UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE TECNOLOGIA E GEOCIÊNCIAS
DEPARTAMENTO DE ENGENHARIA NAVAL E OCEÂNICA
CURSO DE GRADUAÇÃO EM ENGENHARIA NAVAL E OCEÂNICA

BEATRIZ COIMBRA BRANDÃO

OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS DE REDE NEURAL ARTIFICIAL PARA PREVISÃO DE PRESSÃO DE FALHA EM DUTOS CORROÍDOS

Recife

2023

BEATRIZ COIMBRA BRANDÃO

OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS DE REDE NEURAL ARTIFICIAL PARA PREVISÃO DE PRESSÃO DE FALHA EM DUTOS CORROÍDOS

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Engenharia Naval e Oceânica da Universidade Federal de Pernambuco, em Recife/PE, como requisito parcial para a obtenção do título de Engenheira Naval.

Orientador: Adriano Dayvson Marques Ferreira

Coorientador: Paulo Gabriel Santos Campos de Siqueira

Recife

2023

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

Brandão, Beatriz Coimbra.
Otimização por enxame de partículas de rede neural artificial para previsão de pressão de falha em dutos corroídos / Beatriz Coimbra Brandão. - Recife, 2023.
51 : il., tab.

Orientador(a): Adriano Dayvson Marques Ferreira
Cooorientador(a): Paulo Gabriel Santos Campos de Siqueira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Tecnologia e Geociências, Engenharia Naval - Bacharelado, 2023.
Inclui referências, apêndices.

1. Dutos corroídos. 2. Pressões de falha. 3. Rede neural artificial. 4. Particle swarm optimization. 5. Metaheurística. I. Ferreira, Adriano Dayvson Marques. (Orientação). II. Siqueira, Paulo Gabriel Santos Campos de. (Coorientação). III. Título.

620 CDD (22.ed.)

BEATRIZ COIMBRA BRANDÃO

OTIMIZAÇÃO POR ENXAME DE PARTÍCULAS DE REDE NEURAL ARTIFICIAL PARA PREVISÃO DE PRESSÃO DE FALHA EM DUTOS CORROÍDOS

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

Aprovado em: 10/05/2023

BANCA EXAMINADORA

_____

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

_____

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

_____

Profa. Dra. Silvana Maria Bastos Afonso da Silva (Examinador Externo)
Universidade Federal de Pernambuco

AGRADECIMENTOS

Agradeço, primeiramente, a Deus.

À minha mãe e avó, Adriana e Aidê, por todo o esforço ao longo da vida para que eu crescesse bem, por todo amor e palavras de apoio quando eu precisei, e por estarem sempre comigo. Essa conquista é para vocês.

Ao meu orientador, Professor Doutor Adriano Dayvson Marques Ferreira, pela paciência, disponibilidade, competência, confiança e por me permitir dar continuidade ao seu trabalho.

Ao meu coorientador e grande amigo, Paulo Gabriel, pela amizade, muita paciência, e todo apoio ao longo dessa graduação, e principalmente, deste trabalho. Sem você não teria sido o mesmo.

Aos amigos que tive a honra de conhecer ao longo dos anos, Isabela, Eduarda, José Fernando, Evelyne, Manuela, Maria Laura, Maria Luisa, Adel e tantos outros, me sinto realmente privilegiada por ter tantas pessoas especiais na minha vida.

Aos professores do Curso de Engenharia Naval da UFPE, pelos ensinamentos de sala e de vida, vocês foram fundamentais nesse processo.

“You can't always get what you want

But if you try sometimes

Well, you might find

You get what you need”

The Rolling Stones

RESUMO

O estudo de integridade estrutural em dutos com defeitos (e.g., corroídos) é um assunto relevante, visto que falhas podem causar grandes acidentes e perdas financeiras consideráveis. Assim, se faz necessário conhecer as pressões de falha de dutos em tais condições, de modo a garantir a segurança e confiabilidade da operação. Nesse contexto, (FERREIRA, 2022) propôs uma metodologia para predição da pressão de falha em dutos com defeitos axissimétricos do perfil tipo *river bottom*, usando redes neurais artificiais (RNA). Portanto, o objetivo deste trabalho é o de complementar a metodologia proposta por (FERREIRA, 2022), otimizando a arquitetura da RNA, implementando uma metaheurística, o *particle swarm optimization* (PSO) (BRATTON; KENNEDY, 2007), que se baseia no movimento coletivo de partículas pelo espaço de busca até encontrar a solução ótima, resultando no PSO+RNA. Desse modo, busca-se encontrar uma arquitetura (i.e., número de camadas ocultas, neurônios por camada e funções de ativação) cujo erro quadrático médio (EQM) entre a pressão de falha observada e a prevista pela rede seja o menor possível. A metodologia foi implementada em *Python*, e suas execuções foram realizadas através do *Google Colaboratory*. Os resultados obtidos foram comparados com os encontrados por (FERREIRA, 2022), e se mostraram satisfatórios, com valores melhores para o EQM e para o coeficiente de determinação, apesar da rede encontrada ter sido mais complexa. Além disso, foi feita uma análise de sensibilidade para avaliar o impacto do tempo de treinamento e do tamanho do espaço de busca na aplicabilidade do PSO+RNA (i.e., fornecer resultados em tempo hábil). Foi possível observar que esses parâmetros influenciam na qualidade das arquiteturas obtidas bem como no tempo de simulação. Ainda assim, é uma metodologia flexível, permitindo encontrar soluções de acordo com o grau de complexidade requerido, e é capaz de fornecer resultados úteis com respeito à arquitetura ótima a ser usada para a previsão de pressões de falha, e em outros problemas envolvendo RNA.

Palavras-chave: dutos corroídos; pressões de falha; rede neural artificial; *particle swarm optimization*; metaheurística.

ABSTRACT

A study of structural integrity in defected pipelines (e.g., corroded) is a relevant subject, since failure can cause major accidents and considerable financial losses. Thus, it is necessary to know the failure pressures of pipelines under such conditions, in order to guarantee the safety and reliability of the operation. In this context, (FERREIRA, 2022) proposed a methodology for predicting the failure pressure in pipelines with axisymmetric defects of the river bottom profile, using artificial neural networks (ANN). Therefore, the goal of this work is to complement the methodology proposed by (FERREIRA, 2022), optimizing the ANN architecture, implementing a metaheuristic, the particle swarm optimization (PSO) (BRATTON; KENNEDY, 2007), which is based on the collective movement of particles through the search space until finding the optimal solution, resulting in the PSO+ANN model. Thus, we seek to find an architecture (i.e., number of hidden layers, neurons per layer and activation functions) whose mean squared error (MSE) between the observed failure pressure and the predicted by the network is as small as possible. The methodology was implemented in Python, and its runs were carried out through Google Colaboratory. The results obtained were compared with those found by (FERREIRA, 2022), and were satisfactory, with better values for the MSE and for the coefficient of determination, despite the network found being more complex. In addition, a sensitivity analysis was performed to assess the impact of training time and search space size on the applicability of PSO+RNA (i.e., providing results in a timely manner). It was possible to observe that the parameters influence the quality of the architectures obtained as well as the simulation time. Even so, it is a flexible methodology, allowing to find solutions according to the required degree of complexity, and it is able to provide useful results regarding the optimal architecture to be used for predicting failure pressures, and in other problems involving ANN.

Keywords: corroded pipelines; failure pressures; neural artificial network; particle swarm optimization; metaheuristic.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

A malha de dutos de óleo e gás presente no Brasil possui uma extensão de milhares de quilômetros espalhados pelo país (PETROBRAS, 2014). São meios de transporte confiáveis, porém alguns aspectos devem ser considerados quanto à sua segurança, pois sua possível ruptura pode gerar acidentes capazes de causar graves danos ao meio ambiente e à vida humana, além de impactos econômicos. Desse modo, uma forma de garantir a confiabilidade e segurança da operação dos dutos é através de ações de manutenções preventivas ou corretivas.

A área de integridade de dutos vem sendo estudada por muitos autores e sob diferentes aspectos. Pode-se citar, por exemplo, a predição da pressão de falha, responsável por levar o duto à ruptura, quando este sofre o defeito de corrosão, que é a perda de material em uma certa área, causada por reações eletroquímicas entre o material do duto e o meio onde se encontra. Em (QIN; CHENG; ZHANG, 2021) foi desenvolvido um novo método usando uma modelagem de elementos finitos para prever, quantitativamente, o crescimento em três dimensões de um defeito de corrosão em dutos de aço ao longo do tempo, considerando a interação *mechano-electrochemical*; e, também, prever a pressão de falha nos mesmos dutos como função do tempo operacional. Já em (HUANG; QIN; HU, 2022) uma modelagem de elementos finitos foi desenvolvida para prever a pressão de falha em dutos X60 que possuíam mossas com a adição de corrosão submetidos à pressão interna. Este modelo quantificou o efeito causado na pressão interna do duto, como também determinou qual o fator dominante na pressão de falha do duto.

(CABRAL et al., 2022) apresenta uma nova metodologia onde são avaliados dutos com corrosão com o modelo do perfil *river bottom* (i.e., uma aproximação simplificada do perfil de corrosão tridimensional) através de uma modelagem pelo método dos elementos finitos (MEF). O *Patran* (MSC SOFTWARE CORPORATION, 2015) foi o software aplicado nesse estudo para a geração rápida dos modelos de dutos com corrosão do tipo *river bottom*. O uso desse tipo de perfil na análise traz resultados mais rápidos que o uso de modelos em 3D e estes resultados podem ser usados para auxiliar na tomada de decisão com respeito à manutenção do duto.

No trabalho de (WU et al., 2022) são feitos testes experimentais da interação entre a pressão de colapso externa e dutos com mais de uma corrosão interagindo entre si, i.e., defeitos com certa proximidade que interagem reduzindo a resistência geral do duto. Nele, a modelagem de elementos finitos do duto corroído é validada através dos resultados obtidos com o experimento realizado pelo autor. Em seguida, são analisados os parâmetros para que seja

possível selecionar os que mais afetam na interação entre corrosões (e.g., comprimento do defeito). Por fim, é encontrada uma fórmula empírica para a pressão de colapso da interação. Esta fórmula é comparada com os resultados obtidos através do MEF e conclui-se que ela pode ser usada para o cálculo da pressão de colapso de forma conservadora.

(XU et al., 2017) também foca seu trabalho em dutos com a presença de defeitos de corrosão interagindo entre si. O duto é modelado através do MEF que analisa diversos cenários com diferentes geometrias e espaçamento entre os defeitos e, com o auxílio das redes neurais artificias (RNA), procura-se prever qual pressão levará à ruptura do duto devido à presença desses tipos de corrosão.

Em (FERREIRA, 2022) busca-se desenvolver um sistema que possa prever a pressão de ruptura de dutos corroídos, com perfis de corrosão complexos através de modelos híbridos, utilizando análise multiresolução, simulações numéricas e metamodelos. Para isto, são utilizadas geometrias reais de corrosão como entrada para um modelo de RNA que tem como objetivo prever a pressão de ruptura. Os modelos utilizados são classificados como axissimétricos, divididos em perfil *river bottom* e defeitos idealizados, e tridimensionais.

Ainda em (FERREIRA, 2022), também foi empregado o MEF, pois permite aproximar os resultados encontrados em experimentos de laboratório. Para a utilização de diversas modelagens, rotinas em Python (PYTHON, 2023) foram criadas com o objetivo de gerar automaticamente modelos de elementos finitos. Com a aplicação do MEF, pode-se, através de simulações não lineares, estimar a pressão de falha em dutos corroídos sujeitos a pressão interna. O critério de escoamento de von Mises foi utilizado para identificar o escoamento plástico nos dutos e o critério de falha, que no trabalho do autor é o instante em que a tensão equivalente de von Mises em qualquer ponto da área de corrosão atinja a tensão última verdadeira do material.

Para a RNA, foram gerados modelos sintéticos a partir dos modelos híbridos, e estes foram utilizados para treinar e validar a RNA, pois há uma necessidade de uma grande quantidade de dados, e os casos sintéticos gerados apresentam características estatísticas semelhantes às de defeitos reais. Também foram usadas rotinas em Python, a escolha dos seus parâmetros foi feita através de tentativa e erro.

Outras metodologias podem ser incorporadas ao problema de predição da pressão de falha em dutos corroídos. Por exemplo, pode-se utilizar algoritmos de otimização para a seleção dos parâmetros da RNA que otimizem sua performance, em vez de se basear na tentativa e erro,

como foi feito em (FERREIRA, 2022), ou até mesmo evitar métodos exaustivos, como o *grid search* (SUN et al., 2021). Nesse contexto, (LINS et al., 2012) propôs uma combinação entre um modelo de aprendizado de máquina para regressão, o *Support Vector Machine* (SVM), e uma metaheurística, o *Particle Swarm Optimization* (PSO). A junção dessas ferramentas ocorre pois é necessária uma otimização no momento de escolher parâmetros e valores que influenciem na etapa de treinamento do modelo de aprendizagem, buscando uma minimização do erro de predição.

O PSO é uma metaheurística que faz uma analogia entre as partículas do método e a ideia da movimentação de um coletivo de organismos, que se movimentam em conjunto em busca de determinado objetivo. O PSO tem se mostrado uma ferramenta poderosa envolvendo funções objetivo com valores reais, e o faz sem a necessidade de calcular os gradientes, o que em certos casos pode ser inviável, ainda mais em casos onde a função objetivo é uma função que não é suave (MOURA et al., 2014).

1.1 OBJETIVOS

O objetivo geral para este trabalho, que se insere na área de Análise Estrutural, é o de aperfeiçoar uma metodologia capaz de prever a pressão de ruptura de dutos com defeitos axissimétricos do tipo perfil *river bottom*. Isso será possível através do treinamento de uma RNA profunda otimizada pelo PSO. Para atingir esse objetivo geral, têm-se os seguintes objetivos específicos:

- Coletar e pré processar dados para treino e validação da RNA profunda;
- Implementar uma rotina computacional que integre o algoritmo da RNA (FERREIRA, 2022), com o algoritmo do PSO, i.e., PSO + RNA;
- Investigar a aplicabilidade do modelo, em termos de qualidade das soluções e tempo hábil para obtenção dos resultados;
- Comparar com os resultados encontrados por (FERREIRA, 2022).

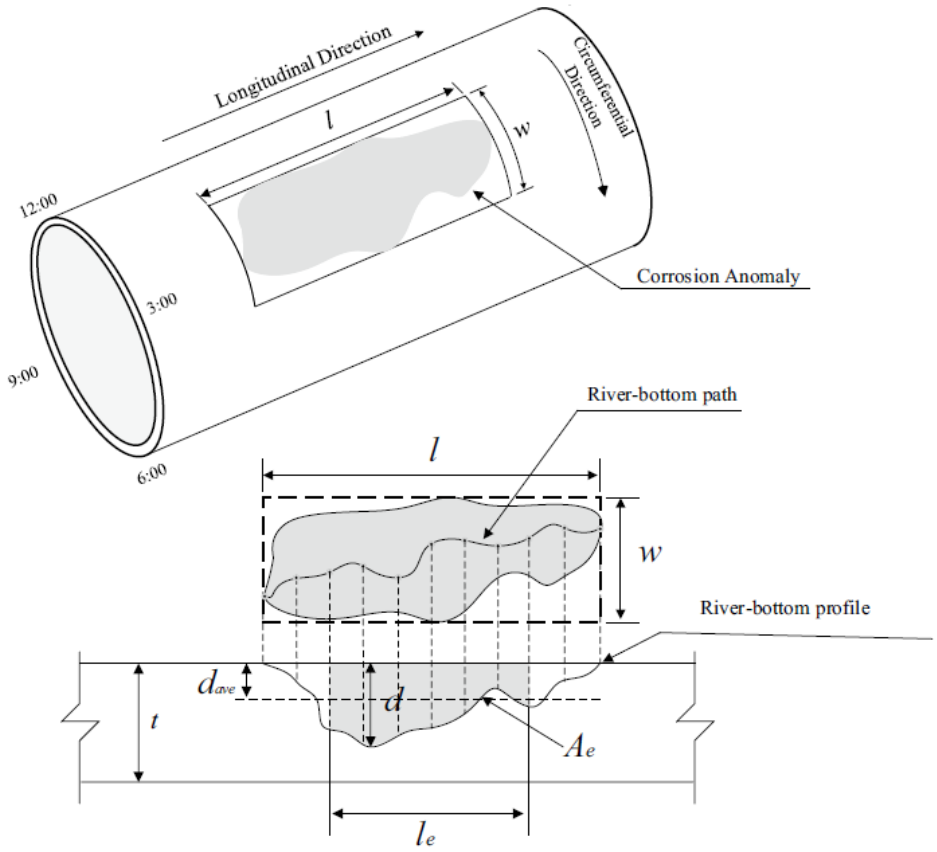
2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão vistos todos os conceitos que serão abordados mais à frente na metodologia deste trabalho.

2.1 Modelo Axissimétrico com perfil *river bottom*

Os defeitos axissimétricos possuem simetria em relação a um eixo. O perfil *river bottom* representa a situação atual da parede do duto após a corrosão, nele estão presentes as espessuras remanescentes em formato 2D, como pode ser visto na Figura 1.

Figura 1 - Ilustração do perfil *river bottom.*



Fonte: BAO; ZHOU (2020).

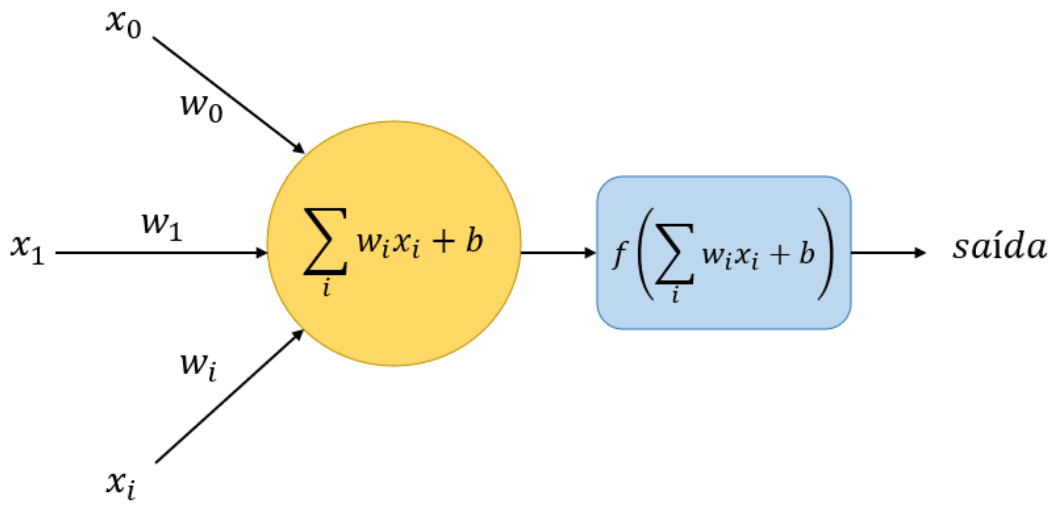
2.2 Redes Neurais Artificiais

As RNA são um método de aprendizagem de máquina que se baseia no cérebro humano, e na sua forma de aprendizado através da experiência (HAYKIN, 2000). Após diversos estudos, descobriu-se que o cérebro humano é formado por uma rede densa de neurônios que se comunicam através de sinapses, transmitindo impulsos nervosos entre elas. Na RNA, existe

uma rede densa de unidades de processamento estruturada em camadas. Esta rede receberá uma quantidade de valores como entrada e produzirá apenas uma saída. Esse tipo de método lida muito bem com problemas de regressão e de classificação (HAYKIN, 2000).

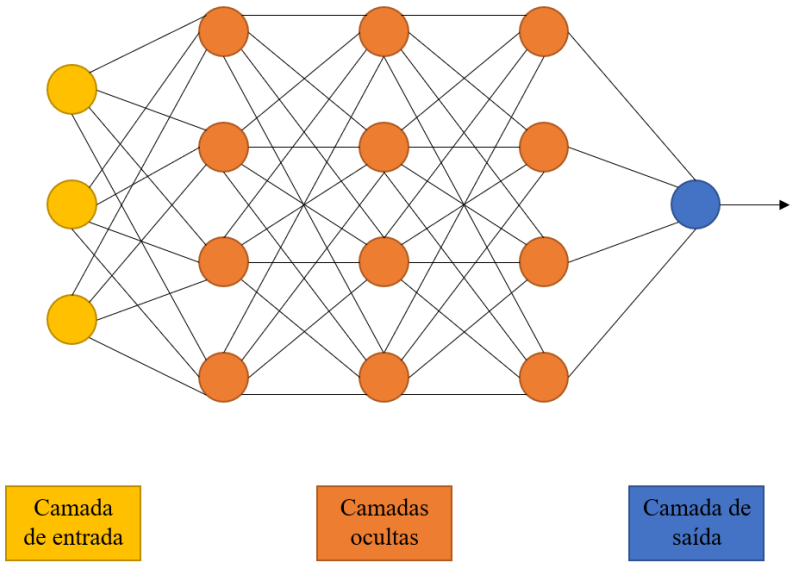
Como base para as arquiteturas de RNA de problemas mais complexos, há *perceptron*, ilustrado na Figura 2, que é considerada a arquitetura mais simples (ROSENBLATT, 1958). Nela, um único neurônio recebe diversas entradas binárias $(x_1, x_2, x_3, \dots, x_n)$ e produz uma única saída também binária. Seu cálculo de saída inclui variáveis conhecidas como pesos $(w_1, w_2, w_3, \dots, w_n)$ e que são usadas para ponderar os parâmetros de entrada $(\sum_n w_n x_n + b)$, definindo o seu grau de importância em relação à saída, no ponderamento também está presente o viés (b), que pode ser interpretado como o valor mínimo que o neurônio precisa para ser ativado. No entanto, o *perceptron* apenas consegue classificar dados linearmente separáveis. Como alternativa, foi proposto o *multilayer perceptron* (MLP), ilustrado na Figura 3, ele é composto de um ou mais neurônios na camada de entrada, uma ou mais camadas ocultas com diversos neurônios e uma camada de saída com o resultado do que foi calculado dentro da RNA (ROSENBLATT, 1957).

Figura 2 - Exemplo de um *perceptron.*



Fonte: A Autora (2023).

Figura 3 - Exemplo de um *mulitlayer perceptron.*

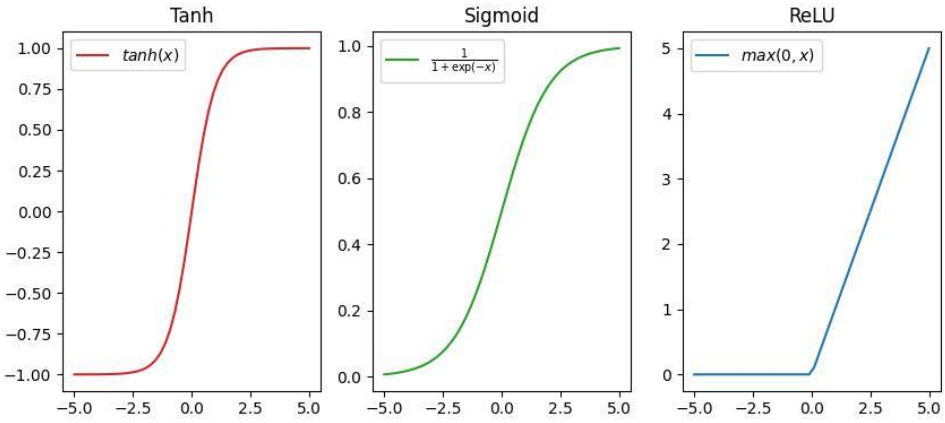


Fonte: A Autora (2023).

Há três tipos de camadas que compõem uma RNA: camada de entrada, camada oculta e camada de saída. A camada de entrada é formada pelas principais características (ou *features)* dos dados, e que serão responsáveis por gerar a variável de resposta na camada de saída. Já as camadas ocultas fazem o processamento dos dados recebidos pela camada inicial, transformando esses dados por meio de um somatório ponderado linear ($\sum_n w_n x_n + b$) e, após aplicar funções de ativação, são enviadas à próxima camada. Quando se tem muitas camadas ocultas, a RNA passa a ser denominada de profunda.

As funções de ativação são aplicadas às saídas de cada neurônio, e estas decidem se o neurônio deve ser ativado ou não, ou seja, se o seu valor de entrada tem importância para a rede ou não. Além disso, elas adicionam não linearidades ao modelo, aumentando sua capacidade de regressão ou classificação. As camadas ocultas geralmente possuem a mesma função de ativação, enquanto que a função de ativação para a camada de saída (e.g., *softmax* para problemas de classificação) costuma ser diferente a depender do tipo de problema. A escolha da função depende do objetivo que se quer atingir com a RNA, existem várias funções de ativação, mas as três mais usadas são: *Sigmoid*, Tangente hiperbólica (Tanh) e *Rectified Linear Unit* (ReLU), presentes na Figura 4.

Figura 4 - Funções de ativação.



Fonte: A Autora (2023).

Os pesos e vieses são encontrados por meio de uma técnica chamada de retropropagação, que consiste em propagar o erro de predição (i.e., função de perda). Ele é propagado ao longo da rede por meio das derivadas parciais da função de perda com relação aos pesos, e é dividido em duas etapas (RUMELHART; HINTON; WILLIAMS, 1986). Primeiro, os dados entram na RNA, com seus pesos fixos, e seus efeitos são propagados através das camadas da rede até a camada de saída. Na segunda etapa, os pesos são ajustados de acordo com a equação de minimização do erro e a saída fornecida é subtraída do valor desejado para calcular o erro. Logo, o erro é propagado de volta pela rede e consecutivamente minimizado por algum algoritmo de otimização, de modo que os pesos são ajustados para melhorar o desempenho da rede.

Para problemas de regressão, i.e., problemas onde busca-se estudar a relação entre variáveis (dependentes e independentes) tendo como objetivo fazer previsões, pode-se classificar os dados de treino conforme dois tipos de variáveis: regressora (x) e resposta (y). A primeira corresponde aos dados com as características do objeto de estudo. A segunda contém os dados relacionados com o que se busca através da RNA, que seria o alvo.

Outros parâmetros também são adotados no uso da RNA, um deles são as épocas, que é quando o conjunto de treino inteiro passa pela RNA totalizando um ciclo, as épocas se repetem até que se atinja um dos critérios de parada. Em resumo, são o número de iterações durante a qual a rede é treinada. Outra variável é o tamanho do lote de dados, que é a quantidade de amostras que entra na RNA a cada iteração, para mais informações sobre os parâmetros, consulte (FERREIRA, 2022).

2.3 *Particle Swarm Optimization*

Particle Swarm Optimization (PSO) é uma metaheurística que se baseia inicialmente na ideia do comportamento social de coletivos de animais (e.g., bando de pássaros) onde eles trabalham em conjunto na busca por uma área ideal que atinja seus objetivos (BRATTON; KENNEDY, 2007). O comportamento demonstrado pelos animais é análogo ao de um método de otimização, onde este, busca por soluções ótimas para equações não lineares em um determinado espaço de busca. É uma metaheurística simples e que necessita de poucos ajustes de parâmetro, em comparação a outras metaheurísticas, e.g., algoritmo genético.

No PSO, o elemento fundamental é a partícula i ($i = 1, \dots, n_p$). Cada partícula é descrita por três vetores: posição atual da partícula no espaço de busca ($\boldsymbol{x}_i$), melhor posição ocupada pela partícula ($\boldsymbol{p}_i$), e sua velocidade ($\boldsymbol{v}_i$). É necessário também definir a função objetivo, responsável por avaliar o desempenho das partículas. Cada partícula que compõe o enxame percorre o espaço de busca à procura do ótimo, usando tanto informações individuais, como do grupo. O processo de busca é governado pelo seguinte conjunto de equações:

$$v_{ij}(t+1) = v_{ij}(t) + c_1\epsilon_1\big(p_{ij}(t) - x_{ij}(t)\big) + c_2\epsilon_2(p_{gj}(t) - x_{ij}(t)) \quad (1)$$

$$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1) \qquad j = 1,2,\dots,d \quad (2)$$

Onde c_1 e c_2 são constantes e denotadas de parâmetro cognitivo e social, respectivamente, ϵ_1 e ϵ_2 são números aleatórios gerados a cada atualização, $\boldsymbol{p}_g$ é a melhor posição encontrada pelos vizinhos da partícula, i representa cada partícula, j é a j-ésima dimensão de cada partícula em um espaço de busca d-dimensional e t representa o passo de tempo atual (i.e., a iteração do PSO). Na Equação (1), o segundo termo representa a capacidade de cognição de cada partícula, enquanto a terceira parte está de acordo com a capacidade social de aprender com seus vizinhos. O algoritmo 1 exemplifica o processo de atualização do PSO. Outras etapas, como a inicialização do algoritmo, serão apresentadas na metodologia.

Algoritmo 1 – Etapas de atualização do PSO

para cada etapa no tempo t **faça**
  para cada partícula i dentro do enxame **faça**
    atualize a posição $\boldsymbol{x}_i(t)$ usando as Eq. (1) e (2)
    calcule o desempenho da partícula $f\big(\boldsymbol{x}_i(t)\big)$
    atualize $\boldsymbol{p}_i, \boldsymbol{p}_g$

Algoritmo 1 – Etapas de atualização do PSO

fim

fim

Os resultados da Equação (2) podem gerar partículas inviáveis, i.e., partículas que fogem do espaço de busca e não atendem às restrições do problema. No entanto, existe uma limitação imposta à velocidade máxima (v_{max}), que atinge todas as partículas, para evitar que as partículas sigam longe do espaço de busca. Essa situação ocorre quando os valores assumidos por ϵ_1 e ϵ_2 fazem com que a velocidade e, por consequência a posição da partícula, aumente muito rápido e se aproxime do infinito. Como um único valor para v_{max} não se aplica a todos os cenários de espaço de busca, outros meios de fazer o controle da explosão da velocidade surgem como uma alternativa, e são eles: o fator de inércia (w) e o fator de constrição (χ).

O fator de inércia tem como objetivo substituir a v_{max} e ajustar a influência da velocidade anterior da partícula na velocidade atual. Isso ocorre para evitar que caso haja uma velocidade no passo de tempo atual t muito alta, esta faça com que na atualização para $(t+1)$ a partícula se afaste demais do espaço de busca. Em (SHI; EBERHART, 1998) foi proposto que este valor dinâmico começasse com um valor maior que 1,0 para incentivar a exploração e, em seguida, fosse reduzido gradualmente para se concentrar na melhor área encontrada. Na equação abaixo pode-se ver a substituição.

$$v_{ij}(t+1) = wv_{ij}(t) + c_1\epsilon_1\big(p_{ij}(t) - x_{ij}(t)\big) + c_2\epsilon_2(p_{gj}(t) - x_{ij}(t)) \quad (3)$$

Por outro lado, tem-se o fator de constrição. Este fator se apresenta como um novo parâmetro que se baseia nos parâmetros cognitivo e social, e pode ser calculado de acordo com a equação abaixo.

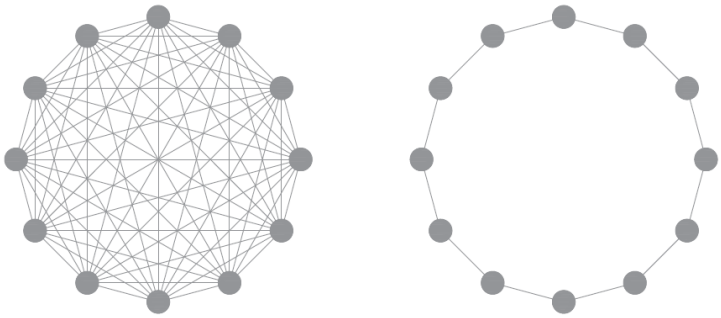
$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|}, \varphi = c_1 + c_2 \quad (4)$$

Os valores sugeridos por (BRATTON; KENNEDY, 2007) para os parâmetros da Equação (4) são: $\varphi = 4{,}1$, $\chi = 0{,}72984$, $c_1 = c_2 = 2{,}05$. A justificativa é que para $\varphi > 4$ a convergência do PSO é garantida. Portanto, a Equação (1) será atualizada para a seguinte equação:

$$v_{ij}(t+1) = \chi(v_{ij}(t) + c_1\epsilon_1\big(p_{ij}(t) - x_{ij}(t)\big) + c_2\epsilon_2(p_{gj}(t) - x_{ij}(t))) \quad (5)$$

Outro elemento importante do PSO é o número de vizinhos de cada partícula, responsável por caracterizar a rede de comunicação entre as partículas no enxame. A vizinhança de cada partícula é caracterizada de acordo com a topologia que está sendo trabalhada, podendo ser *global best* (*gbest*), em que cada partícula pode se comunicar com todas as outras, ou *local best* (*lbest*), sendo a vizinhança de uma partícula composta por apenas algumas partículas. A forma mais simples de representar essa rede de comunicações é pela topologia em anel, onde cada partícula está ligada a pelo menos duas outras partículas. Ambas as topologias estão presentes na Figura 5.

Figura 5 - Topologias a) *gbest* e b) *lbest* (topologia em anel).



Fonte: Adaptado de BRATTON; KENNEDY (2007).

Originalmente a topologia g*best* apresenta melhor performance que a *lbest*. Porém, a performance da *lbest* pode ser melhorada se for usada em conjunto com outras melhorias, conseguindo assim retornar resultados melhores. De qualquer forma, a topologia a ser adotada neste trabalho será a *gbest*.

A atualização das velocidades e posições das partículas, bem como a avaliação da função objetivo, conforme apresentado no Algoritmo 1, é repetido até que um critério de parada seja satisfeito.

3 METODOLOGIA

Neste trabalho, o PSO + RNA será usado para simultaneamente otimizar os parâmetros que são inseridos na RNA, e esta, por sua vez irá prever a pressão de ruptura de dutos com defeito axissimétrico. Esses parâmetros são as variáveis de decisão do algoritmo do PSO, são eles: número de camadas ocultas (N_c), número de neurônios (N_n) e a função de ativação (N_f). Há mais parâmetros que podem ser otimizados em uma RNA (e.g., taxa de aprendizado, *dropout*, tamanho do lote), porém não serão abordados nesse trabalho. Portanto, o espaço de busca do PSO possuirá dimensão $d = 3$, e a i-ésima partícula é descrita por meio de vetores tridimensionais $x_i = (x_{i1}, x_{i2}, x_{i3}), v_i = (v_{i1}, v_{i2}, v_{i3}), p_i = (p_{i1}, p_{i2}, p_{i3})$. A primeira, segunda e terceira dimensão correspondem a N_c, N_n, N_f, respectivamente.

3.1 Etapas do PSO+RNA

Nesta subseção, as principais etapas da metodologia do PSO+RNA serão apresentadas.

3.1.1 Inicialização do PSO

Para inicializar o PSO, é preciso primeiro definir o espaço de busca das variáveis de decisão. No caso de N_c e N_n, é preciso definir um espaço de busca inteiro, já que o número de camadas e de neurônios são valores inteiros. Portanto, o espaço de busca para N_c pode ser definido, arbitrariamente, como [1, 5], em que a RNA tem que ter pelo menos uma camada oculta e no máximo 5. Seguindo a mesma lógica, os limites de busca para N_n podem ser definidos, também arbitrariamente, como [1, 15], onde cada camada oculta tem que ter pelo menos um neurônio e no máximo 15, sendo a quantidade de neurônios igual para todas as camadas ocultas. Para N_c e N_n, apesar dos limites escolhidos terem sido arbitrários, foram escolhidos valores relativamente pequenos, de modo a evitar arquiteturas muito extensas ou complexas (i.e., com muitas camadas e neurônios). Para N_f, é preciso também escolher uma codificação inteira, já que se tratam de funções de ativação, i.e., para cada função será atribuído de modo arbitrário um número inteiro. A codificação escolhida pode ser vista a seguir na Tabela 1:

Tabela 1 - Codificações das funções de ativação.

Função de ativação	Codificação
Sigmoid	1
Tanh	2

Função de ativação	Codificação
ReLU	3

Fonte: A Autora (2023).

A topologia utilizada será a *gbest*, i.e., cada partícula troca informações com todas as partículas do enxame, sendo esta, a sua vizinhança. Os principais parâmetros do algoritmo são apresentados na Tabela 2, com valores padronizados, como proposto por (BRATTON; KENNEDY, 2007).

Tabela 2 - Parâmetros sugeridos para o PSO.

Parâmetro	Símbolo	Valor
Fator de constrição	χ	0,72984
Parâmetro cognitivo	c_1	2,05
Parâmetro social	c_2	2,05
Número de partículas	n_p	30
Número de iterações	T	100

Fonte: BRATTON; KENNEDY (2007).

As posições iniciais das partículas ($\boldsymbol{x}_i$), são escolhidas aleatoriamente e distribuídas uniformemente. A melhor posição ($\boldsymbol{p}_i$) será inicializada igual à posição de cada partícula. Já as velocidades ($\boldsymbol{v}_i$) são inicializadas de forma diferente, a velocidade máxima (v_j^{max}) para cada dimensão será igual a 10% do intervalo onde ela é definida. Sendo assim, o novo intervalo de velocidades será $[-v_j^{max}, v_j^{max}]$ e a velocidade para cada partícula será escolhida aleatoriamente. Este procedimento é para garantir que os valores das velocidades dela sejam baixos, pois isso impede que a partícula seja direcionada para áreas distantes logo nas fases iniciais do algoritmo.

3.1.2 Avaliação do desempenho das partículas

Nesta etapa haverá a integração que resulta no PSO + RNA. A função objetivo do PSO é a do erro quadrático médio (EQM), presente na Equação (6), obtido da comparação entre a variável de resposta obtida pela RNA ($\hat{y}_p$), e o valor real (y_p), onde p denota o índice daquela observação, e N é o tamanho do conjunto de dados, seja de treino ou teste. A rede será treinada com os parâmetros (N_c, N_n, N_f) obtidos pelo PSO, utilizando o conjunto de dados de treino. Em seguida, a RNA será aplicada no conjunto de dados de validação para então obter o EQM,

que será o valor a ser minimizado pelo PSO+RNA. Por fim, serão usados os dados de teste para realizar a previsão da pressão de falha de fato.

$$EQM = \frac{1}{N}\sum_{p=1}^{N}\left(y_p - \hat{y}_p\right)^2 \quad (6)$$

Por meio do valor do EQM, as posições e velocidades das partículas serão atualizados, dando seguimento ao procedimento de busca de modo a minimizar cada vez mais o EQM. Note que existe uma otimização interna, feita pela RNA para poder ajustar os pesos e vieses da rede, e a otimização proposta por este trabalho, que busca otimizar a arquitetura da rede. Para esse caso, a função de perda da RNA também será o EQM.

3.1.3 Atualização das velocidades e posições das partículas e penalização

Para a atualização das partículas, são levados em conta o fator de constrição, para evitar grandes valores de velocidades, em conjunto com um mecanismo de penalização. O mecanismo de penalização é uma estratégia para penalizar partículas que violem as restrições dos limites do espaço de busca. Por exemplo, uma partícula assumindo um valor alto de quantidade de neurônios será penalizada, recebendo um acréscimo no valor da função objetivo, aumentando o valor de uma função que deveria ser minimizada (Equação (7)).

$$penalização = \begin{cases} (it+1) * \sum_{j=1}^{3}\left(\frac{x_i - u_b}{u_b}\right), se\ x_i > u_b \\ 0,\ c.c. \end{cases} \quad (7)$$

Onde it é o valor da iteração atual e u_b é o valor do limite superior do espaço de busca. Em resumo, essa penalização vai aumentando conforme as iterações evoluem, de modo a desencorajar que as partículas fiquem muito longe do espaço de busca ao longo da otimização. Também, é um valor que é proporcional à diferença relativa entre a posição da partícula e o espaço de busca. Note que quando a partícula está dentro do espaço de busca ela não é penalizada. Assim, a função objetivo será a somatória do EQM retornado pela RNA e da penalização. Caso não haja penalização, a função objetivo se reduz ao EQM.

A atualização das partículas é feita através das Equações (2) e (5), que serão implementadas em rotina computacional. Note ainda que as variáveis devem assumir valores inteiros, e, portanto, após cada atualização os valores das variáveis serão arredondados. No entanto, este procedimento não é o ideal, sendo uma limitação do modelo, já que o arredondamento impacta na trajetória das partículas pelo espaço de busca (STRASSER et al., 2016). Há ainda um truncamento das partículas caso estas assumam valores abaixo do limite

inferior, uma vez que valores menores ou iguais a zero ocasionariam um erro no algoritmo e interromperia a busca.

3.1.4 Atualização do ótimo global e da vizinhança da partícula

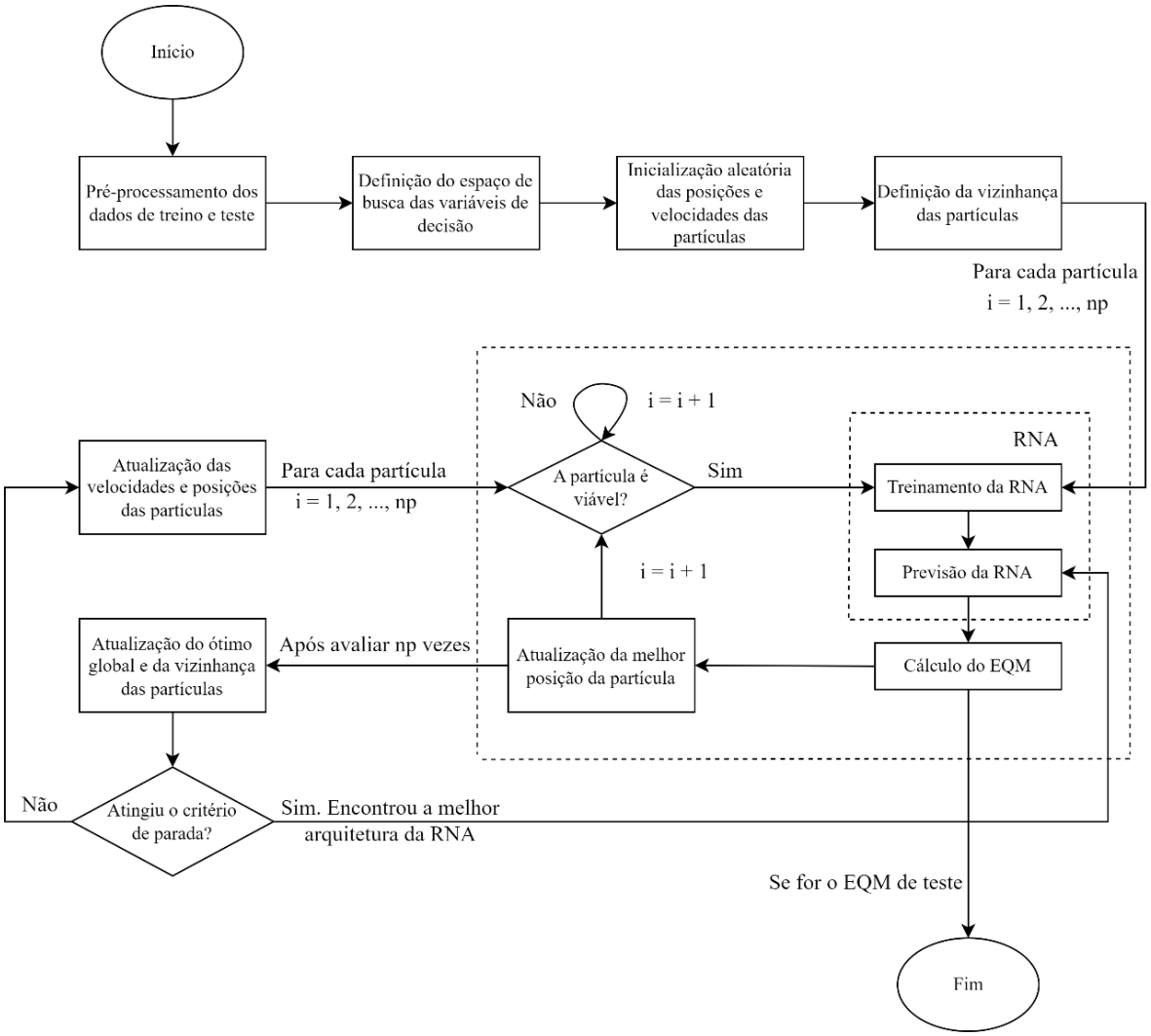
A atualização do ótimo global se dá pela identificação e armazenamento da partícula que retornou o menor valor de todos para a função objetivo. Também é armazenado o índice correspondente à iteração atual, que será usado para o critério de parada.

3.1.5 Critério de parada

Os critérios de parada estabelecidos para o PSO são dois: (i) número máximo de iterações, (ii) a partícula ótima global e seu valor para a função objetivo, deve ser a mesma em 10% do número máximo de iterações (LINS et al., 2012).

As etapas apresentadas anteriormente devem ser repetidas até que um desses critérios seja atendido. Quando isso ocorrer, o PSO+RNA fornecerá os parâmetros ótimos para a RNA. Ela, por sua vez, será capaz de prever pressão de ruptura de dutos com defeitos axissimétricos. Na Figura 6, as etapas estão apresentadas na forma de fluxograma.

Figura 6 - Fluxograma da metodologia do PSO+RNA.



Fonte: A Autora (2023).

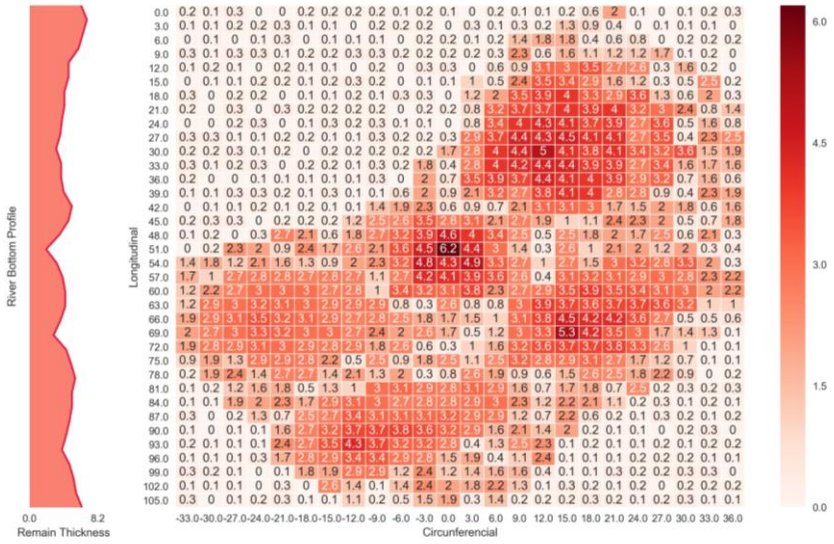
3.2 Dados para treino e validação da RNA

Os dutos a serem analisados possuem modelo axissimétrico com perfil *river bottom*. Para o treinamento e validação da RNA, foram usados como dados de entrada, por (FERREIRA, 2022), arquivos de texto na extensão .txt que contém a forma do perfil *river bottom* divido em dois parâmetros: posição longitudinal e espessura, ambos em milímetros. Como a quantidade de dados para o treinamento da RNA não era suficiente, visto que essa ferramenta requer grandes quantidades de dados para que tenha um resultado satisfatório, foram gerados 9612 casos sintéticos, sendo estes modelos gerados aleatoriamente e posteriormente armazenados em um arquivo de extensão *pickle* (.pkl), um formato específico do *Python* que comprime os dados. Na divisão dos dados dos casos sintéticos para conjuntos de teste e treino é usada uma proporção onde 70% são conjuntos de treino, 20% de validação e 10% de teste.

Para as variáveis regressoras (*features*), foram utilizados os dados do perfil *river bottom* e neles foi aplicada a transformada *wavelet* para obter os coeficientes. Para mais detalhes sobre o procedimento e aplicação da transformada *wavelet* consultar (FERREIRA, 2022).

Na Figura 7 tem-se um exemplo dos dados representando o perfil *river bottom* coletados através de uma inspeção *in-line inspection* (ILI). Estes dados são representados através de um mapa de espessuras onde a partir dos pontos formados pela posição circunferencial e posição longitudinal, é fornecida a espessura remanescente da parede naquele ponto.

Figura 7 - Exemplo de mapa de espessuras representando o perfil *river bottom.*



Fonte: FERREIRA (2022).

3.3 *Google Colaboratory*

A partir do procedimento detalhado acima, foi desenvolvido um código na linguagem *Python* para representá-lo. Este código será executado utilizando o *Google Colaboratory* (*Colab*), um serviço de nuvem gratuito oferecido pela Google com foco na pesquisa de aprendizado de máquina e inteligência artificial. O *Colab* possui uma interface similar ao *Jupyter Notebook*, permitindo que o código seja separado em células facilitando alterações ao longo do código, também pode ser compartilhado, alterado e executado por diversas pessoas.

4 RESULTADOS E DISCUSSÃO

Neste capítulo serão apresentados os resultados das simulações, bem como os casos considerados para o PSO+RNA, e também uma análise comparativa com os resultados obtidos em (FERREIRA, 2022).

4.1 Problema original

(FERREIRA, 2022) desenvolveu um modelo para prever a pressão de ruptura de dutos corroídos, com perfis de corrosão do tipo *river bottom*. Na Tabela 3 apresenta uma amostra do conteúdo de arquivos de texto que contém as características do perfil *river bottom* em uma determinada posição do duto.

Tabela 3 - Amostra de dados de um perfil *river bottom.*

Posição longitudinal [mm]	Espessura [mm]
0,00	13,987
2,00	14,018
3,00	14,025
5,00	14,072
6,00	14,072
7,00	14,114
9,00	14,118
10,00	14,114
12,00	14,114
13,00	14,134
15,00	14,125
16,00	14,143
17,00	14,146
19,00	14,144
20,00	14,115

Fonte*:* FERREIRA (2022).

Na Tabela 4 podem ser vistas amostras dos dados sintéticos utilizados para o treinamento da RNA de modo que ela seja capaz de prever a pressão de falha dado a geometria do perfil. No total foram utilizados 9612 casos sintéticos. As *features* são as variáveis regressoras, e são representadas pelos coeficientes obtidos pela transformada *wavelet*, com um

total de 15 coeficientes. Para mais detalhes sobre a extração das *features* a partir do perfil *river bottom*, veja (FERREIRA, 2022). Já a pressão de falha para este perfil é a variável de resposta.

Tabela 4 - Amostra de dados das variáveis regressoras e da variável resposta.

	Features							
Nº do caso sintético	1	2	3	...	13	14	15	Pressão de falha
1	5,144	5,148	5,129	...	4,956	0	6,251	14,8
2	10,604	10,603	10,600	...	10,472	10,468	16,082	14,6
3	0	5,230	5,225	...	0	0	4,464	14,7
4	0	5,088	5,079	...	5,057	0	5,170	18,9
5	0	5,009	5,020	...	0	0	4,880	14,7
6	0	9,969	9,953	...	0	0	15,393	27,9
7	5,157	5,158	5,153	...	5,003	5,013	6,393	14,7
8	3,680	3,681	3,666	...	3,480	0,000	3,070	15,0
9	10,636	10,641	10,625	...	10,656	10,657	16,082	14,6
...	...	...	...	...	...	...	...	...
9612	0	7,450	7,459	...	0	0	8,715	15,0

Fonte: FERREIRA (2022).

4.2 Simulações do PSO+RNA

Para realizar as simulações do modelo PSO+RNA, é necessário definir alguns parâmetros que vão refletir na aplicabilidade do código. Com isso, o algoritmo poderá ser de fato útil e retornará resultados em tempo hábil. Estes parâmetros são: iterações, épocas, e os limites máximos das variáveis de decisão, principalmente nas camadas ocultas e nos números de neurônios por camada.

Após inicializar o enxame geral de partículas (vide Seção 3.1.1), será analisado o caso em que o número de épocas para treinamento da rede é 10, o limite máximo de camadas ocultas é 5 e o limite máximo do número de neurônios é 15. Esse será o caso original (C0). Os limites de camadas e neurônios foram escolhidos visando priorizar redes menos complexas e custosas, já a quantidade relativamente baixa de épocas tem como objetivo reduzir o custo computacional do PSO+RNA e, ainda assim, fornecer uma avaliação comparativa entre as rodadas e apontar a que teve o melhor resultado (i.e., arquitetura com o menor EQM e que não teve violação das suas restrições). O algoritmo será rodado 10 vezes para avaliar melhor a variabilidade e o comportamento das soluções obtidas. Esse valor foi escolhido por conta dos altos custos computacionais envolvidos em cada rodada de simulação. O algoritmo utilizado para otimização da RNA (i.e., para otimizar seus pesos) é o *Adam* (KINGMA; BA, 2015). Na Tabela

5 podem-se observar os principais resultados encontrados após a otimização do caso original, i.e., a quantidade de camadas ocultas e a quantidade de neurônios por camada, a função de ativação, o EQM, a penalização, a iteração em que ótimo foi atingido e o tempo total de simulação. Recorde que a função objetivo é o somatório do EQM e da penalização. Apenas três rodadas não tiveram penalizações, e dentre elas, a que retornou a solução ótima (destaque em cinza na Tabela 5) foi a com menor EQM.

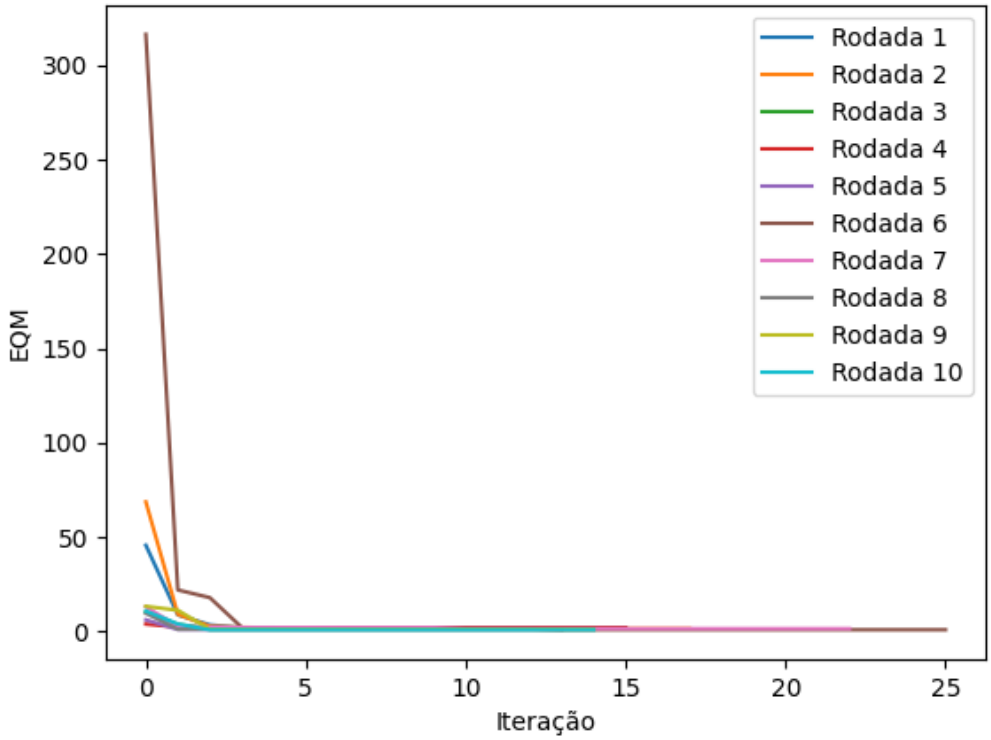
Tabela 5 - Principais resultados do PSO+RNA para o C0.

Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
1	6	20	ReLU	0,4106	1,60	3	1315,14
2	6	11	ReLU	0,4766	1,00	5	1470,92
3	8	14	ReLU	0,4235	1,20	2	1305,00
4	7	14	ReLU	0,7745	1,20	3	1363,33
5	5	11	ReLU	1,0609	0,00	1	1157,69
6	4	13	ReLU	0,9070	0,00	13	2140,06
7	5	14	ReLU	1,5164	0,00	10	1827,60
8	5	18	ReLU	0,9689	0,20	1	1126,88
9	6	16	ReLU	0,5910	0,53	2	1219,99
10	5	20	ReLU	0,2058	0,67	2	1195,82

Fonte: A Autora (2023).

A Figura 8 apresenta o histórico do EQM ao longo das rodadas, cada curva representa uma rodada com seus valores de EQM ao longo das iterações.

Figura 8 - Histórico do Erro Quadrático Médio para o C0.



Fonte: A Autora (2023).

Mesmo com um baixo número, de épocas, ainda assim esta simulação levou um tempo moderado para convergir (Tabela 5). É possível perceber que houve penalizações em mais da metade das rodadas, i.e., as soluções dessas rodadas apresentaram uma tendência a sair dos limites superiores estabelecidos. Todas as rodadas dessa simulação convergiram rapidamente, entre 1 e 13 iterações, sendo a maioria abaixo de 5, atingindo assim o critério de parada. Essa é uma vantagem importante do método, já que não foi necessário o uso das 100 iterações para encontrar a solução, o que demandaria muito mais tempo. A rodada 6 apresentou a melhor arquitetura dessa simulação, contendo 4 camadas ocultas e 13 neurônios em cada, e a função de ativação foi a ReLU.

4.3 Sensibilidade do PSO+RNA

Para avaliar a sensibilidade do modelo ao número de épocas e ao limite superior das variáveis, serão simulados novos casos contemplando essas alterações. A justificativa é para avaliar se, ao se reduzir o número de épocas a qualidade das soluções se mantém em prol de um custo computacional menor, já que se espera que o modelo retorne resultados mais rapidamente. Já para os limites superiores, o intuito é investigar se há uma tendência do PSO+RNA buscar soluções mais complexas, de fato priorizando maior quantidade de camadas

e de neurônios para melhorar o desempenho da rede. Os casos propostos se encontram na Tabela 6.

Tabela 6 - Resumo dos limites das variáveis para cada caso.

Caso	Limites das variáveis	Funções de ativação	Épocas
C0	$N_c = [1, 5]$ e $N_n = [1, 15]$	$N_f = [\text{Sigmoid}, \text{Tanh}, \text{ReLU}]$	10
C1	$N_c = [1, 5]$ e $N_n = [1, 15]$	$N_f = [\text{Sigmoid}, \text{Tanh}, \text{ReLU}]$	5
C2	$N_c = [1, 10]$ e $N_n = [1, 30]$	$N_f = [\text{Sigmoid}, \text{Tanh}, \text{ReLU}]$	5
C3	$N_c = [1, 10]$ e $N_n = [1, 30]$	$N_f = [\text{Sigmoid}, \text{Tanh}, \text{ReLU}]$	10

Fonte: A Autora (2023).

Os limites superiores originais são 5 camadas ocultas e 15 neurônios, e, ao dobrá-los, assumem os valores de 10 camadas ocultas e 30 neurônios. Desse modo, será possível investigar um *tradeoff* entre a qualidade das soluções e a aplicabilidade do modelo.

4.3.1 Primeiro caso (C1): limite superior original e 5 épocas

Observa-se nesse caso que, por possuir poucas épocas e limite superior em valores baixos, a simulação tende a levar pouco tempo para convergir. No entanto, o valor do EQM é maior quando comparado ao caso original (Tabela 5). Houve penalização em todas as rodadas. Uma hipótese para esse comportamento é que, devido à falta de tempo suficiente de treino, o PSO+RNA tentou melhorar o desempenho das soluções buscando por arquiteturas mais complexas. Ainda assim, a convergência em todas as rodadas foi bem rápida e ocorreu abaixo de 10 iterações.

Na Tabela 7 estão os resultados para cada uma das rodadas simuladas. Destacado em cinza na tabela, há a rodada que retornou a solução com o menor EQM, porém todas as rodadas desta simulação violaram as restrições do caso, portanto nenhuma dessas pode ser escolhida como a solução ótima.

Tabela 7 - Principais resultados do PSO+RNA para o C1.

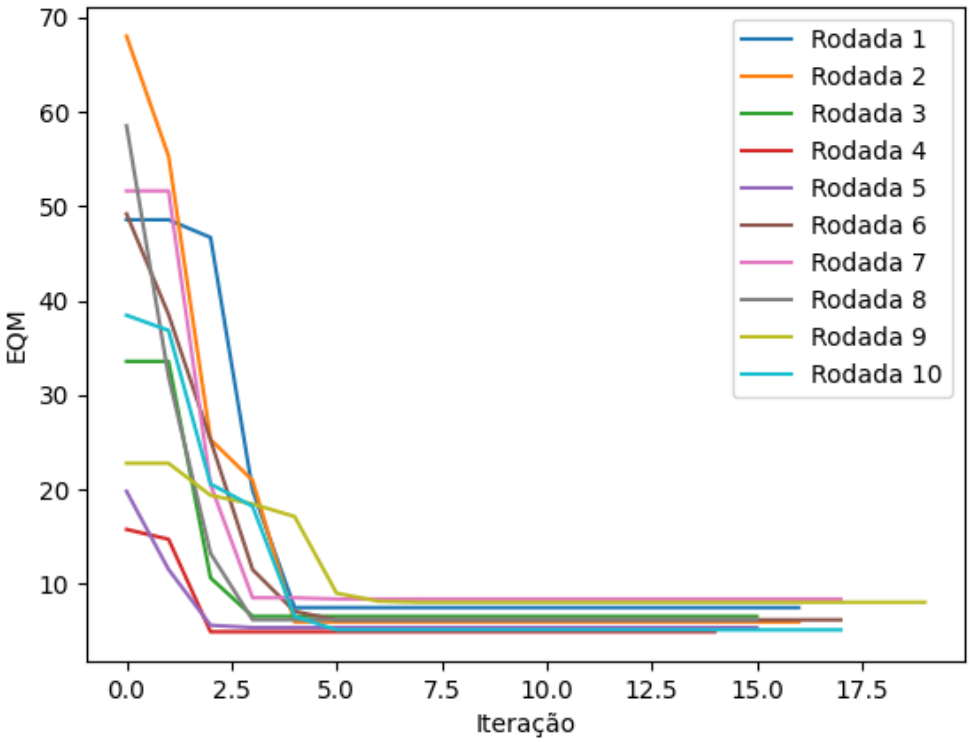
Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
1	10	19	ReLU	2,4039	5,067	4	1045,71
2	4	24	ReLU	3,5487	2,400	4	895,99
3	6	29	ReLU	3,1471	3,400	3	882,18
4	10	20	ReLU	2,2387	2,667	2	938,89
5	8	30	ReLU	0,5527	4,800	3	930,56
6	6	22	ReLU	2,8364	3,333	5	1032,02
7	12	14	ReLU	1,3830	7,000	5	1183,27

Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
8	5	36	ReLU	1,9852	4,200	3	848,25
9	5	20	ReLU	5,7018	2,333	7	1111,31
10	5	22	ReLU	2,7737	2,333	5	1024,84

Fonte: A Autora (2023).

A Figura 9 apresenta o histórico do EQM ao longo das rodadas, cada curva representa uma rodada com seus valores de EQM ao longo das iterações.

Figura 9 - Histórico do Erro Quadrático Médio para o C1.



Fonte: A Autora (2023).

4.3.2 Segundo caso (C2): limite superior dobrado e 5 épocas

Nesse caso temos novamente uma simulação com 5 épocas, o que contribui para uma simulação mais rápida. Houve apenas duas penalizações nessa simulação, e com valores baixos em comparação aos casos anteriores (Tabela 5 e Tabela 7). Esse comportamento está em concordância com a hipótese assumida na simulação anterior. O PSO+RNA de fato buscou arquiteturas mais complexas em busca do ótimo. Como os limites superiores para este caso foram dobrados, não houve grande necessidade de penalizar as soluções. A convergência das rodadas ocorreu entre 0 e 14 iterações. Curioso destacar que, na última rodada, o valor ótimo foi atingido já na inicialização.

A melhor arquitetura pode ser encontrada na rodada 10, contendo 8 camadas ocultas e 28 neurônios em cada, vale observar que nessa mesma rodada houve uma penalização. Na Tabela 8 estão os resultados para cada uma das rodadas simuladas. Destacado em cinza na tabela, há a rodada que retornou a solução com o menor EQM.

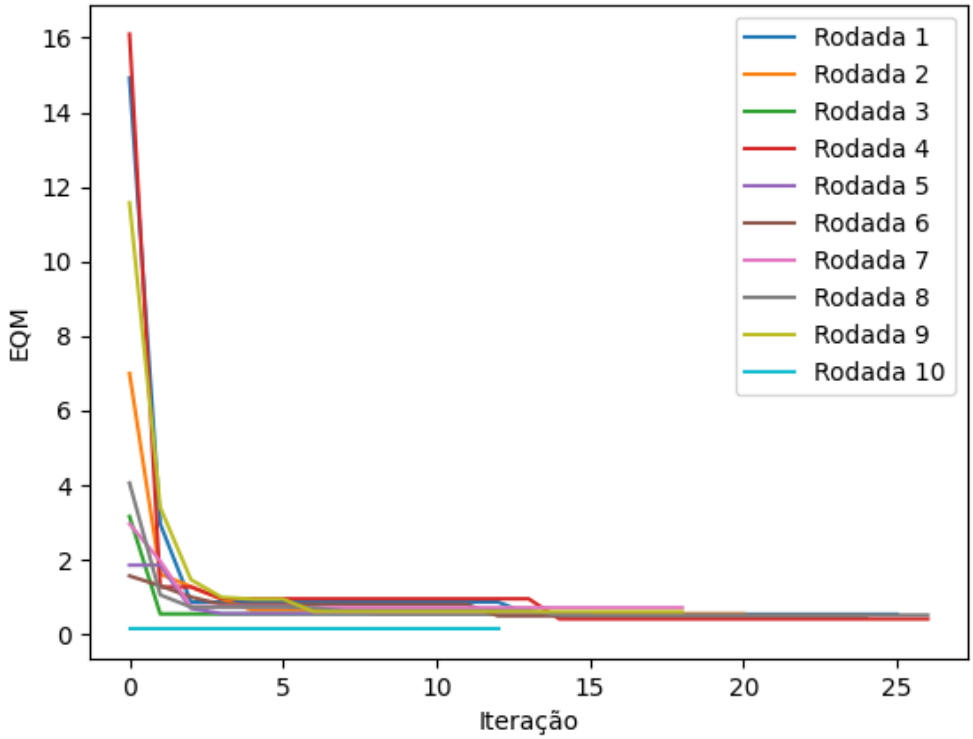
Tabela 8 - Principais resultados do PSO+RNA para o C2.

Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
1	8	29	ReLU	0,5410	0,0	13	1617,64
2	10	30	ReLU	0,5493	0,0	8	1345,70
3	8	22	ReLU	0,5498	0,0	1	883,95
4	10	28	ReLU	0,4158	0,0	14	1694,91
5	10	30	ReLU	0,5598	0,0	3	1043,47
6	10	28	ReLU	0,4976	0,0	12	1666,79
7	11	30	ReLU	0,1171	0,6	6	1285,58
8	9	28	ReLU	0,5193	0,0	14	1748,38
9	10	31	ReLU	0,4106	0,2	6	1254,73
10	8	28	ReLU	0,1366	0,0	0	806,97

Fonte: A Autora (2023).

A Figura 10 apresenta o histórico do EQM ao longo das rodadas, cada curva representa uma rodada com seus valores de EQM ao longo das iterações.

Figura 10 - Histórico do Erro Quadrático Médio para o C2.



Fonte: A Autora (2023).

4.3.3 Terceiro caso (C3): limite superior dobrado e 10 épocas

Neste caso, tem-se novamente uma simulação com 10 épocas, mas desta vez com limites superiores maiores. Nela não houve penalizações, e os valores de EQM são os menores entre todos os casos. Mais uma vez, ajudando a corroborar a hipótese levantada para C1 (Seção 4.3.1). O PSO+RNA parece, de fato, priorizar arquiteturas mais complexas durante a otimização. A quantidade de iterações que cada rodada levou para convergir variou entre 1 e 30.

A melhor arquitetura encontra-se na rodada 3, ela apresenta 8 camadas ocultas e 28 neurônios em cada. Na Tabela 9 encontram-se as saídas do C3 ao longo das rodadas. Destacado em cinza na tabela, há a rodada que retornou a solução com o menor EQM, dessa vez tendo o valor da penalização nulo.

Tabela 9 - Principais resultados do PSO+RNA para o C3.

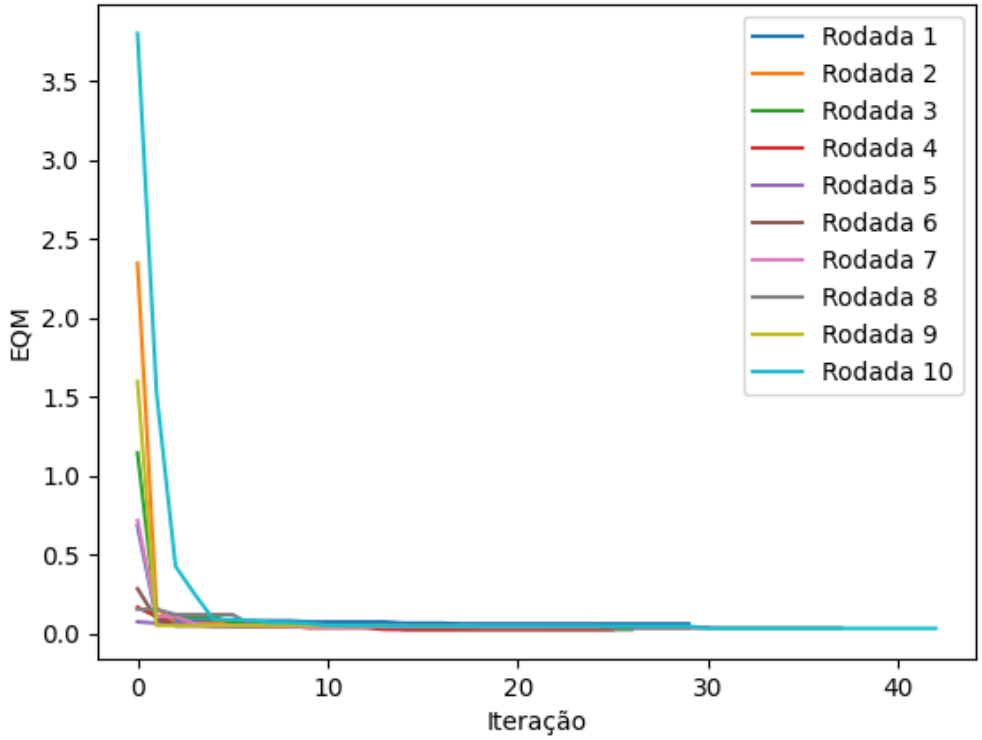
Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
1	9	30	ReLU	0,0620	0	17	2823,66
2	10	25	ReLU	0,0450	0	6	1751,90
3	8	28	ReLU	0,0257	0	14	2407,92
4	10	22	ReLU	0,0260	0	13	2445,94
5	8	30	ReLU	0,0431	0	9	2038,91

Rodada	Camadas Ocultas	Neurônios	Função de Ativação	EQM	Penalização	Iteração	Tempo [s]
6	8	30	ReLU	0,0377	0	17	2759,61
7	10	27	ReLU	0,0398	0	18	3003,65
8	10	28	ReLU	0,0356	0	25	3775,45
9	7	25	ReLU	0,0504	0	1	1227,37
10	8	23	ReLU	0,0333	0	30	3987,17

Fonte: A Autora (2023).

A Figura 11 apresenta o histórico do EQM ao longo das rodadas, cada curva representa uma rodada com seus valores de EQM ao longo das iterações.

Figura 11 - Histórico do Erro Quadrático Médio para o C3.



Fonte: A Autora (2023).

4.4 Discussão

Levando em conta todas as simulações utilizadas neste trabalho, pode-se observar que em todas as simulações, a função de ativação ótima foi a ReLU. As principais variações nos parâmetros da rede foram no número de camadas ocultas e quantidade de neurônios por camadas. Além disso, ao se investigar a sensibilidade do modelo ao número de épocas e ao limite superior, foi observado diferentes níveis de qualidade entre as simulações, já que as combinações entre esses parâmetros afetaram a estrutura da arquitetura, o valor do EQM, a quantidade de penalizações, quantidade de iterações para convergir e o tempo de simulação.

Ao aumentar o número de épocas em uma simulação, significa que a rede será treinada por mais tempo, aumentando o seu desempenho. Isso faz com que ela possua resultados melhores ao comparar com redes treinadas usando menos épocas, e, portanto, redes que não tiveram o refinamento necessário para minimizar seu erro. Ao combinar esse fator com o limite superior maior, foi observado que não houve penalizações, isso ocorre pois o enxame de partículas buscou redes mais complexas se mantendo dentro dos limites estabelecidos, que desta vez foi maior. Portanto, pode-se concluir que existe uma tendência, por parte do PSO, a escolher arquiteturas mais complexas. Ainda assim, caso se priorize redes menores, basta definir limites menores no PSO+RNA que o algoritmo irá obter a melhor solução dentro dessas circunstâncias.

Outra observação que pode ser feita sobre as combinações de parâmetros diz respeito aos valores retornados para o EQM em cada rodada, quanto mais épocas na simulação, mais refinado fica o resultado, i.e., o EQM retorna valores menores. Ao se avaliar as penalizações, é possível observar que esta é influenciada tanto pelo número de épocas como pelos limites superiores. Em simulações onde a quantidade de épocas foi 10, a quantidade de soluções penalizadas foi reduzida para menos da metade, até o caso em que nenhuma rodada foi penalizada (Tabela 9). Na Tabela 10 a seguir, foi feita uma comparação entre o EQM obtido na simulação original e comparada com os demais casos, junto com a diferença percentual (Equação (8)), a média e o desvio padrão (DP) das rodadas.

$$\Delta\% = 100 * \left(\frac{EQM_{Ck} - EQM_{C0}}{EQM_{C0}}\right), k = 1,2,3 \quad (8)$$

Note como para o C3, o EQM foi menor em todos os casos em comparação ao original, resultado da combinação entre maior tempo de treinamento e limites permitindo arquiteturas mais complexas. É possível observar também como, em média, C2 e C3 apresentam menor EQM, e ambos possuem limites maiores.

Tabela 10 - Comparação entre os EQM para todos os casos.

	C0	C1		C2		C3	
Rodada	EQM	EQM	Δ%	EQM	Δ%	EQM	Δ%
1	0,4106	2,4039	485,41%	0,5410	31,75%	0,0620	-84,90%
2	0,4766	3,5487	644,55%	0,5493	15,26%	0,0450	-90,56%
3	0,4235	3,1471	643,09%	0,5498	29,82%	0,0257	-93,93%
4	0,7745	2,2387	189,06%	0,4158	-46,31%	0,0260	-96,64%
5	1,0609	0,5527	-47,90%	0,5598	-47,24%	0,0431	-95,94%
6	0,9070	2,8364	212,71%	0,4976	-45,14%	0,0377	-95,84%
7	1,5164	1,3830	-8,79%	0,1171	-92,28%	0,0398	-97,38%

	C0	C1		C2		C3	
Rodada	EQM	EQM	Δ%	EQM	Δ%	EQM	Δ%
8	0,9689	1,9852	104,88%	0,5193	-46,40%	0,0356	-96,33%
9	0,5910	5,7018	864,85%	0,4106	-30,51%	0,0504	-91,47%
10	0,2058	2,7737	1247,55%	0,1366	-33,63%	0,0333	-83,82%
Média	0,7335	2,6571	262,24%	0,4297	-83,61%	0,0399	-104,77%
DP	0,3905	1,3781	252,92%	0,1683	-93,35%	0,0110	-101,18%

Fonte: A Autora (2023).

Além disso, foi investigado também o tempo de simulação, para verificar se os resultados podem ser fornecidos em tempo hábil. É possível perceber que a quantidade de iterações até a convergência foi aumentando de acordo com a quantidade de épocas e a possibilidade de estruturas mais complexas. Sendo assim, é possível concluir que o procedimento de busca realizado pelo PSO vai ficando mais custoso. O tempo levado para a convergência de cada rodada também seguiu o padrão de aumentar proporcionalmente ao aumento dos limites e da quantidade de épocas. Esses tempos podem ser observados na Tabela 11 a seguir.

Tabela 11 - Comparação entre os tempos para todos os casos.

	C0	C1		C2		C3	
Rodada	Tempo [s]	Tempo [s]	Δ%	Tempo [s]	Δ%	Tempo [s]	Δ%
1	1315,14	1045,71	-20,49%	1617,64	23,00%	2823,66	114,70%
2	1470,92	895,99	-39,09%	1345,70	-8,51%	1751,90	19,10%
3	1305,00	882,18	-32,40%	883,95	-32,26%	2407,92	84,52%
4	1363,33	938,89	-31,13%	1694,91	24,32%	2445,94	79,41%
5	1157,69	930,56	-19,62%	1043,47	-9,87%	2038,91	76,12%
6	2140,06	1032,02	-51,78%	1666,79	-22,12%	2759,61	28,95%
7	1827,60	1183,27	-35,26%	1285,58	-29,66%	3003,65	64,35%
8	1126,88	848,25	-24,73%	1748,38	55,15%	3775,45	235,04%
9	1219,99	1111,31	-8,91%	1254,73	2,85%	1227,37	0,61%
10	1195,82	1024,84	-14,30%	806,97	-32,52%	3987,17	233,42%
Média	1412,24	989,30	-29,95%	1334,81	-5,48%	2622,16	85,67%
DP	326,43	107,94	-66,94%	344,26	5,46%	851,03	160,70%

Fonte: A Autora (2023).

4.5 Análise comparativa

Apesar do caso original ser o C0, foi observado que todas as soluções encontradas no caso C3 são viáveis, pois além do baixo valor do EQM, não violam as restrições das variáveis.

Portanto, foi escolhida a melhor solução desse caso para poder comparar com a rede proposta por (FERREIRA, 2022), ambas foram treinadas usando 200 épocas e o resultado desta comparação pode ser visto na Tabela 12.

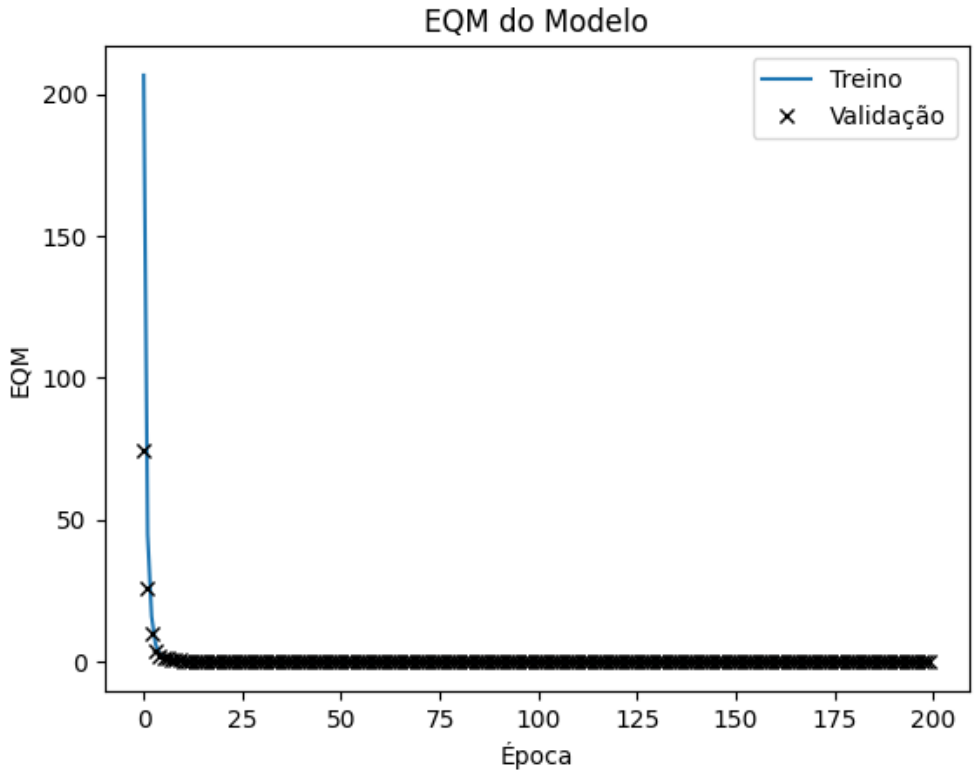
Tabela 12 - Comparação de parâmetro entre as arquiteturas.

	Caso C3	(FERREIRA, 2022)	Δ%
Camadas ocultas	8	3	166,667%
Neurônios	28	15	86,667%
EQM treino	0,0146	0,0162	-9,877%
EQM teste	0,0071	0,0111	-36,327%
R^2	0,9998	0,9997	0,011%

Fonte: A Autora (2023).

Na Figura 12 pode ser visto o histórico do EQM ao longo de todas as épocas para a rede encontrada na simulação inicial. Note que esse EQM apresentado é o utilizado pelo algoritmo interno da RNA para otimizar seus parâmetros.

Figura 12 - Histórico do EQM para os conjuntos de treino e validação.



Fonte: A Autora (2023).

Diante dos resultados, é possível perceber que a arquitetura apresentada como ótima na simulação original é mais complexa que a usada por (FERREIRA, 2022), o que nos mostra a preferência que o PSO demonstra para redes mais complexas, e contraria um pouco o objetivo

de priorizar redes mais simples e menos custosas. Por outro lado, o EQM de treino e teste obtidos por meio da rede encontrada por este trabalho possui valores menores em comparação aos obtidos por (FERREIRA, 2022).

Além do EQM, usou-se também outra métrica adicional, o coeficiente de determinação ($0 \leq R^2 \leq 1$). Este coeficiente pode ser interpretado como o nível de variabilidade nos dados que pode ser explicado pelo modelo (MONTGOMERY; RUNGER, 2018). Em outras palavras, o quanto as *features* utilizadas realmente são capazes de prever a pressão de falha através do modelo desenvolvido por (FERREIRA, 2022). O resultado encontrado para o R^2, aplicados no conjunto de treino, é maior que o de (FERREIRA, 2022), mostrando que a rede deste trabalho se ajusta melhor à regressão, mostrando ser uma rede que tem maior capacidade preditiva da pressão de falha. Os resultados destas duas métricas mostram que, de fato para o problema em questão, arquiteturas mais complexas resultaram em um maior desempenho.

5 CONCLUSÕES

O objetivo proposto por esse trabalho visa aperfeiçoar uma metodologia para previsão de pressões de falha em dutos com defeitos axissimétricos utilizando RNA. Isso é feito propondo-se uma nova metodologia para otimização desta RNA utilizando o PSO, resultando no PSO+RNA. Esse modelo utiliza como variáveis de decisão o número de camadas ocultas, número de neurônios e a função de ativação para selecionar uma arquitetura ótima. A função objetivo a ser minimizada é a somatória do EQM (i.e., erro entre o valor real e o valor previsto) e da penalização (i.e., acréscimo no EQM para evitar soluções de arquiteturas mais complexas). Posteriormente, essa arquitetura foi comparada com a rede proposta por (FERREIRA, 2022) para previsão de pressões de falha em dutos corroídos. Da análise comparativa, pode-se concluir que a rede encontrada através deste trabalho, embora seja mais complexa, se ajusta melhor à regressão e assim demonstra ter uma capacidade maior na previsão de pressão de falha. Adicionalmente, foi realizada uma análise de sensibilidade para investigar o desempenho do PSO+RNA ao variar o número de épocas e os limites superiores, visando um *tradeoff* entre qualidade das soluções e obtenção delas em tempo hábil. Observou-se que o modelo tende a encontrar arquiteturas mais complexas.

Sobre as vantagens do modelo, uma delas é a possibilidade de selecionar a arquitetura da rede de forma mais inteligente, por meio de metaheurísticas, que permite a otimização paralela através das partículas, e se beneficiando da troca de informações entre elas. Ao realizar uma busca manual por tal arquitetura, muito tempo será consumido e pode não ser possível encontrar a solução ótima em um tempo viável. Dessa forma, se torna indispensável a aplicação de métodos de otimização na procura por resultados mais corretos possíveis. Este é um método melhor que o *grid search*, por exemplo, onde se enumera as possíveis soluções e todas são avaliadas. O modelo também é flexível, uma vez que ele se adequa às preferências do analista (e.g., complexidade da rede), apenas fazendo mudanças nos limites do espaço de busca. Outro ponto positivo é o fato de que, apesar de terem sido usadas apenas três funções de ativação, o modelo não se limita somente a elas, já que outras funções também podem ser incluídas. Sobre a sua convergência, verificou-se que ela ocorre rapidamente pelo fato de não haver a necessidade de o modelo percorrer todas as iterações definidas até encontrar o ótimo, por conta do critério de parada. Esse modelo também pode ser ajustado para ser utilizado em problemas envolvendo outros tipos de RNA, como redes convolucionais e recorrentes.

Por outro lado, algumas limitações foram identificadas ao longo do trabalho. Uma delas é a necessidade de arredondar os valores das variáveis de decisão a cada atualização das partículas, já que estas precisam ser inteiras. No entanto, essa ação não é o ideal pois acaba interferindo na trajetória das partículas através do espaço de busca. Além disso, existe um alto custo computacional envolvido, já que é necessário avaliar a RNA diversas vezes. No método da tentativa e erro, valores vão sendo testados, mas por este não ser um método devidamente estruturado, não há como ter certeza de que o resultado encontrado é o melhor resultado possível. Já com a otimização, existe um procedimento estruturado que vai garantir que ao final o valor encontrado seja o melhor valor possível, i.e., aquele em que o erro seja o menor possível. A metodologia desenvolvida neste trabalho também pode ser utilizada em outras aplicações de RNA, não se limitando a otimizar para prever pressão de falha, e servindo também a problemas de classificação, além do de regressão tratado nesse trabalho.

5.1 Trabalhos futuros

Como sugestão para trabalhos futuros, para vencer a limitação do arredondamento das variáveis de decisão, poderia aplicar o PSO de forma mais apropriada para lidar com variáveis inteiras, sendo este mais adequado para lidar com o problema deste trabalho. Outra opção, seria a aplicação de outras metaheurísticas que também são conhecidas por trabalharem com valores inteiros, e.g., algoritmo genético.

6 REFERÊNCIAS BIBLIOGRÁFICAS

1 BAO, J.; ZHOU, W. Influence of the Corrosion Anomaly Class on Predictive Accuracy of Burst Capacity Models for Corroded Pipelines. **International Journal of Geosynthetics and Ground Engineering**, v. 6, n. 4, p. 45, 14 dez. 2020.

2 BRATTON, D.; KENNEDY, J. **Defining a Standard for Particle Swarm Optimization**. 2007 IEEE Swarm Intelligence Symposium. **Anais**...IEEE, abr. 2007Disponível em: <http://ieeexplore.ieee.org/document/4223164/>

3 CABRAL, R. M. S. et al. Assessment by finite element modeling of pipelines with corrosion defects based on River-Bottom Profile model. **Engineering Structures**, v. 261, p. 114246, jun. 2022.

4 FERREIRA, A. D. **Análise Multiresolução e Redes Neurais Profundas para Avaliação de Dutos Corroídos**. [s.l.] Universidade Federal de Pernambuco – UFPE, 2022.

5 HAYKIN, S. **Redes Neurais: Princípios e Prática**. [s.l.] Bookman Editora, 2000.

6 HUANG, Y.; QIN, G.; HU, G. Failure pressure prediction by defect assessment and finite element modelling on pipelines containing a dent-corrosion defect. **Ocean Engineering**, v. 266, p. 112875, dez. 2022.

7 KINGMA, D.; BA, J. **Adam: A Method for Stochastic Optimization**. International Conference on Learning Representations. **Anais**...2015

8 LINS, I. D. et al. A particle swarm-optimized support vector machine for reliability prediction. **Quality and Reliability Engineering International**, v. 28, n. 2, p. 141–158, mar. 2012.

9 MONTGOMERY, D. C. .; RUNGER, G. C. **Applied Statistics and Probability for Engineers**. 7th. ed. [s.l.] Wiley, 2018.

10 MOURA, M. DAS C. et al. A competing risk model for dependent and imperfect condition–based preventive and corrective maintenances. **Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability**, v. 228, n. 6, p. 590–605, 7 dez. 2014.

11 MSC SOFTWARE CORPORATION. **Patran Complete FEA Modeling Solution**, 2015. Disponível em: <http://www.mscsoftware.com/product/patran>

12 PETROBRAS. **Malha de gasodutos e oleodutos da Transpetro é operada e monitorada com alta tecnologia**. Disponível em: <https://petrobras.com.br/fatos-e-dados/malha-de-gasodutos-e-oleodutos-da-transpetro-e-operada-e-monitorada-com-

alta-tecnologia.htm>.

13 PYTHON. **Python Documentation**. Disponível em: <https://docs.python.org/3/>.

14 QIN, G.; CHENG, Y. F.; ZHANG, P. Finite element modeling of corrosion defect growth and failure pressure prediction of pipelines. **International Journal of Pressure Vessels and Piping**, v. 194, p. 104509, dez. 2021.

15 ROSENBLATT, F. **The Perceptron, a Perceiving and Recognizing Automaton**. [s.l.] Cornell Aeronautical Laboratory, 1957.

16 ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. **Psychological Review**, v. 65, n. 6, p. 386–408, 1958.

17 RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. **Nature**, v. 323, n. 6088, p. 533–536, 1986.

18 SHI, Y.; EBERHART, R. **A Modified Particle Swarm Optimizer**. 1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360). **Anais**...IEEE, 1998Disponível em: <http://ieeexplore.ieee.org/document/699146/>

19 STRASSER, S. et al. **A New Discrete Particle Swarm Optimization Algorithm**. Proceedings of the Genetic and Evolutionary Computation Conference 2016. **Anais**...New York, NY, USA: ACM, 20 jul. 2016Disponível em: <https://dl.acm.org/doi/10.1145/2908812.2908935>

20 SUN, Y. et al. An improved grid search algorithm to optimize SVR for prediction. **Soft Computing**, v. 25, n. 7, p. 5633–5644, 20 abr. 2021.

21 WU, H. et al. Experimental and numerical studies on collapse of subsea pipelines with interacting corrosion defects. **Ocean Engineering**, v. 260, p. 112066, set. 2022.

22 XU, W.-Z. et al. Corroded pipeline failure analysis using artificial neural network scheme. **Advances in Engineering Software**, v. 112, p. 255–266, out. 2017.

ANEXO A – ROTINA COMPUTACIONAL: PSO+RNA

```
from google.colab import drive
drive.mount('/content/drive/')

#%% importar as bibliotecas
import os, re
import pandas as pd
import matplotlib.pyplot as plt
import pandas as pd
from time import time
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import tensorflow as tf
import pywt
import numpy as np
from sklearn.metrics import confusion_matrix
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.optimizers import Adam

!cp drive/MyDrive/Codigo-oficial/func_auxiliares.py .

from func_auxiliares import thick, press_eval, press_eval_ansys, eval_data,\
      set_features

# uma função pra pre processar os dados
def preprocess(filename):
    # file_AI_full_results='full_inout_AI.pkl'
    AI_full_results = pd.read_pickle('drive/MyDrive/Codigo-oficial/' + filename)
    AI_full_results=AI_full_results.head(9612) # excluindo o rpb1
    wvlt='rbio2.2'
    re = 304.8

    # eval_features coefs da wavelet
    eval_features, y_eval_list, ft_add = eval_data(wvlt, re)
    max_len_evfeat = len(max(eval_features, key = len))
    for i in range(len(eval_features)):
        if len(eval_features[i]) < max_len_evfeat:
            eval_features[i] = np.pad(eval_features[i], \
                                      int((max_len_evfeat - len(eval_features[i])) / 2), \
                                        mode = 'constant')
        if len(eval_features[i]) < max_len_evfeat:
            eval_features[i] = np.append(eval_features[i], 0)
        eval_features[i] = np.append(eval_features[i], ft_add[i])

    y_eval = pd.DataFrame(y_eval_list, columns = ['Ult Press'])
    X_eval=pd.DataFrame(eval_features,\
                        columns=[str(x) for x in range(max_len_evfeat + 1)])

    list_features, ft = set_features(AI_full_results, wvlt, re)
    max_len_feat = len(max(list_features,key=len))
    for i in range(len(list_features)):
        if len(list_features[i]) < max_len_feat:
```

```
55              list_features[i]=np.pad(list_features[i],\
56                                      int((max_len_feat-len(list_features[i]))/2),\
57                                          mode='constant')
58          if len(list_features[i]) < max_len_feat:
59              list_features[i]=np.append(list_features[i],0)
60          list_features[i]=np.append(list_features[i],ft[i])
61
62      x_data = pd.DataFrame(list_features,\
63                            columns=[str(x) for x in range(max_len_feat+1)])
64      y_val = AI_full_results['Ult Press']
65      y_val.to_csv('press_output.csv')
66      x_data.to_csv('ann_input.csv')
67
68      X_train, X_val, y_train, y_val = train_test_split(x_data,y_val,\
69                                                        test_size=0.3,\
70                                                          random_state=None)
71
72      # scale the feature data
73      scaler = MinMaxScaler()
74      scaler.fit(X_train)
75      X_train = pd.DataFrame(data=scaler.transform(X_train), \
76                             columns = X_train.columns, \
77                              index = X_train.index)
78      X_val = pd.DataFrame(data=scaler.transform(X_val), \
79                           columns = X_val.columns, \
80                             index = X_val.index)
81      X_eval = pd.DataFrame(data=scaler.transform(X_eval), \
82                            columns = X_eval.columns, \
83                              index = X_eval.index)
84
85      return X_train, X_val, y_train, y_val
86
87  # função da RNA
88  def RNA_regressao(param, dados_pre_proc):
89
90      nr_camadas, nr_neuronios, func_ativacao = param
91      nr_camadas, nr_neuronios = int(nr_camadas), int(nr_neuronios)
92
93      func_ativ_dict = {1: 'sigmoid',
94                        2: 'tanh',
95                        3: 'relu'}
96
97      func_ativ = func_ativ_dict[int(func_ativacao)]
98
99      X_train, X_val, y_train, y_val = dados_pre_proc
100     input_dim = X_train.shape[1]
101
102     model = Sequential()
103     model.add(Dense(nr_neuronios, activation = func_ativ, input_dim = input_dim))
104     for _ in range(nr_camadas - 1):
105       model.add(Dense(nr_neuronios, activation = func_ativ))
106     model.add(Dense(1, activation = func_ativ))
107
108     model.compile(loss = 'mse', optimizer = Adam(), metrics = ['mse'])
109
110     history = model.fit(X_train, y_train,
111                         batch_size = 128,
```

```
112                     epochs = 10,
113                     verbose = 1)
114
115     prediction = model.predict(X_val.values)
116     mse = mean_squared_error(y_val.values, prediction)
117
118     return mse
119
120 def pso_simulation(dados_pre_proc, iters, runs, swarmsize, phip, phig, omega):
121
122     # x = [nr_camadas, nr_neuronios, func_ativacao]
123
124     lb = np.asarray([1, 1, 1])
125     ub = np.asarray([5, 15, 3])
126
127     vhigh = 0.1*(ub - lb)
128     vlow = - vhigh
129
130     fig_history, ax_history = plt.subplots(1,1)
131     ax_history.set_title(r'Histórico do Erro Quadrático Médio')
132     ax_history.set_xlabel('Iteração')
133     ax_history.set_ylabel('EQM')
134
135     S, N = swarmsize, len(ub)
136     optim_trace = []
137     cost_trace = []
138     colunas = []
139
140     for run in range(runs):
141         start = time()
142         colunas.append('run ' + str(run + 1))
143         # inicializar o swarm
144         x = np.zeros([S, N])
145         v = np.zeros_like(x) # velocidades
146         p = np.zeros_like(x) # melhores posições
147         fx = np.zeros(S)     # valor atual de f de cada partícula
148         fp = np.ones(S) * np.inf # melhor valor de cada partícula
149         pen = np.zeros(S)
150         peng = 0
151         g = [] # melhores posições do swarm
152         fg = np.inf # valor inicial da melhor posição
153         cost_history = []
154         # n_feasible = 0 # contador de soluções viáveis
155
156         # inicializar as posições das partículas
157         x = lb + np.random.rand(S, N) * (ub - lb)
158         x = np.round(x)
159         ind_act = x[:, 2] > 3
160         x[:,2][ind_act] = 3
161
162         it, iter_opt = 0, 0
163
164         # inicializando a função objetivo para cada partícula
165         for i in range(S):
166             #pen = it**2 * np.sum((x[i] - ub) / (ub - lb))
167             fx[i] = RNA_regressao(x[i], dados_pre_proc)
168
```

```
169         # armazenar a melhor posição de cada partícula
170         i_update = (fx < fp)
171         p[i_update, :] = x[i_update, :].copy()
172         fp[i_update] = fx[i_update]
173 
174         # atualizar a posição do swarm
175         i_min = np.argmin(fp)
176         if fp[i_min] < fg:
177             fg = fp[i_min]
178             g = p[i_min, :].copy()
179         else:
180             g = x[0, :].copy()
181 
182         cost_history.append(fg)
183         print('\r run {:} => Iteration {:}: {:} {:}'.format(run, it, fg, g),\
184                end='', flush=True)
185 
186         # inicializar a velocidade de cada partícula
187         v = vlow + np.random.rand(S, N) * (vhigh - vlow)
188 
189         # iterar até satisfazer condição de parada
190         while it < iters:
191             rp = np.random.uniform(size = (S, N))
192             rg = np.random.uniform(size = (S, N))
193 
194             # atualiza as velocidades
195             v = v + phip * rp * (p - x) + phig * rg * (g - x)
196             # atualiza as posições
197             x = x + omega * v
198             # corrigir as violações dos limites
199             maskl = x < lb
200             masku = x > ub
201             # desse jeito trunca no limite mínimo
202             x = x * (1 - maskl) + lb * maskl
203             x = np.round(x)
204             ind_act = x[:, 2] > 3
205             x[:,2][ind_act] = 3
206             # desse jeito trunca no limite mínimo e máximo
207             # y= x*(~np.logical_or(maskl,masku))+min_bound*maskl+max_bound*masku
208 
209             for i in range(S):
210                 # + sum((it + 1) * maskl[i, :] * (abs(x[i, :] - lb) / lb)**2)
211                 pen[i] = sum((it + 1) * masku[i, :] * abs(x[i, :] - ub) / ub)
212                 fx[i] = RNA_regressao(x[i], dados_pre_proc) + pen[i]
213 
214             # fx[i] = fx[i] + pen(x[i,:], i, max_bound, min_bound)
215             # armazenar a melhor posição de cada partícula
216             i_update = (fx < fp)
217             p[i_update, :] = x[i_update, :].copy()
218             fp[i_update] = fx[i_update]
219 
220             # comparar a melhor posição do swarm com o melhor global
221             i_min = np.argmin(fp)
222             p_min = p[i_min, :].copy()
223             fg_old = fg
224 
225             if fp[i_min] < fg:
```

```
                g = p_min.copy()
                fg = fp[i_min]
                peng = pen[i_min]
                iter_opt = it + 1

            # critério de parada
            # basicamente, ele diz que se não tiver mudança na função objetivo em
            # 10% das iterações consecutivas, então o algoritmo para.
            If (it - iter_opt) > int(0.1 * iters):
                it = iters - 1

            it += 1
            cost_history.append(fg)
            print('\r run {:} => Iteration {:}: {:} {:}'.format(run + 1, it, fg, g),\
                  end='', flush=True)

        ax_history.plot(cost_history, label = r'run {}'.format(run + 1))
        # ax_history.legend(loc = 'best')
        tempo = time() - start
        print(' | tempo = %.2f s' % (tempo))

        func_ativ_dict = {1: 'sigmoid',
                          2: 'tanh',
                          3: 'relu'}

        optim_trace.append([run + 1, int(g[0]), int(g[1]), func_ativ_dict[int(g[2])],\
                            fg - peng, peng, iter_opt, tempo])
        cost_trace.append(cost_history)

    df_out = pd.DataFrame(optim_trace, columns = ['run', 'camadas',\
                                                  'neuronios', 'ativação',\
                                                     'MSE', 'penalização',\
                                                         'iteração', 'tempo [s]'])
    print('\n')
    print(df_out.to_string(index = False))
    fig_history.savefig('msq history', dpi = 300)

    # ajeitando os dados de saída do cost
    len_max = 0
    for el in cost_trace:
      if len(el) > len_max: len_max = len(el)

    for el in cost_trace:
      len_aux = len_max - len(el)
      for _ in range(len_aux):
        el.append(0)

    # print(cost_trace)
    df_cost = pd.DataFrame(np.vstack(cost_trace).T, columns = colunas)

    # salvando os principais outputs em dataframe
    path_out = 'drive/MyDrive/Codigo-oficial/outputs.xlsx'
    path_cost = 'drive/MyDrive/Codigo-oficial/cost_history.xlsx'

    with open(path_out, 'wb') as f_out:
      df_out.to_excel(f_out)

```

```
283     with open(path_cost, 'wb') as f_cost:
284       df_cost.to_excel(f_cost)
285 
286 #%% lendo os dados pré processados
287 
288 filename = 'full_inout_AI.pkl'
289 
290 # dados_pre_proc = [X_train, X_val, y_train, y_val, input_func, feat_cols]
291 dados_pre_proc = preprocess(filename)
292 
293 # msq = RNA_regressao([3, 10, 1], dados_pre_proc)
294 #%%
295 iters = 100
296 runs = 10
297 swarmsize = 30
298 phip, phig = 2.05, 2.05
299 omega = 0.729843
300 
301 pso_simulation(dados_pre_proc, iters, runs, swarmsize = swarmsize,
302                phip = phip, phig = phig, omega = omega)
```

ANEXO B – ROTINA COMPUTACIONAL: FUNÇÕES AUXILIARES

```
#%% imports
import os
import pandas as pd
import pywt
import numpy as np

#%% funções auxiliares

re=304.8
t=[14.56, #0391
   12.58, #1373
   8.22, #3601
   6.27, #5882
   6.51, #6515
   6.51, #6598
   6.55, #7075
   6.45, #7312
   6.33, #7528
   6.63, #7613
   6.45, #8449
   6.39, #9436
   6.55] #rbp1

def thick(fname):
    thickness={'0391':t[0],
    '1373':t[1],
    '3601':t[2],
    '5882':t[3],
    '6515':t[4],
    '6598':t[5],
    '7075':t[6],
    '7312':t[7],
    '7528':t[8],
    '7613':t[9],
    '8449':t[10],
    '9436':t[11],
    'rbp1':t[12]}
    return thickness.get(fname,'Invalid name')

def press_eval(fname):
    press_eval={'0391':32.8,
    '1373':27.0,
    '3601':18.8,
    '5882':14.6,
    '6515':14.9,
    '6598':14.6,
    '7075':15.0,
    '7312':14.7,
    '7528':14.5,
    '7613':15.0,
    '8449':14.6,
    '9436':14.6,
    'rbp1':9.7}
    return press_eval.get(fname,'Invalid name')
```

```
55
56 def press_eval_ansys(fname):
57     press_eval={'0391':30.2,#30.2
58     '1373':25.13,#25.13
59     '3601':17.45,#17.45
60     '5882':13.94,#13.94
61     '6515':13.94,#13.94
62     '6598':13.68,#13.68
63     '7075':14.43,#14.43
64     '7312':13.82,#13.82
65     '7528':13.62,#13.62
66     '7613':14.43,#14.43
67     '8449':13.76,#13.76
68     '9436':13.72,#13.72
69     'rbp1':8.63}#8.63
70     return press_eval.get(fname,'Invalid name')
71
72 def eval_data(wvlt,re):
73     files=[]
74     feat_eval=[]
75     y_eval=[]
76     ft_add=[]
77     directory = '/content/drive/MyDrive/Codigo-oficial/adjust_dist'
78     for filename in os.listdir(directory):
79         if filename.endswith(".txt"):
80             files.append(filename)
81     for datas in files:
82         t_data=thick(datas[0:4])
83         data = pd.Series(np.loadtxt(directory+'/'+datas,delimiter='\t')[:,1])
84         longcoord = pd.Series(np.loadtxt(directory+'/'+datas,delimiter='\t')[:,0])
85         DWTcoeffs = pywt.wavedec(np.asarray(list(data))/t_data,wvlt,axis=-1)
86         feat_eval.append(DWTcoeffs[0])
87         y_eval.append(press_eval(datas[0:4]))
88         ft_add.append(max(list(longcoord))/(re*2*t_data)**0.5)
89     return feat_eval,y_eval,ft_add
90
91 def set_features(AI_full_results,wvlt,re):
92     list_features = []
93     list_coeffs=[]
94     ft=[]
95     for index, row in AI_full_results.iterrows():
96         t_case=thick(row['Case Name'].split('_')[0][0:4])
97         list_thicks=row['Rand Thicks']
98         # Normalizing Remain Thickness. list_thicks[1]/t_case -> Testar com e sem
99         DWTcoeffs = pywt.wavedec(list_thicks[1]/t_case,wvlt,axis=-1)
100        list_coeffs.append(DWTcoeffs)
101        # Test get features if problems with atual method
102        features = DWTcoeffs[0]
103        list_features.append(features)
104        ft.append(max(list_thicks[0])/(re*2*t_case)**0.5)
105    return (list_features, ft)
```