ratio <= 0.06:
            if c_piD_ratio <= 0.1:
                moderate_defects.append(sample)
```

```
            elif c_piD_ratio >= (0.15 - 0.25 * dt_ratio):
                moderate_defects.append(sample)
        # Defeitos profundos e estreitos
        elif 0.4 < dt_ratio <= 0.6 and tD_ratio <= 0.1:
            if c_piD_ratio <= (0.2 - 0.25 * dt_ratio):
                deep_defects.append(sample)
            elif c_piD_ratio >= (0.1 - 0.125 * dt_ratio):
                deep_defects.append(sample)
    return shallow_defects, moderate_defects, deep_defects
# Classifica os defeitos nas categorias rasos, intermediários e profundos
shallow_defects, moderate_defects, deep_defects = classify_defects(all_samples)
# Mostrar a quantidade de amostras mantidas após o descarte
total_retidos = len(shallow_defects) + len(moderate_defects) + len(deep_defects)
print(f"Total de amostras geradas: {len(all_samples)}")
print(f"Total de amostras retidas após a classificação: {total_retidos}")
print(f"Defeitos rasos: {len(shallow_defects)}")
print(f"Defeitos moderados: {len(moderate_defects)}")
print(f"Defeitos profundos: {len(deep_defects)}")
# Função mawp para calcular a pressão operacional
def mawp(tD , sige):
        return (2 * 0.72 * sige * tD)
# Função para calcular as falhas (pressões de falha corroídas)
def calculate_failures(samples):
    failure_count = 0
    total_samples = len(samples)
```

```
    # Lista para armazenar as pressões de falha corroídas
    falha_corroida_amostras = []
    for sample in samples:
        dt, tD, c_PiD, l_10D = sample
        # Pressão de falha para duto intacto (usando a fórmula de Barlow)
        sigma_y = 490  # Limite de escoamento (MPa) adotado para dutos submarinos
        P_falha_intacto = 2 * tD * sigma_y
        # Calcula a relação usando a função Rel para dutos corroídos
        Rel_value = Rel(dt, c_PiD, l_10D)
        P_falha_corroido = P_falha_intacto * Rel_value
        # Adiciona a pressão de falha corroída à lista
        falha_corroida_amostras.append(P_falha_corroido)
        # Calcula a pressão operacional
        P_Ope = mawp(tD, sigma_y)
        # Verifica se a pressão de operação é maior que a de falha
        if P_Ope > P_falha_corroido:
            failure_count += 1
    # Retorna apenas a lista de pressões de falha corroídas
    return falha_corroida_amostras
# Calcula as pressões de falha corroídas para todos os defeitos
falha_corroido_all = calculate_failures(all_samples)
import pickle
# Gerando dados
data_dict = {'amostras': all_samples, 'pressao_falha': falha_corroido_all}
# Salvando com Pickle
with open('dados_falha.pkl', 'wb') as f:
```

```
    pickle.dump(data_dict, f)
# Função para calcular o desvio padrão do erro
def calcular_desvio_padrao(predicoes, valores_reais):
    return np.std(predicoes - valores_reais)
# Carregando os dados salvos
with open('dados_falha.pkl', 'rb') as f:
    data_dict = pickle.load(f)
X = np.array(data_dict['amostras'])  # Entradas (ex.: dt, c_piD, l_10D)
y = np.array(data_dict['pressao_falha'])  # Saída (pressão de falha corroída)
# Ajustar y para ser 1D, se necessário
y = y.ravel()
# Dividindo os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
# Função para criar a RNA
def criar_modelo(nr_camadas, nr_neuronios, func_ativacao):
    model = Sequential()
    model.add(Input(shape=(X_train.shape[1],)))  # Definindo o formato da entrada
    model.add(Dense(nr_neuronios, activation=func_ativacao))
    for _ in range(nr_camadas - 1):
        model.add(Dense(nr_neuronios, activation=func_ativacao))
    model.add(Dense(1, activation='linear'))  # Saída linear para prever pressão de falha
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model
def rna_sem_otimizacao():
    model = Sequential()
    model.add(Input(shape=(X_train.shape[1],)))  # Definindo o formato da entrada corretamente
```

```
    model.add(Dense(64, activation='relu'))  # Primeira camada oculta
    model.add(Dense(32, activation='relu'))  # Segunda camada oculta
    model.add(Dense(1, activation='linear'))  # Saída linear (pressão de falha)
    model.compile(optimizer=Adam(), loss='mean_squared_error')
    return model
# Treinando e avaliando a RNA sem otimização
def treinar_e_avaliar_rna_basica(X_train, y_train, X_test, y_test):
    model = rna_sem_otimizacao()
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)  # Treinando o modelo
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test, pred)
    print(f"MSE RNA sem otimização: {mse}")
    return model, mse
# Função GRASP para otimizar a RNA
def grasp_optimization_reativo(X_train, y_train, X_test, y_test, iteracoes_grasp=7, alphas=[0.1, 0.3,
0.5, 0.7, 0.9], delta=10):
    melhor_modelo = None
    melhor_mse = float('inf')
    rcl_camadas = []  # Lista de candidatos recentes para número de camadas
    rcl_neuronios = []  # Lista de candidatos recentes para número de neurônios
    rcl_ativacao = []  # Lista de candidatos recentes para função de ativação
    funcoes_ativacao = ['relu', 'tanh', 'sigmoid']
     probabilidade_alphas = np.ones(len(alphas)) / len(alphas)  # Inicializa as probabilidades de cada
alpha
    f_star = float('inf')  # Melhor valor de mse até o momento
```

```
for it in range(iteracoes_grasp):
    # Escolher um alpha de forma reativa
    alpha_index = np.random.choice(len(alphas), p=probabilidade_alphas)
    alpha = alphas[alpha_index]
    # Definindo parâmetros aleatórios
    nr_camadas = random.randint(1, 5)
    nr_neuronios = random.randint(1, 64)
    func_ativacao = random.choice(funcoes_ativacao)
    # Criar e treinar a RNA
    model = criar_modelo(nr_camadas, nr_neuronios, func_ativacao)
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    # Avaliar o modelo
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test, pred)
    # Atualizar a melhor solução (f_star) se necessário
    if mse < f_star:
        f_star = mse
    # Atualizar a RCL (Lista de Candidatos Restritos) com base no alpha
    if mse < melhor_mse:
        melhor_mse = mse
        melhor_modelo = model
        rcl_camadas.append(nr_camadas)
        rcl_neuronios.append(nr_neuronios)
        rcl_ativacao.append(func_ativacao)
    # Recalcular as probabilidades dos alphas com base nos resultados
```

```
            mi_values = [mean_squared_error(y_test, criar_modelo(nr_camadas, nr_neuronios, func_ativacao).predict(X_test)) for alpha in alphas]
        q_values = [(f_star / mi) ** delta for mi in mi_values]
        probabilidade_alphas = np.array(q_values) / np.sum(q_values)  # Atualiza as probabilidades
        # Formar a RCL
        rcl_size = int(len(rcl_camadas) * alpha)
        rcl_indices = np.argsort(rcl_camadas)[:rcl_size]
        rcl_camadas = [rcl_camadas[i] for i in rcl_indices]
        rcl_neuronios = [rcl_neuronios[i] for i in rcl_indices]
        rcl_ativacao = [rcl_ativacao[i] for i in rcl_indices]
          print(f"Iteração {it+1}: Camadas: {nr_camadas}, Neurônios: {nr_neuronios}, Função de ativação: {func_ativacao}, MSE: {mse}, Alpha: {alpha}")
    return melhor_modelo, melhor_mse
def pso_objective(particle_pos):
    if particle_pos.ndim > 1:
        particle_pos = particle_pos[0]
    nr_camadas = int(particle_pos[0])
    nr_neuronios = int(particle_pos[1])
    func_ativacao_idx = int(particle_pos[2])
    func_ativacao = ['relu', 'tanh', 'sigmoid'][func_ativacao_idx]
    # Exibir a função de ativação e parâmetros utilizados
    print(f"PSO Tentativa: Camadas: {nr_camadas}, Neurônios: {nr_neuronios}, Função de ativação: {func_ativacao}")
    # Criar e treinar a RNA
    model = criar_modelo(nr_camadas, nr_neuronios, func_ativacao)
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
```

```
    # Avaliar o modelo
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test, pred)
    return mse
def pso_optimization(X_train, y_train, X_test, y_test):
    lb = [1, 5, 0]  # Limites inferiores: camadas, neurônios, ativação
    ub = [5, 64, 2]  # Limites superiores
    bounds = (np.array(lb), np.array(ub))
    # Definindo as opções para o PSO
    options = {
        'c1': 2.05,  # coeficiente de aprendizado cognitivo
        'c2': 2.05,  # coeficiente de aprendizado social
        'w': 0.9,   # peso da inércia
    }
        optimizer = ps.single.GlobalBestPSO(n_particles=10, dimensions=3, bounds=bounds,
options=options)
    cost, pos = optimizer.optimize(pso_objective, iters=7)
    nr_camadas = int(pos[0])
    nr_neuronios = int(pos[1])
    func_ativacao_idx = int(pos[2])
    func_ativacao = ['relu', 'tanh', 'sigmoid'][func_ativacao_idx]
    melhor_modelo_pso = criar_modelo(nr_camadas, nr_neuronios, func_ativacao)
    melhor_modelo_pso.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    return melhor_modelo_pso, cost
```

```
# Função de objetivo para o Optuna (ajuste para retornar o melhor modelo)
def optuna_objective(trial):
    nr_camadas = trial.suggest_int('nr_camadas', 1, 5)
    nr_neuronios = trial.suggest_int('nr_neuronios', 5, 64)
    func_ativacao = trial.suggest_categorical('func_ativacao', ['relu', 'tanh', 'sigmoid'])
    # Criar e treinar o modelo
    model = criar_modelo(nr_camadas, nr_neuronios, func_ativacao)
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)
    # Avaliar o modelo
    pred = model.predict(X_test)
    mse = mean_squared_error(y_test, pred)
    # Salvar o melhor modelo como atributo da trial
    trial.set_user_attr("model", model)
    return mse
# Função para otimização com Optuna (ajuste para retornar o melhor modelo)
def optuna_optimization(X_train, y_train, X_test, y_test, n_trials=7):
    study = optuna.create_study(direction='minimize')
    study.optimize(optuna_objective, n_trials=n_trials)
    best_model = study.best_trial.user_attrs["model"]  # Recuperar o modelo da melhor trial
    best_mse = study.best_value
    return best_model, best_mse
# Comparando os métodos com mais parâmetros
def comparar_metodos():
    resultados = []
    # Função para capturar resultados de um método
```

```
        def capturar_resultados(nome_metodo, model, mse_teste, mse_treino, tempo_treino, desvio_padrao_teste, desvio_padrao_treino):
            return {
                'metodo': nome_metodo,
                'mse_teste': mse_teste,
                'mse_treino': mse_treino,
                'tempo_treino': tempo_treino,
                'desvio_padrao_teste': desvio_padrao_teste,
                'desvio_padrao_treino': desvio_padrao_treino
            }
    # Método Básico
    print("\n--- RNA sem otimização ---")
    inicio = time.time()
        melhor_modelo_basico, melhor_mse_basico = treinar_e_avaliar_rna_basica(X_train, y_train, X_test, y_test)
    fim = time.time()
    pred_teste_basico = melhor_modelo_basico.predict(X_test)
    pred_treino_basico = melhor_modelo_basico.predict(X_train)
    mse_treino_basico = mean_squared_error(y_train, pred_treino_basico)
    desvio_padrao_teste_basico = calcular_desvio_padrao(pred_teste_basico, y_test)
    desvio_padrao_treino_basico = calcular_desvio_padrao(pred_treino_basico, y_train)
    tempo_treino_basico = fim - inicio
        resultados.append(capturar_resultados("basico", melhor_modelo_basico, melhor_mse_basico, mse_treino_basico, tempo_treino_basico, desvio_padrao_teste_basico, desvio_padrao_treino_basico))
```

```
    # Método PSO
    print("\n--- Método PSO ---")
    inicio = time.time()
    melhor_modelo_pso, melhor_mse_pso = pso_optimization(X_train, y_train, X_test, y_test)
    fim = time.time()
    pred_teste_pso = melhor_modelo_pso.predict(X_test)
    pred_treino_pso = melhor_modelo_pso.predict(X_train)
    mse_treino_pso = mean_squared_error(y_train, pred_treino_pso)
    desvio_padrao_teste_pso = calcular_desvio_padrao(pred_teste_pso, y_test)
    desvio_padrao_treino_pso = calcular_desvio_padrao(pred_treino_pso, y_train)
    tempo_treino_pso = fim - inicio
        resultados.append(capturar_resultados("PSO", melhor_modelo_pso, melhor_mse_pso, mse_treino_pso, tempo_treino_pso, desvio_padrao_teste_pso, desvio_padrao_treino_pso))
    # Método GRASP
    print("\n--- Método GRASP ---")
    inicio = time.time()
    melhor_modelo_grasp, melhor_mse_grasp = grasp_optimization_reativo(X_train, y_train, X_test, y_test)
    fim = time.time()
    pred_teste_grasp = melhor_modelo_grasp.predict(X_test)
    pred_treino_grasp = melhor_modelo_grasp.predict(X_train)
    mse_treino_grasp = mean_squared_error(y_train, pred_treino_grasp)
    desvio_padrao_teste_grasp = calcular_desvio_padrao(pred_teste_grasp, y_test)
    desvio_padrao_treino_grasp = calcular_desvio_padrao(pred_treino_grasp, y_train)
    tempo_treino_grasp = fim - inicio
```

```
    resultados.append(capturar_resultados("GRASP", melhor_modelo_grasp, melhor_mse_grasp, mse_treino_grasp, tempo_treino_grasp, desvio_padrao_teste_grasp, desvio_padrao_treino_grasp))
    # Método Optuna
    print("\n--- Método Optuna ---")
    inicio = time.time()
    melhor_modelo_optuna, melhor_mse_optuna = optuna_optimization(X_train, y_train, X_test, y_test)
    fim = time.time()
    pred_teste_optuna = melhor_modelo_optuna.predict(X_test)
    pred_treino_optuna = melhor_modelo_optuna.predict(X_train)
    mse_treino_optuna = mean_squared_error(y_train, pred_treino_optuna)
    desvio_padrao_teste_optuna = calcular_desvio_padrao(pred_teste_optuna, y_test)
    desvio_padrao_treino_optuna = calcular_desvio_padrao(pred_treino_optuna, y_train)
    tempo_treino_optuna = fim - inicio
    resultados.append(capturar_resultados("Optuna", melhor_modelo_optuna, melhor_mse_optuna, mse_treino_optuna, tempo_treino_optuna, desvio_padrao_teste_optuna, desvio_padrao_treino_optuna))
    # Exibindo os resultados
    for resultado in resultados:
        print(f"\nMétodo: {resultado['metodo']}")
        print(f"MSE Teste: {resultado['mse_teste']}")
        print(f"MSE Treino: {resultado['mse_treino']}")
        print(f"Tempo de Treinamento: {resultado['tempo_treino']} segundos")
        print(f"Desvio Padrão (Teste): {resultado['desvio_padrao_teste']}")
        print(f"Desvio Padrão (Treino): {resultado['desvio_padrao_treino']}")
```

```
        return resultados, melhor_modelo_basico, melhor_modelo_grasp, melhor_modelo_pso, melhor_modelo_optuna
# Chamar a função de comparação e salvar os modelos
resultados, melhor_modelo_basico, melhor_modelo_grasp, melhor_modelo_pso, melhor_modelo_optuna = comparar_metodos()
# Salvando as RNAs Treinadas
melhor_modelo_basico.save('modelo_basico.keras')         # Salva o modelo basico
melhor_modelo_grasp.save('modelo_grasp.keras')           # Salva o modelo GRASP
melhor_modelo_pso.save('modelo_pso.keras')               # Salva o modelo PSO
melhor_modelo_optuna.save('modelo_optuna.keras')         # Salva o modelo Optuna
# Função para calcular a diferença percentual de todas as amostras
def calcular_diferenca_percentual_total(pressao_formulacao, pressao_modelo):
    diferencas_percentuais = []
    for pf, pm in zip(pressao_formulacao, pressao_modelo):
        if pf != 0:
            diferenca = abs((pf - pm[0]) / pf) * 100  # Cálculo da diferença percentual
            diferencas_percentuais.append(diferenca)
    media_diferenca = np.mean(diferencas_percentuais) if diferencas_percentuais else 0
    return media_diferenca
# Função para gerar amostras LHS com dt e tD
def generate_lhs_samples_for_defect_type(N, dt_range, tD_range, c_piD_range, l_10D_range):
    lhs_sample = lhs(4, samples=N)  # Ajustado para 4 variáveis: dt, tD, c_piD, l_10D
    dt_samples = dt_range[0] + (dt_range[1] - dt_range[0]) * lhs_sample[:, 0]  # Relação d/t
    tD_samples = tD_range[0] + (tD_range[1] - tD_range[0]) * lhs_sample[:, 1]  # Relação t/D
     c_piD_samples = c_piD_range[0] + (c_piD_range[1] - c_piD_range[0]) * lhs_sample[:, 2]  # Relação C/(Pi*D)
```

```
    l_10D_samples = l_10D_range[0] + (l_10D_range[1] - l_10D_range[0]) * lhs_sample[:, 3]  # Relação L/(10*D)
    # Combina os resultados em uma matriz de amostras
    samples = np.column_stack([dt_samples, tD_samples, c_piD_samples, l_10D_samples])
    return samples
# Número de amostras para análise aleatória
num_amostras_al = 10000
# Gerar as amostras LHS
Al_samples = generate_lhs_samples_for_defect_type(num_amostras_al,
                                                  dt_range=[0.1, 0.6],
                                                  tD_range=[0.02, 0.1],
                                                  c_piD_range=[0.01, 0.1],
                                                  l_10D_range=[0.1, 1])
# Classificar as amostras em defeitos rasos, intermediários e profundos
Al_raso_defeito, Al_interm_defeito, Al_prof_defeito = classify_defects(Al_samples)
# Combinar todas as amostras em um conjunto só
Al_defeitos = np.vstack([Al_raso_defeito, Al_interm_defeito, Al_prof_defeito])
# Calcular as pressões de falha com a formulação usando `calculate_failures`
Al_Press_falha = calculate_failures(Al_defeitos)
# Carregar os modelos salvos
modelo_basico = load_model('modelo_basico.keras')
modelo_grasp = load_model('modelo_grasp.keras')
modelo_pso = load_model('modelo_pso.keras')
modelo_optuna = load_model('modelo_optuna.keras')
# Modelos carregados em um dicionário para facilitar a comparação
modelos = {
```

```
    'Modelo Básico': modelo_basico,
    'Modelo GRASP': modelo_grasp,
    'Modelo PSO': modelo_pso,
    'Modelo Optuna': modelo_optuna
}
# Obter as pressões de falha previstas por cada modelo
pressao_falha_basico = modelo_basico.predict(Al_defeitos)
pressao_falha_grasp = modelo_grasp.predict(Al_defeitos)
pressao_falha_pso = modelo_pso.predict(Al_defeitos)
pressao_falha_optuna = modelo_optuna.predict(Al_defeitos)
# Calcular a diferença percentual total entre as pressões da formulação e os modelos
media_diferenca_basico = calcular_diferenca_percentual_total(Al_Press_falha,
pressao_falha_basico)
media_diferenca_grasp = calcular_diferenca_percentual_total(Al_Press_falha, pressao_falha_grasp)
media_diferenca_pso = calcular_diferenca_percentual_total(Al_Press_falha, pressao_falha_pso)
media_diferenca_optuna = calcular_diferenca_percentual_total(Al_Press_falha,
pressao_falha_optuna)
# Imprimir as médias das diferenças percentuais
print(f"Média da diferença percentual (Modelo Básico): {media_diferenca_basico:.2f}%")
print(f"Média da diferença percentual (Modelo GRASP): {media_diferenca_grasp:.2f}%")
print(f"Média da diferença percentual (Modelo PSO): {media_diferenca_pso:.2f}%")
print(f"Média da diferença percentual (Modelo Optuna): {media_diferenca_optuna:.2f}%")
# Parâmetros iniciais
sige = 490
pi = 3.14
```

```
# Carregando os modelos de RNA
modelo_basico = load_model('modelo_basico.keras')
modelo_grasp = load_model('modelo_grasp.keras')
modelo_pso = load_model('modelo_pso.keras')
modelo_optuna = load_model('modelo_optuna.keras')
# Função para converter de polegadas para metros
def inches_to_meters(value_in_inches):
    return value_in_inches * 0.0254
# Função para variar os parâmetros conforme a distribuição especificada
def variar_parametros(D, t, l, d, c, pressao_operacional, N_variacoes):
    D_var = np.random.normal(D, 0.03 * D, N_variacoes)
    t_var = np.random.lognormal(np.log(t), 0.05 * t, N_variacoes)
    l_var = np.random.normal(l, 0.05 * l, N_variacoes)
    d_var = np.random.normal(d, 0.05 * d, N_variacoes)
    c_var = np.random.normal(c, 0.05 * c, N_variacoes)
    # Calcular a variação da pressão operacional usando distribuição Gumbel
    media_pressao = 1.05 * pressao_operacional
    limite_inferior = media_pressao - 0.03 * media_pressao
    limite_superior = media_pressao + 0.03 * media_pressao
        pressao_var = stats.gumbel_r.rvs(loc=media_pressao, scale=0.03 * media_pressao,
size=N_variacoes)
    pressao_var = np.clip(pressao_var, limite_inferior, limite_superior)
    return D_var, t_var, l_var, d_var, c_var, pressao_var
# Função para selecionar amostras próximas da falha
def selecionar_amostras_proximas_falha(amostras, pressoes_operacionais, pressoes_falha,
tolerancia=0.05, num_amostras=5):
```

```
    amostras_validas = []
    for i in range(len(amostras)):
        # Verifica se a pressão operacional está próxima da pressão de falha (dentro da tolerância)
        if abs(pressoes_falha[i] - pressoes_operacionais[i]) <= tolerancia * pressoes_falha[i]:
            amostras_validas.append(amostras[i])
        # Para selecionar apenas o número desejado de amostras
        if len(amostras_validas) == num_amostras:
            break
    return np.array(amostras_validas)
# Função para calcular D, t, l e d a partir das relações
def calcular_parametros_relacoes(amostras, N_variacoes):
    parametros = []
    pressoes_operacionais_var = []
    D_inicial = inches_to_meters(2)  # 2 inches para metros
    for amostra in amostras:
        dt = amostra[0]   # Relação d/t
        tD = amostra[1]   # Relação t/D
        c_piD = amostra[2] # Relação c/(πD)
        l_10D = amostra[3] # Relação l/(10D)
        # Calcular t a partir de tD
        t = tD * D_inicial
        # Calcular d a partir de dt
        d = t * dt
        # Calcular c a partir de c_PiD
        c = c_piD * D_inicial * pi
```

```
        # Calcular l a partir de l_10D e D
        l = l_10D * 10 * D_inicial
        pressao_operacional = mawp(tD, sige)  # Chamada da função mawp(tD, sige)
        # Variar os parâmetros conforme a distribuição especificada
            D_var, t_var, l_var, d_var, c_var, pressao_var = variar_parametros(D_inicial, t, l, d, c, pressao_operacional, N_variacoes)
        for i in range(N_variacoes):
            # Calcular dt_var, c_PiD_var e l_10D_var
            dt_var = d_var[i] / t_var[i]
            tD_var = t_var[i] / D_var[i]
            c_PiD_var = c_var[i] / (pi * D_var[i])
            l_10D_var = l_var[i] / (10 * D_var[i])
            # Adicionar os parâmetros variáveis à lista
            parametros.append([D_var[i], t_var[i], l_var[i], d_var[i], c_var[i], dt_var, tD_var, c_PiD_var, l_10D_var])
                pressoes_operacionais_var.append(pressao_var[i])  # Armazenar a pressão operacional variada
    return np.array(parametros), np.array(pressoes_operacionais_var)
def calcular_pressao_operacional(amostras, sige):
    pressoes_operacionais = []
    for amostra in amostras:
        tD = amostra[1]
        pressao_operacional = mawp(tD, sige)  # Chamada da função mawp(tD, sige)
        pressoes_operacionais.append(pressao_operacional)
    return np.array(pressoes_operacionais)
```

```
def calcular_pressao_falha_variada(parametros_variados, usar_formula, modelo=None):
    if usar_formula:
        return calculate_failures(parametros_variados[:, 5:])  # Passando dt_var, tD_var, c_PiD_var, l_10D_var
    else:
        # Se usar o modelo RNA, criar matriz para previsão
        pressao_falha_variados = modelo.predict(parametros_variados[:, 5:])
        return pressao_falha_variados[:, 0]
# Função para calcular o percentual de falhas por amostra
def calcular_percentual_falhas_por_amostra(pressoes_falha_variadas, pressoes_operacionais_var, N_variacoes):
    total_amostras = len(pressoes_falha_variadas) // N_variacoes
    falhas_por_amostra = []
    for i in range(total_amostras):
        start_idx = i * N_variacoes
        end_idx = (i + 1) * N_variacoes
        # Verifica se a pressão de falha é menor que a operacional
        N_falhas = sum(pressoes_falha_variadas[start_idx:end_idx] < pressoes_operacionais_var[start_idx:end_idx])
        percentual_falhas = (N_falhas / N_variacoes) * 100  # Percentual de falhas para essa amostra
        falhas_por_amostra.append(percentual_falhas)
    return falhas_por_amostra
# Exemplo de uso do código
N_variacoes = 100000  # Defina o número desejado de variações por amostra
# Gerar as pressões operacionais (já calculadas pela função mawp) e de falha (calculate_failures)
Al_pressao_operacional = calcular_pressao_operacional(Al_defeitos, sige)
```

```
# Calcular a pressão de falha
Al_pressao_falha = calculate_failures(Al_defeitos)
# Selecionar as 5 amostras com pressão operacional < pressão de falha
amostras_proximas_falha = selecionar_amostras_proximas_falha(Al_defeitos,
Al_pressao_operacional, Al_pressao_falha, 0.1, 20)
# Calculando variações e percentuais para cada abordagem
parametros_variados, pressoes_operacionais_var =
calcular_parametros_relacoes(amostras_proximas_falha, N_variacoes)
# Calcular a pressão de falha variada usando a formulação
# Fórmula de falha
pressoes_falha_formula_variadas = calcular_pressao_falha_variada(parametros_variados,
usar_formula=True)
# Calcular a pressão de falha variada usando os modelos de RNA
pressoes_falha_RNA_basico = calcular_pressao_falha_variada(parametros_variados,
usar_formula=False, modelo=modelo_basico)
pressoes_falha_RNA_grasp = calcular_pressao_falha_variada(parametros_variados,
usar_formula=False, modelo=modelo_grasp)
pressoes_falha_RNA_pso = calcular_pressao_falha_variada(parametros_variados,
usar_formula=False, modelo=modelo_pso)
pressoes_falha_RNA_optuna = calcular_pressao_falha_variada(parametros_variados,
usar_formula=False, modelo=modelo_optuna)
# Calcular o percentual de falhas para cada abordagem
percentual_falhas_formula =
calcular_percentual_falhas_por_amostra(pressoes_falha_formula_variadas,
pressoes_operacionais_var, N_variacoes)
```

```
percentual_falhas_RNA_basico = calcular_percentual_falhas_por_amostra(pressoes_falha_RNA_basico, pressoes_operacionais_var, N_variacoes)
percentual_falhas_RNA_grasp = calcular_percentual_falhas_por_amostra(pressoes_falha_RNA_grasp, pressoes_operacionais_var, N_variacoes)
percentual_falhas_RNA_pso = calcular_percentual_falhas_por_amostra(pressoes_falha_RNA_pso, pressoes_operacionais_var, N_variacoes)
percentual_falhas_RNA_optuna = calcular_percentual_falhas_por_amostra(pressoes_falha_RNA_optuna, pressoes_operacionais_var, N_variacoes)
# Resultados
print("Percentual de falhas pela formulação:", percentual_falhas_formula)
print("Percentual de falhas pela RNA Básico:", percentual_falhas_RNA_basico)
print("Percentual de falhas pela RNA GRASP:", percentual_falhas_RNA_grasp)
print("Percentual de falhas pela RNA PSO:", percentual_falhas_RNA_pso)
print("Percentual de falhas pela RNA Optuna:", percentual_falhas_RNA_optuna)
```