UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

RENAN MENDES FROTA

PHYSICS-INFORMED NEURAL NETWORKS APLICADAS À SIMULAÇÃO DE BIORREATORES:

UM ESTUDO DE CASO COM A PRODUÇÃO DE ÁCIDO LÁTICO

FORTALEZA
2022

RENAN MENDES FROTA

PHYSICS-INFORMED NEURAL NETWORKS APLICADAS À SIMULAÇÃO DE BIORREATORES:

UM ESTUDO DE CASO COM A PRODUÇÃO DE ÁCIDO LÁTICO

Dissertação de Mestrado submetida à coordenação do Programa de Pós-Graduação em Engenharia Química, do Centro de Tecnologia, da Universidade Federal do Ceará, como requisito parcial para obtenção do título de Mestre em Engenharia Química. Área de concentração: Processos Químicos e Bioquímicos. Área de concentração: Processos Químicos e Bioquímicos.

Orientador: Prof. Dr. Ivanildo José da Silva Júnior
Coorientador: Prof. Dr. Amaro Gomes Barreto Júnior

FORTALEZA

2022

RESUMO

Redes Neurais com Embasamento Físico (PINN, do inglês *Physics-Informed Neural Network*) são uma técnica relativamente recente capaz de simular sistemas com embasamento físico com Redes Neurais (NN, do inglês *Neural Network*). Para isso, os dados necessários para otimização da NN são gerados a partir de equações advindas de modelos matemáticos. PINNs podem substituir métodos numéricos e outras formas de resolver sistemas de equações empregados em Engenharia Química, como processos de produção, purificação e reações químicas. Biorreatores são um desafio natural para PINNs por sua complexidade intrínseca e, portanto, estratégias que busquem reduzir a *loss* (erro) e melhorar a convergência dos resultados à solução correta precisam ser investigados. O processo de produção do ácido lático foi escolhido como um estudo de caso para investigar técnicas de otimização de PINNs aplicados à simulação de biorreações devido à abundância de literatura científica relacionada, além de suas grandes importâncias técnica e econômica. Os valores ótimos para o tamanho e largura de camada da NN, pesos da função *loss* e fatores (*scalers*) de adimensionalização foram determinados para a produção de ácido lático por *Lactobacillus casei* em um reator batelada. A adimensionalização do tempo resultou em um erro maior em praticamente todos os casos. O sistema proposto foi capaz de reproduzir com grande fidelidade (comparado com um método numérico alternativo e dados experimentais) o processo estudado, alcançando valores de loss inferiores a 10^{-4} em volume constante (reator batelada). Contudo, o modelo se demonstrou deficiente ao reproduzir modelos com variação de volume (CSTR e batelada alimentada). A performance da simulação do CSTR e batelada alimentada pode ser melhorada aplicando X_M (concentração de biomassa inibitória) como o fator de adimensionalização da concentração da biomassa, e quando o V_{max} (volume máximo do reator) como o fator de adimensionalização do volume. A concentração de biomassa e o volume de líquido no reator foram determinadas como as variáveis mais relevantes para a redução da *loss* e consequente geração de resultados mais precisos.

Palavras-chave: PINN; Ácido Lático; Simulação; Biorreator; Machine Learning

ABSTRACT

Physics-Informed Neural Network (PINN) is a relatively new technique capable of simulating physics-based systems with Neural Networks (NN) using mathematical models equations to generate the necessary data for NN optimization. PINNs can replace numerical methods and other techniques in solving systems of equations used by Chemical Engineering, such as production and purification processes, and chemical reactions. Bioreactors pose a challenge for PINNs because of their intrinsic complexity, and thus strategies that improve loss reduction and solution convergence must be studied. Lactic acid production was chosen as a case study for the investigation of PINN optimization strategies for the simulation of bioreactions because of its great economic and technological importance and abundant scientific literature available. The optimal layer deepness and wideness, loss weights and nondimensionalization factors were determined for a batch reactor production of lactic acid by Lactobacillus casei. The nondimensionalization of time increased loss in all cases. PINNs were able to reproduce with great fidelity numerical results and experimental data in constant volume (batch reactor) with loss values < 10-4, but showed poor performance in models with volume variation (CSTR and fed-batch). CSTR and fed-batch performance improved when X_M (inhibitory biomass concentration) was used as a nondimensionalization scaler of biomass concentration and V_{max} (reactor maximum volume) as a nondimensionalization scaler of volume. Biomass concentration and reactor liquid content volume were also the most relevant variables to reduce loss values and impact significantly in the final solution.

Keywords: PINN; Lactic Acid; Simulation; Bioreactor; Machine Learning

LISTA DE FIGURAS

LISTA DE GRÁFICOS

LISTA DE TABELAS

LISTA DE QUADROS

SUMÁRIO

1 INTRODUÇÃO

Há aproximadamente 70 anos, a Indústria Química foi a primeira das Indústrias da sociedade civil a empregar o uso de computadores, ainda na década de 1950. Desde então, os custos para a aquisição de computadores caiu consideravelmente, enquanto que a capacidade de processamento e armazenamento aumentou de forma expressiva (JOHNS, 2011). Os custos com a aquisição de computadores e softwares, portanto, se justificam não somente por permitirem economizar ao escolher projetos ótimos e realizar centenas de cálculos que seriam muito dispendiosos e praticamente impossíveis de outra forma.

Com a introdução de programas computacionais mais complexos, se faz mais evidente a necessidade de melhorar a padronização e capacidade de leitura do código, para aumentar a reutilização, reduzir erros e falhas de lógica e permitir a comunicação entre profissionais das diferentes áreas abordadas pela Engenharia Química. Dentre as principais estratégias presentes, destacam-se as melhorias focadas em programação (voltadas a testar o código, separação em pastas e modelos e padrões de estrutura) e em design (voltados à nomeação adequada de classes, funções e variáveis, além da organização de bancos de dados e interfaces de comunicação entre diferentes programas).

Para muitos autores, as estratégias de melhorias focadas em design passam a ganhar destaque a partir da década de 1970, com o nascimento o conceito de Arquitetura da Informação. Focado em organizar informações e responsável por grandes contribuições na área de interação humano-computador, como interfaces gráficas e processadores de texto, o PARC (do inglês *Xerox Palo Alto Research Center*, ou Centro de Pesquisas da Xerox em Palo Alto) foi um dos grandes responsáveis pelo seu desenvolvimento (REMINI; ROSATI, 2011). A Arquitetura da Informação tem por objetivo a organização e clareza, tornando dados complexos mais claros e simples. Em resumo, é responsável por proporcionar maior entendimento, melhor interação entre pessoas e computadores, aumentar a clareza na expressão e interpretação de ideias e organizar a informação. O termo ficou muito mais popular com o advento da WWW (do inglês *World Wide Web*, a sigla representa a rede mundial de computadores interligados), pela necessidade de

tornar informações claras para diversas pessoas, com diferentes perfis acadêmico, pessoal e profissional e, por fim, democratizar a internet.

As estratégias focadas em programação existem desde o início da ciência da computação em si. O código precisa ser conciso, claro e legível, mas sobretudo testável e sempre que possível testado. Códigos confusos, feitos sem um planejamento prévio de arquitetura e sem seguir padrões (mesmo que estabelecidos exclusivamente para aquela aplicação) podem dificultar expressivamente o acesso futuro ao software, e em alguns casos é dispendioso até mesmo para os autores originais realizarem a manutenção e expansão de funcionalidade. Em alguns casos, o código fica tão intrincado e mal organizado que é impossível de torná-lo limpo. Com o passar do tempo, a produtividade da equipe que trabalha com esse software problemático se aproxima assintoticamente de zero (MARTIN, 2009). São nesses momentos que ocorrem projetos de refatoração (reescrita e reorganização de parte do código) ou o projeto é reescrito por completo. Tais atividades implicam custos financeiros e de tempo e, portanto, reduzi-las é uma prioridade para a Engenharia.

O LA (Ácido Lático, do inglês *Lactic Acid*) é uma molécula de grande interesse econômico, com aplicações nas indústrias alimentícia, farmacêutica, têxtil e cosmética (KOMESU; MACIEL; FILHO, 2017; LÓPEZ-GÓMEZ et al., 2019). É ainda empregado na síntese de PLA (Poli-Ácido Lático), matéria-prima de polímeros comumente usados em impressões 3D. Sua produção pode se dar por via fermentativa ou síntese química, sendo a fermentativa usada em cerca de 90% dos casos. Dentre as principais linhas de pesquisa recentes relacionadas ao LA, destacam-se as pesquisas por diferentes fontes de carbono para a via fermentativa, a redução dos custos de produção e a redução dos custos de purificação.

Com cada vez mais conhecimento, modelos e dados empíricos disponíveis, a necessidade de organização e classificação adequada da informação para lidar com tamanha quantidade de conteúdo é fundamental. Embora sejam aplicados com frequência na Indústria de IT (Tecnologia da Informação, do inglês *Information Techonology*) (ROSENFELD; MORVILLE; ARANGO, 2015), esses princípios são frequentemente ignoradas em programas de engenharia química básica. São esses os princípios os responsáveis por permitir a classificação adequada de milhões de arquivos em aplicativos de música, gerenciar milhares de usuários em aplicativos para dispositivos móveis e garantir a funcionalidade tanto do código como da apresentação ao usuário (interface). Sua aplicação orientada à Engenharia Química

visa promover o melhor uso dos recursos disponíveis, facilitar a leitura e manutenção de código existente e proporcionar maior clareza, reduzindo o tempo de correção de ineficiência computacional ou falhas de lógica e, consequentemente, aumentar o tempo disponível para o estudo e descoberta de novas tecnologias, materiais e modelos em Engenharia Química.

A simulação de processo de Engenharia Química frequentemente emprega métodos numéricos tradicionais, como Ruge-Kutta e Euler, e pode se beneficiar significativamente das estratégias de organização e modularização citadas. Contudo, essas técnicas de solução numérica ainda estão sujeitas a instabilidades e pode ser bastante sensível à estratégia de discretização numérica empregada. A precisão pode ser aumentada empregando malhas mais finas, com mais pontos de discretização, mas isso também pode gerar erros no sistema e/ou aumentar consideravelmente o esforço computacional necessário. Além disso, sua implementação requer conhecimento técnico tanto em arquitetura de software e programação quanto em modelagem matemática, além dos conhecimentos específicos da área de interesse. O uso de softwares proprietários, de código fechado ou não acessíveis como Matlab, Aspen Plus e COMSOL Multiphysics é empregado (JANOSKA; BUIJS; VAN GULIK, 2023; LI; XU, 2023) por poder reduzir o esforço necessário, ou tornar o fluxo de trabalho mais agradável ou, ainda, melhorar os resultados obtidos ao fornecer técnicas aprimoradas. Dependendo do modelo a ser realizado, o custo experimental pode ser relativamente elevado e até mesmo proibitivo.

Redes Neurais Artificiais (ANN, do inglês Artificial Neural Network) são aproximadores universais, capazes de representar as mais diversas funções de classificação e regressão numérica. Muitas vezes conseguem representar fenômenos extremamente complexos, mesmo com uma quantidade relativamente reduzida de parâmetros. Contudo, podem ser computacionalmente intensivas e normalmente necessitam de uma grande quantidade de dados (*big data*). Essa aquisição de uma grande quantidade de dados normalmente é muito custosa e, por vezes, proibitiva. Uma grande vantagem das ANNs é que, apesar de ser alto custo para construção e otimização do modelo (treino), o uso do modelo já pronto é relativamente simples e pode ser feito em computadores menos robustos.

Redes Neurais com Embasamento Físico (PINN, do inglês Physics-Informed Neural Networks) são uma abordagem relativamente recente, que emprega ANN

para a solução de sistemas de equações diferenciais, em alternativa aos tradicionais métodos numéricos (RAISSI; PERDIKARIS; KARNIADAKIS, 2019; SANTANA et al., 2022). Nessa abordagem, a capacidade de aproximação universal das ANN é empregada para a solução do sistema de equações, e a grande quantidade de dados necessárias para treino da ANN é contornada ao se empregar modelos matemáticos. Em resumo, a rede é responsável por predizer as variáveis dependentes (saídas da rede neural) com base nas entradas (variáveis independentes). Desse modo, a grande capacidade de predição e solução de problemas das ANN pode ser aplicada à solução de equações diferenciais, ao mesmo tempo em que contorna adequadamente a limitação de dados. Para a otimização de um PINN, o erro (*loss* total) é reduzida ao balancear os pesos entre os diferentes neurônios da rede neural. Essa técnica é muito relevante em casos onde a solução numérica é extremamente complicada ou custosa.

Reações biológicas são naturalmente um desafio para PINNs, por apresentarem com frequência dependência múltipla entre diversas variáveis e suas respectivas derivadas. Contundo, o uso de PINNs abre caminho para grandes inovações na simulação de bioprocessos, pois pode permitir uma maior precisão e adequação em diversos modelos, bem como facilitar a reprodução de resultados através de compartilhamento dos modelos já otimizados. Como é uma técnica relativamente recente, ainda são necessários esforços no sentido de entender melhor o funcionamento e sua otimização aplicada a bioprocessos já que, embora seja muito relevante, nem sempre é capaz de resolver adequadamente os sistemas simulados (LI; XU, 2023; SANTANA et al., 2022)

O trabalho se propôs a aplicar os princípios de programação e arquitetura da informação citados para a simulação através de PINN da produção de LA por *L. casei* conforme descrito por (ALTIOK; TOKATLI; HARSA, 2006). Para isso, foram estudados diversos parâmetros, hiperparâmetros e configurações em reatores operando nos regimes batelada, batelada-alimentada e contínuo. A biblioteca DeepXDEm e, Python, foi empregada para a criação dos modelos PINN (LU et al., 2021). Foram determinadas as condições ótimas para produção de PINNs que pudessem representar adequadamente o processo fermentativo descrito, bem como discutidas as diferenças entre os modelos de operação empregados e seu impacto na capacidade dos PINNs de representar ou não adequadamente as concentrações e volume do reator.

2 OBJETIVOS

2.1 Objetivo Geral

O objetivo geral deste trabalho é avaliar diferentes estratégias para o emprego de PINNs (do inglês Physics-Informed Neural Networks, Redes Neurais com Embasamento Físico) em simulações de biorreatores e reações biológicas.

2.2 Objetivo Específicos

- Simular o funcionamento de um reator batelada para a produção de LA por *Lactobacillus casei* através de PINN;
- Comparar o resultado da simulação por PINN com métodos numéricos tradicionais e dados experimentais;
- Determinar a aplicabilidade do uso de PINNs para a simulação em regimes estacionários e transientes e a relevância de cada variável estudada

3 REVISÃO BIBLIOGRÁFICA

3.1 Arquitetura da Informação (IA)

A Arquitetura da Informação (IA, do inglês *Information Architecture*) é a área da ciência que se dispõe a propor melhores maneiras de classificar, expor e relacionar informações. Para muitos autores, o seu nascimento ocorre nos esforços em desenvolvimento de interfaces humano-computador realizados pela equipe de pesquisadores do PARC (do inglês *Xerox Palo Alto Research Center*, ou Centro de Pesquisas da Xerox em Palo Alto) (REMINI; ROSATI, 2011). Assim como arquitetos buscam empregar linhas e maquetes para propor construções, o arquiteto de informação busca propor maneiras de dar à informação mais clareza e objetividade. O intuito é que as informações sejam acessíveis, organizadas de maneira lógica e clara e que proporcionem o melhor entendimento por parte dos humanos (usuários) que as usem, quer seja em aplicativos móveis quer seja profissionalmente em contexto de engenharia.

A IA se volta, então, à necessidade de encontrar o ponto ótimo de acessibilidade (encontrar) e compreensibilidade (entender) as informações presentes (ROSENFELD; MORVILLE; ARANGO, 2015). Para otimizar a acessibilidade, é possível definir identificadores únicos, relacionamento entre diferentes dados e documentação adequada do contexto da informação. Para otimizar a compreensibilidade, é necessário manter padrões bem documentados, dar preferência a nomes autoexplicativos que requiram pouco esforço cognitivo para compreensão da representação. Por exemplo, uma variável nomeada “reator1” é muito mais expressiva que uma nomeada como “r1”, tornando mais fácil a leitura do código por terceiros e o encontro de erros de lógica. A importância da legibilidade é tamanha que alguns autores afirmam que, mesmo que uma certa linguagem de programação apresentasse performance (velocidade de execução) 10 vezes menor do que outra, caso a execução ainda fosse rápida o suficiente para a tarefa em

questão e ela possuísse características como maior legibilidade, seria uma boa escolha (HETLAND, 2014).

A internet representou um grande marco da civilização humana. A possibilidade de comunicação instantânea e capaz de transcender fronteiras foi, sem dúvidas, responsável por grande parte dos avanços científicos das últimas décadas, bem como pela produção de uma grande quantidade de descobertas, produção de dados e publicação de artigos científicos. Em resumo, foi responsável pela produção de uma grande quantidade de informações úteis à Engenharia Química. Para que essa imensidão de dados sirva como base para novas descobertas e futuras pesquisas minimizando os impactos negativos (dificuldade em localizar recursos, referências a dados incorretas e dificuldades em verificar e corrigir tais erros), organizar a abundância de informações se tornou uma tarefa crucial.

3.1.1 Aplicação em Química e Engenharia Química

Grandes quantidades de dados são produzidos todos os anos em Engenharia Química, seja em artigos publicados na literatura ou em projetos privados. Diversos simuladores empregam, ainda, o uso de pacotes de dados, que se valem de informações (como dados empíricos, equações e modelos) para prover mais recursos aos usuários. Todas essas informações são armazenadas em bancos de dados. Bancos de dados relacionais como o *PostgreSQL* são responsáveis pelo armazenamento de grandes quantidades de dados que estão relacionados uns com os outros. Por exemplo, relacionando fórmulas químicas de substâncias aos seus números CAS através de uma tabela de relacionamento. São bancos de dados robustos, aprovados por décadas de uso em diferentes indústrias e altamente normalizados – o que significa que praticamente sempre os dados de um mesmo tipo são armazenados em um único registro e dali referenciados, e não duplicados (DEBARROS, 2018). As informações armazenadas em tais bancos têm, no geral, estrutura bem definida e com pouca probabilidade de mudança nessa estrutura. Dentre suas principais vantagens estão a garantia de uma estrutura bem definida, bem como a não duplicação de dados redundantes. Bancos de dados não relacionais, como o *Firebase* empregam uma alternativa diferente: costumeiramente

são desnormalizados (o que significa que há maior duplicação de dados em diferentes partes do banco de dados e, como consequência, uma atualização de um único registro pode requerer a atualização em vários documentos ou inserções no banco) (SULLIVAN, 2015). Como vantagens, apresenta grande flexibilidade, sobretudo para dados que podem mudar ao longo do tempo, além de permitir uma horizontalização mais fácil dos serviços de hospedagem.

3.1.1.1 Classificação e Identificadores Únicos

Seja em bancos de dados relacionais ou não relacionais, é necessário referenciar informações, ora advinda do próprio banco de dados, ora de terceiros. Para tanto, se faz o uso de identificados únicos ou IDs. Os identificadores únicos mais comuns são números sequenciais (como 1, 2, 3...) ou identificadores do tipo UUID(Identificador Universal Único, do inglês *Universally Unique Identifier*).

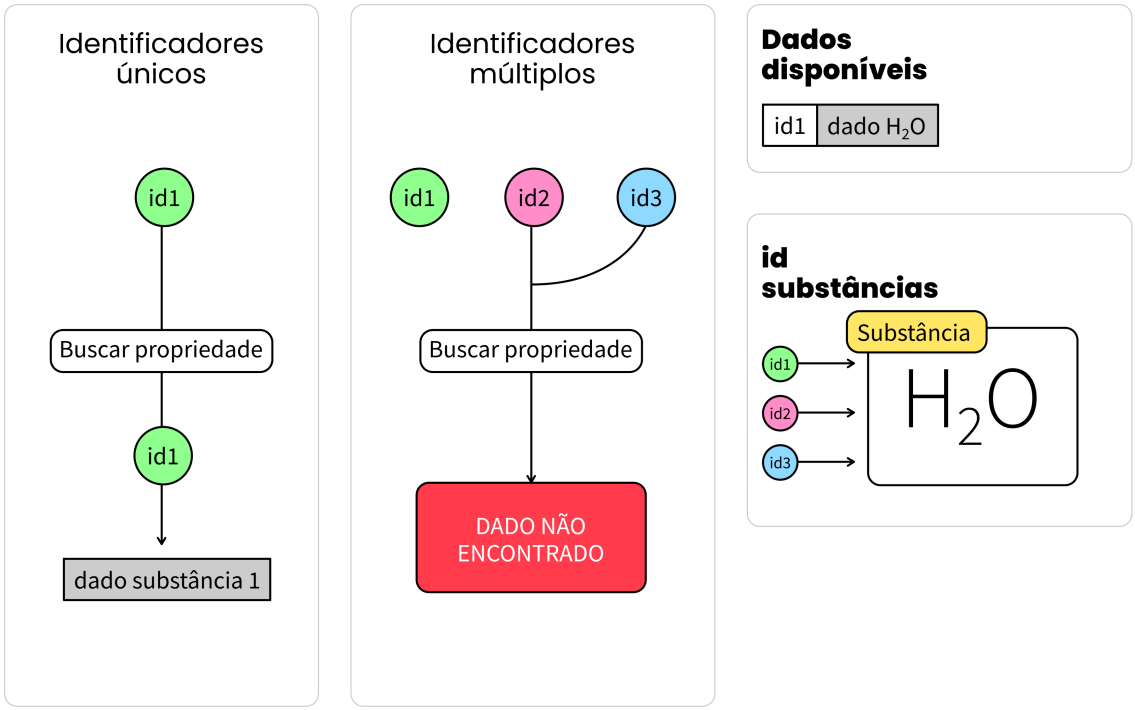
Dentre identificadores únicos disponíveis para a identificação de substâncias, podem ser empregados tanto os nomes científico ou seguindo a nomenclatura IUPAC (União Internacional de Química Pura e Aplicada, do inglês *International Union of Pure and Applied Chemistry*). Existem ainda registros especificamente criados para representar substâncias de forma precisa e muito específica, como o SMILES (Sistema de registro de moléula em formato de linha, do inglês simplified molecular-input line-entry system) e o OpenSMILES (versão do SMILES de código aberto), o CAS RN (Número de Registro do CAS, do inglês CAS Register Number), O InChI (Identificador Química Internacional IUPAC, do inglês IUPAC International Chemical identifier) e a SubID (Identificação de Substância, do inglês Substance Identification), que é mais voltado a substância com efeitos farmacêuticos.

Registros de identidade como o nome segundo a norma da IUPAC ou o nome científico revelam também a natureza ou estrutura da substância descrita, mas podem existir duas ou mais formas de identificar a mesma substância (o que pode levar a duplicatas no banco de dados ou a dificuldade de encontrar dados para uma substância por ela ter sido registrada sobre um outro identificador). O CAS e o InChI buscam resolver os problemas das alternativas descritivas usando identificadores

únicos, que normalmente são atribuídos pela ordem em que a substância foi adicionada ao banco de dados. Embora não seja rigorosamente necessário o uso de apenas um identificador por substância química, tal restrição é bastante conveniente e prática por limitar possibilidade de falhas de comunicação ou compreensão (HELLER et al., 2015). A figura 1 ilustra uma dessas falhas. A substância de interesse (água, H_2O) possui três identificadores: “id1”, “id2” e “id3”. Contudo, os dados do banco de dados fazem referência a apenas um: “id1”. Assim, as consultas em busca de propriedades da água em um programa empregando as ids 2 e 3, embora façam referência à mesma substância, não encontrarão nenhum dado no banco de dados. Quando um único identificador é empregado, além de reduzir a possibilidade de tais desencontros de informações, a legibilidade também é favorecida, por identificar a mesma coisa (no caso, uma substância química) sempre com o mesmo valor de “id”.

Para classificação de seres vivos de interesse em Engenharia Química, sobretudo microrganismos como bactérias e leveduras, ainda não existem alternativas tão robustas e amplamente aceitas como o CAS e o InChI. Uma das possibilidades é o uso do nome científico da espécie já que, ao contrário do nome químico, normalmente só há um. Outra possibilidade é o uso de um identificador próprio (número inteiro sequencial) para representação daquela espécie dentro do banco de dados, e uma relação separada entre cada identificador e o nome científico da espécie.

Figura 1 - Desvantagens do uso de identificadores múltiplos



Fonte: Autoria Própria (2022)

3.1.1.2 Ensino

O mundo inteiro já apresentava uma grande transição para os meios digitais, mas a pandemia de COVID-19 acelerou o processo bruscamente. Na educação o fenômeno também foi observado, e a importância de uma didática clara, concisa e versátil para se adaptar a alunos de diferentes *backgrounds* (MCCORMACK; CALDWELL, 2022), que já era uma necessidade do sistema de ensino brasileiro, também se agravou com a pandemia. É necessário aumentar o engajamento do aluno, melhorar a compreensão do conteúdo e prover os meios necessários para que o estudante possa construir o conhecimento.

Nas disciplinas relacionadas a modelagem ou programação, duas dificuldades adicionais existem: a variação de sintaxe entre linguagens de programação (que pode servir de barreira para alunos a depender de sua experiência) e a padronização do código. A padronização de nomenclatura de variáveis,

equipamentos, substâncias e outros objetos e valores pode levar a um código mais legível, menos propício a *bugs* (falhas computacionais, do inglês *bug* [inseto]). Esse aumento na legibilidade, por sua vez, proporciona mais fácil compreensão do assunto abordado, bem como reduz o desconforto de alunos pouco acostumados com a linguagem de programação empregada. Em resumo, a aplicação de princípios de Arquitetura de Informação ao código, no ensino de ciências e engenharias, permite que mais tempo seja dedicado às atividades de compreensão dos fenômenos estudados, e menos tempo seja necessário às atividades de programação em si.

3.2 Python

A linguagem Python é frequentemente adotada em instituições de ensino e pesquisa por sua simplicidade, fácil leitura e forte tipagem dinâmica (o que não requer a declaração de tipos como *int* [inteiro] ou *float* [dígito numérico de ponto flutuante]). Foi por causa desses fatores que a linguagem se popularizou no ensino de ciências e engenharias, somados ao fato de possuir código aberto e ser gratuita. E foi por causa dessa popularização que bibliotecas de grande robustez, como a *Pyplot* e *Numpy* foram criadas em Python. Assim, nasceu um ecossistema muito propício ao seu uso em ciências e engenharia, consolidando o Python no meio acadêmico. Os arquivos Python são finalizados em "*.py".*

3.2.1 Tipagem

A tipagem de variáveis diz respeito a declarar cada variável com um tipo próprio, e além de trazer mais clareza ao código, pode ser responsável por impedir erros ao acessar métodos ou variáveis internas indisponíveis para determinado objeto. São chamadas linguagens fortemente tipadas aquelas que o tipo de cada variável e expressão é conhecido pelo programa mesmo sem a execução do código.

Esse tipo de recurso permite que erros que outrora ocorreriam em tempo de execução (*runtime errors*) sejam capturados ainda em tempo de compilação. Em resumo, uma linguagem mais tipada perde um pouco de flexibilidade em função de mais garantias sobre os acessos e tipos de objetos, variáveis e funções, e torna o código, em geral, mais seguro. São chamadas linguagens estaticamente tipadas aquelas em que o tipo da variável precisa ser declarado antes que ela possa ser usada.

O uso de tipagem forte pode limitar os argumentos passados em funções, tornando a execução da função mais compreensível e didática, ao mesmo tempo em que limita erros de lógica e até mesmo erros de digitação, uma vez que tipos incorretos serão apontados como tal pelo programa. Normalmente, quando um tipo incorreto é aplicado a uma função, um alerta ou erro é emitido pela própria IDE (Ambiente de Desenvolvimento Integrado, do inglês *Integrated Development Environment*) e o código não pode ser executado. A Figura 2 - Exemplo de tipagem em função em linguagem Dart exibe uma função *multiplicarDouble*, em linguagem *Dart* e IDE VScode, que recebe como argumento uma variável do tipo *double* e retorna uma variável também do tipo *double*. *O* erro causado por uso do tipo errado (*int*) na função é evidenciado pelo destaque em vermelho abaixo do argumento *numeroInt*, na linha 9. Embora seja um exemplo simples, o mesmo princípio poderia ser usado para, ainda antes da execução do código, prevenir erros ao se tentar executar reações em um equipamento que não suportasse o recurso. Em sumo, o emprego adequado de tipagem permite a redução de erros de digitação ou de inserção de argumentos inadequados em funções, facilitando a testagem e reduzindo a possibilidade de erros nesses aspectos.

Figura 2 - Exemplo de tipagem em função em linguagem Dart

example.dart > ...

```dart
1   double multiplicarDouble(double numero) {
2     return numero * 2;
3   }
4
5   final double numeroDouble = 3.0;
6   final int numeroInt = 5;
7
8   final resultadoDouble = multiplicarDouble(numeroDouble);
9   final resultadoInt = multiplicarDouble(numeroInt);
10
```

Fonte: Autoria Própria (2022)

O Python é uma linguagem que possui tipagem forte mas dinâmica. O fato de ser dinâmica significa que o tipo de cada variável não precisa ser declarado. A informação é obtida, quando necessário, a partir do objeto ao qual a variável faz referência. O mesmo é válido para funções. A partir da versão 3, a linguagem permitiu a declaração de variáveis e do tipo de retorno de funções, mas o intuito é tão somente indicar os tipos esperados (melhorar a usabilidade). A linguagem não faz nenhuma restrição quanto a tipos diferentes retornados, nem tampouco expressa erros de compilação relacionados a isso (BARRY, 2017).

Outra característica marcante do Python é seu sistema de indentação, fundamental para o funcionamento do código. Em diversas linguagens, blocos de código (como os condicionais if/else, funções e loops de repetição) são definidos com o emprego de colchetes “{ }”. Em Python, a indentação é responsável inteiramente por separar blocos de execução, junto com os dois pontos “:” que delimitam o início do bloco. O uso de ponto e vírgula “;” após declarações também é opcional. Embora esse sistema seja interessante, ele pode ser um tanto visualmente confuso e propício a erros difíceis de se identificar.

Os principais tipos de estruturas de dados da linguagem são:

- *List* (Lista)
    - Listas são coleções mutáveis ordenadas de objetos. Seu tamanho pode crescer ou diminuir com o acréscimo ou remoção de objetos, o que confere sua mutabilidade. Também são heterogêneas, o que significa tanto a lista não precisa ter um tipo previamente declarado quanto suporta objetos de tipos diferentes em uma mesma lista.
- *Tuple*
    - Um *tuple* é uma coleção ordenada imutável de objetos. Ao contrário de uma *list*, não é possível adicionar ou remover membros de uma *tuple* uma vez que tenha sido criada.
- *Dictionary*
    - O conceito de *dictionary* em Python se aproxima ao de *maps* em outras linguagens. Consiste de um conjunto de pares chave-valor não ordenados. Cada chave única tem um valor associado, e são suportados infinitos pares chave-valor. Embora permita valores

duplicados, não se é possível ter dois valores diferentes para uma mesma chave. Ou seja, cada chave é única. Sobrescrever uma chave já existente substituirá seu valor anterior. Uma das grandes vantagens de pares chave-valor como *dictionaries* é que permitem uma organização lógica do acesso às informações (dependem de uma chave única que relacione à propriedade de interesse, e não somente da ordem de inserção), além de serem consideravelmente rápidos para acessarem os valores.

- *Set*
  - Um *set* é uma lista de objetos não ordenados únicos. É semelhante à lista, mas proíbe objetos duplicados. Por exemplo, um *set* com os valores (3, 3, 1) é inválido e resultaria apenas em (3, 1) enquanto uma lista aceitaria diversas cópias do mesmo valor.

3.2.2 Funções

O tipo de retorno das funções frequentemente não é declarado pelos motivos mencionados na seção 3.2.1. As funções são definidas através da *keyword* (palavra-chave) *def*. Argumentos são declarados entre parênteses. A definição de argumentos com valores padrão pode ser feita através da inserção do símbolo “=” entre o nome do parâmetro e o valor padrão associado. A Figura 3 demonstra um exemplo com uma função *multiplicar*. O parâmetro *multiplicador*, por padrão, é 10, mas pode ser substituído por outro valor. O valor do parâmetro pode ser passado tanto a partir da ordem em que aparece na lista de parâmetros (linha 4) quanto através do uso de parâmetros nomeados (linha 5) sem necessidade de declaração de decoradores ou informações adicionais.

Figura 3 - Exemplo de função em Python

```
1 def multiplicar(n, multiplicador=10):
2     return n*multiplicador;
3
4 print(multiplicar(1, 5)); #5
5 print(multiplicar(10, multiplicador=2)); #20
```

Fonte: Autoria Própria (2022)

3.2.3 Módulos

Um módulo é um arquivo Python contendo definições (de funções, classes, etc) e declarações (SANTANAM, 2018). Um módulo pode ser importado por outros arquivos/programas em Python e permite uma melhor organização do código a partir da separação em seções, arquivos e pasta temáticos. Essa divisão em pastas e módulos é capaz de não só melhorar a produtividade, como permitir que diversas pessoas trabalhem simultaneamente na mesma base de código, minimizando conflitos e garantindo maior eficiência. Permite, ainda, uma melhor estruturação dos testes, uma vez que os módulos devem ser relativamente independentes. Isso implica que erros de lógica ou programação podem ser encontrados e corrigidos de forma mais rápida e eficiente.

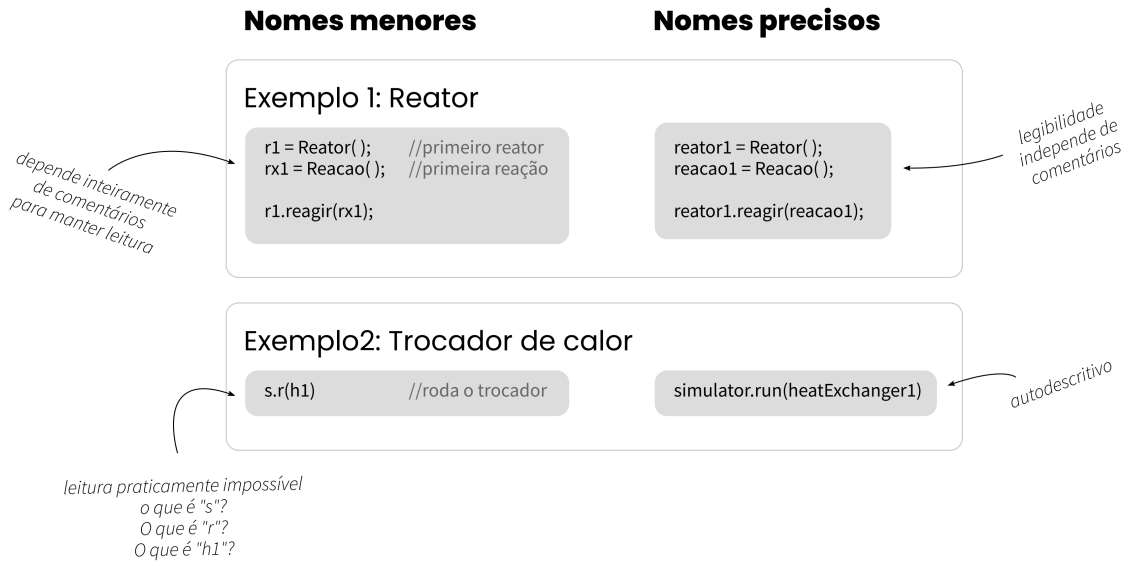
3.3 Código Limpo e Arquitetura Limpa

Embora o conceito de Código Limpo (*Clean Code*) varie conforme o autor, no geral há um consenso de que o código deve ter as seguintes características (MARTIN, 2009):

- Dificultar o aparecimento de *bugs*, e facilitar sua correção;
- Ser testável;
- Minimiza o número de dependências, classes e variáveis necessárias;
- Evita duplicação;
- Ser de fácil manutenção;
- Ser de fácil leitura. Simples e direto. As intenções de quem escreveu o código devem ser absolutamente claras para um terceiro que o ler.

A grande ênfase na legibilidade do código destaca um dos princípios mais importantes de um *Clean Code*: Ele não é dependente do autor original. Terceiros (ou mesmo o próprio autor alguns meses ou anos no futuro) devem ser capazes de ler, interpretar e promover melhores no código. Por isso, é de grande importância que variáveis, classes e funções deem preferência à clareza em oposição ao praticidade ao digitar. A Figura 4 ilustra dois exemplos contrastantes de escolha de nomes. O primeiro caso (Nomes menores) é, sem dúvidas, menor, mas é necessário consultar frequentemente os comentários (linha iniciadas por "//") para compreender o algoritmo. O segundo caso (Nomes precisos) demonstra que, com nomes autoexplicativos, alguns comentários são desnecessários, e o código se torna de fácil compreensão. Em resumos, a preferência deve ser dada a nomes curtos, desde que sejam claros no que representam. Tornar nomes de variáveis e funções menores abrindo mão de legibilidade e manutenibilidade para códigos com interesse de longo prazo é injustificável.

Figura 4 - Nomeação de Variáveis em Clean Code

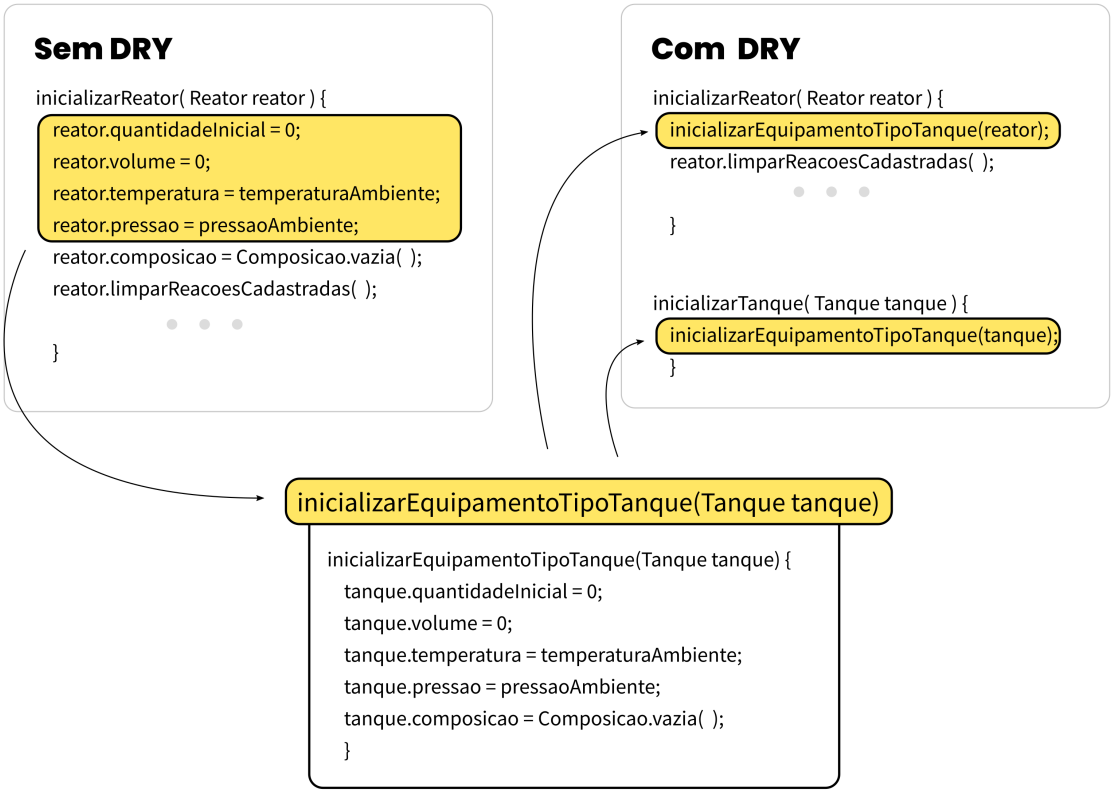


Fonte: Autoria Própria (2022)

Dentre outros princípios, destaca-se a especialização. Funções, por exemplo, devem preferencialmente ser curtas, ter objetivo claro e o executarem bem. O DRY (Não se repita, do inglês *Don't Repeat Yourself*) é um das mais conhecidas siglas da Arquitetura Limpa, e indica o combate à duplicação de código. Da mesma forma que

múltiplas “ids” se referindo ao mesmo objeto causam problemas conforme indicado na seção 3.1.1.1 (Classificação e Identificadores Únicos), a repetição de itens ou funções com o mesmo fim pode se provar bastante problemático, justificando a importância do DRY. Caso três funções, *inicializarReator*, *inicializarTrocadorDeCalor* e *inicializarTanque* façam o mesmo procedimento no início do método, o ideal é separar esse procedimento independente em uma função *inicializarEquipamento*, que por ver é chamada internamente a cada variação específica. Essa não repetição de lógica implica que, caso seja determinado alguma falha, será necessário corrigi-la em um único ponto. A figura 5 - - Aplicação do DRY em um programa deEngenharia Química demonstra um exemplo de aplicação do conceito. Na imagem, o reator e o tanque possuem características em comum (volume, temperatura, pressão...), o que permite que parte da lógica de inicialização seja compartilhada. Assim, facilita a testagem de código (pois pontos de falha são expostos de forma mais explícita) e o torna mais legível, mesmo para pessoas de fora da área. Isso possibilita, caso seja necessário, por exemplo, um Engenheiro Químico recorrer a um Engenheiro da Computação para organização do código, mesmo que este não possua conhecimento diretamente ligado à aplicação.

Figura 5 - Aplicação do DRY em um programa deEngenharia Química



Fonte: Autoria Própria (2022)

3.3.1.1 Desenvolvimento Orientado a Objetos

Uma classe é uma forma de declarar um tipo de objeto, capaz de juntar comportamentos (funções) e atributos ou estados (variáveis). Classes são úteis para representar valores ou comportamentos intimamente relacionados. Por exemplo, um reator pode ser seus valores de volume, temperatura e composição armazenados separadamente, mas também pode ser criado um objeto reator que tem as variáveis *volume*, *temperatura* e *composição* internamente. O segundo caso é mais legível, menos propenso a erros de digitação e mais fácil de testar.

Os mais importantes princípios da OOP (Desenvolvimento Orientado a Objetos, do inglês *Object-Oriented Programming*) são abstração, herança, encapsulamento e polimorfismo (KRISHNA, 2022; MONNUS, 2022). Os conceitos são brevemente definidos a seguir.

- Abstração
  - Abstrações almejam permitir a construção de códigos complexos sem que se conheça detalhes de implementação de outras classes, mas apenas seus retornos e resultados esperados;
- Herança
  - Quando classes compartilham estrutura e/ou lógica entre si, elas podem herdar uma da outra ou de uma terceira. A herança significa que a classe *filha* irá *herdar* os comportamentos e variáveis da classe *pai*. Na prática, isso permite expandir o comportamento de uma classe que já funciona bem, reutilizando o código mas sem necessidade de adicionar complexidade aos objetos já criados. Um exemplo clássico de herança se dá entre um reator e um tanque, pois todo reator compartilha características (como a existência de uma variável volume) com um tanque. Assim, a classe *Reator* pode herdar a classe *Tanque* e apenas implementar as diferenças necessárias.
- Encapsulamento

    - Consiste em armazenar as variáveis (encapsular) de forma a não poderem ser acessadas diretamente da classe. Dentre os intuitos, estão proibir mudanças ilegais em valores. Por exemplo, se ao invés de uma variável volume poder ser modificada livremente ela precisar passar por um método *setVolume*, o método pode fazer uma verificação interna e proibir valores considerados ilegais (como aqueles abaixo de 0, uma vez que não existem volumes negativos).
- Polimorfismo
    - O polimorfismo diz respeito a uma *classe filha* ou *subclasse* poder sobrescrever ou adaptar métodos da *superclasse* ou *classe pai*. Em resumo, permite a uma subclasse reimplementar comportamentos da classe da qual herdou, quando necessário.

A Figura 6 demonstra a declaração de diferentes propriedades de dois reatores através de variáveis individuais. Embora o código seja pequeno, já se faz visível determinada dificuldade de compreensão. Embora o emprego de numerações após as variáveis não as torne impossível de entender, nesse contexto, é muito difícil, senão impossível, determinar, em momentos posteriores, porque os dados do segundo reator são sufixados com numeração e os do primeiro não. Com a criação de um objeto reator, como na Figura 7, é possível tornar tanto a declaração de parâmetros quanto o acesso a estes no futuro mais uniforme e previsível. Por exemplo, todo reator é garantido de possuir uma variável chamada *composicao,* o que reduz a possibilidade de *bugs* posteriormente ao tentar acessar variáveis inexistentes.

Figura 6 - Declaração de propriedades de dois reatores como variáveis

```
#Dados do reator 1
diametro = 10;
altura = 5;
volume = calcularVolume(altura, diametro);
composicao = {};
#Para cada novo reator, será necessário criar uma
#variação das palavras

#Dados do reator2
diametro2 = 8;
altura2 = 6;
volume2 = calcularVolume(altura, diametro);
composicao2 = {};
```

Fonte: Autoria Própria (2022)

Figura 7 - Declaração de objeto reator e propriedades como exemplo

```
def Reator:
    __init__(self, altura, diametro, composicao):
        self.diametro=diametro;
        self.altura=altura;
        self.composicao;
        self.area = calcularArea(altura, diametro);

#Reator1
reator1 = Reator(altura=10, dimaetro=5, composicao={});
#Reator2
reator2 = Reator(altura=6, diametro=8, composicao={});
```

Fonte: Autoria Própria (2022)

3.4 Machine Learning

ML (Aprendizado de Máquina, do inglês Machine Learning) é um termo geral para denominar a área da ciência que cria algoritmos capazes de executar funções para as quais os algoritmos não foram explicitamente programados. Em outras, palavras, esses algoritmos têm a capacidade de aprender de forma implícita, e daí se vem o conceito de Aprendizagem pela Máquina (computador) (GÉRON, 2017).

Algoritmos de ML são versáteis, adaptáveis e muitas vezes capazes de representar ou interpretar situações que seriam demasiadamente complexas se feitas de outra forma.

Uma grande desvantagem dos métodos de ML é a necessidade de uma grande quantidade de dados. Muitas vezes, não é possível obter uma quantidade tão grande a ponto de ajustar os modelos propostos em condições normais de laboratório, sobretudo na área de engenharia química. Em um estudo com reatores batelada, por exemplo, alguns estudos coletam de 7 a 14 pontos experimentais (ALTIOK; TOKATLI; HARSA, 2006; GUILHERME, [s.d.]), enquanto que muitas vezes são necessário dados ao menos uma ordem de grandeza acima (de 100 a 1000) para poder produzir e testar um modelo.

Ao contrário de modelos matemáticos, que são propostos muitas vezes usando argumentação lógica, as expressões matemáticas derivadas desse raciocínio e, quando necessário, ajustes empíricos, modelos de machine learning são muito mais dispersos, no sentido de que podem representar quaisquer fenômenos, inclusive relações não existentes. É, portanto, necessário ressaltar que tais modelos podem ser superajustados com muita facilidade (isto é, o modelo representa perfeitamente o caso de teste, mas não é capaz de generalizar para outros contextos similares porém não idênticos). Assim, um desafio extra se dá na necessidade de ter uma grande quantidade de dados não só para produzir o modelo (etapa chamada de *train*) mas também para testá-lo (etapa chamada de *test*). Essa separação é crucial para evitar que o modelo *superajuste*, sendo capaz de reproduzir com perfeição os dados de teste, mas incapaz de ser usado para os casos para o qual está sendo desenvolvido.

Uma das técnicas que possibilita usar o mesmo conjunto de dados para teste e treino, sem necessidade de realizar novos experimentos, é a *Cross-Validation* (Validação Cruzada). A validação cruzada consiste em separar, dentro de um mesmo conjunto de dados, uma parte para ser usada na etapa de *train* da ML, e outra para ser usada na etapa de *test* do modelo treinado. Esses testes e treinos são feitos de modo intercalado, até que se atinja um limite determinado (erro mínimo, número de iterações [epochs] e conceitos parecidos). Normalmente, a quantidade destinada ao teste é complementar à do treino, no sentido em que ambas somam em 100%. Por exemplo, um conjunto de dados com 1000 pontos e com 400 pontos de teste teria 600 pontos de treino, totalizando o valor original. Essa técnica permite, ainda, testar

como o modelo se comporta de acordo com a quantidade de dados (proporções 20:80 e 50:50 entre percentual de treino e percentual de teste, por exemplo) e produzir modelos otimizados. Outra vantagem é que a comparação entre o erro médio ou desvio das predições entre a seção destinada a treino e aquela destinada a testes permite-se ter uma noção se o sistema está superajustado aos testes ou ao treino. Idealmente, o valor de ambos deve ser próximo, pois indica que o sistema tanto foi capaz de reproduzir os dados de treino adequadamente, quanto foi capaz de extrapolar e predizer dados com parâmetros de entrada distintos.

Uma das grandes dificuldades ao se trabalhar com ML é a necessidade, além da uma abundância de dados, do balanceamento dos mesmos. Sistemas que contenham muitos dados de determinado tipo podem favorecer predições com baixo erro mas que, ainda assim, serão modelos inadequados para as situações em que se pretende empregá-los. Por exemplo, consideremos um modelo que tenta predizer a presença de bactérias em um meio com base nos gases emitidos ao longo do tempo. Caso sejam alimentados 990 dados para a bactéria tipo A e 10 dados para a bactéria tipo B, o modelo poderá se limitar a prever que em 0% dos casos a bactéria B está presente e ainda assim acertar em 99% das vezes. Para esse tipo de dado, é comum o uso de um *balanceamento*, que pode ocorrer pela duplicação dos dados faltantes (replicar os dados da bactéria B 90 vezes, de modo que o sistema passe a ter medidas balanceadas dos casos A e B) ou remoção dos dados em excesso de A (reduzindo-os de 900 para 20, por exemplo) (GÉRON, 2017).

Outro ponto sensível de ML é na surpreendente grande dependência do fator humano: a filtragem dos dados que serão alimentados no sistema, bem como a escolha de algoritmos adequados e de interpretação adequada dos resultados é crucial para propor modelos que realmente representem o fenômeno estudado, fazendo jus ao esforço empregado. Desta forma, é necessário que a seleção dos dados seja feita não somente por profissionais da área de engenharia de dados ou de aprendizado de máquina, mas também por profissionais da área de coleta e/ou de aplicação desses dados.

Um ponto de destaque é que vários modelos de ML, em especial Redes Neurais (NN, do inglês *Neural Network*) podem demorar bastante tempo para serem otimizados. Contudo, uma vez que o modelo esteja construído, ele pode ser executado em frações de segundo (GÉRON, 2017; SANTANA et al., 2022). A implicação dessa característica é que, normalmente, são necessários computadores

mais robustos, bem como um esforço computacional e tempo de *treino* maior em comparação com métodos tradicionais de classificação ou regressão. Contudo, os resultados são *data-driven*, capazes de representar fenômenos sem equacionamento matemático próprio e, mais importante, os modelos gerados são relativamente leves. Em outras palavras, a maior exigência computacional na fase de testes e criação do modelo é compensada pois uma vez que o modelo tenha sido construído e validado, pode ser usado indefinidamente por um custo computacional muito reduzido.

Os algoritmos que empregam ML podem se dividir em basicamente duas grandes categorias: classificação e regressão. Os algoritmos de classificação são responsáveis por predizer categorias ou classificações de modelos com base em dados de entrada. Um exemplo clássico são programas capazes de analisar fotografias ou vídeos e determinar as pessoas existentes nessas imagens com base no rosto. Nesse sentido, um algoritmo supervisionado seria aquele que já foi treinado com as pessoas que deve reconhecer, e classifica os rostos detectados como uma dessas pessoas. No mesmo exemplo, um algoritmo não supervisionado seria aquele que detecta todos os rostos de pessoas e os agrupa para determinar as pessoas existentes em um conjunto de fotografias, mas não as rotula. Ou seja, a ideia de supervisão é que as classes ou categorias são informadas previamente, e o algoritmo deve determinar a categoria a cada objeto de interesse analisado pertence. No não supervisionado, o algoritmo deve produzir as possíveis categorias ao mesmo tempo em que já associa cada objeto analisado a uma delas. Um algoritmo de regressão é aquele que produz resultados numéricos. Dentre os principais algoritmos de regressão supervisionados, destacam-se (GÉRON, 2017):

- k-Nearest Neighbours (Vizinhos Próximos);
- Regressão Linear;
- Regressão Logística;
- Support Vector Machines (SVMs);
- Árvores de decisão (DF, do inglês Decision Trees);
- Floresta Randômica (RF, do inglês Random Forest);
- Redes Neurais.

Dado que em Engenharia Química comumente se trabalha com dados numéricos, algoritmos de regressão tem um maior interesse imediato. Existem

estudos aplicando alguns dos modelos citados à predição de variáveis de efluentes (MATEO PÉREZ et al., 2021), à produção de bio-hidrogênio (PANDEY et al., 2023) e cream cheese (LI et al., 2021), ao tratamento de efluentes (BAGHERZADEH et al., 2021), à digestão anaeróbia (ANDRADE CRUZ et al., 2022), dentre outros. Todos os modelos aqui apresentados requerem uma quantidade razoável de dados disponíveis para boa performance sem superajuste. Como muitas vezes essa abundância de dados, ou por ser muito custosa, ou por ser impraticável, ou pelos estudos que geraram dados terem sido realizados para uso interno de uma empresa ou instituição, não existem na literatura científica. Por isso, outras estratégias foram buscadas, como os PINNs.

3.4.1 Neural Networks

ANN (Redes neurais artificiais, do inglês Artificial Neural Networks) são algoritmos altamente adaptáveis (ANDRADE CRUZ et al., 2022), capazes de simular o funcionamento do cérebro humano. Possuem a capacidade de funcionar como aproximadores universais de qualquer função com entradas e saídas (SANTANA et al., 2022). Foram citados pela primeira vez em 1943, com um modelo matemática que tentava reproduzir o comportamento de neurônios tendo como base o cérebro de animais (GÉRON, 2017). O conceito rapidamente criou *momentum* e se difundiu, atraindo a atenção de diversas áreas do conhecimento. Inicialmente, criou-se uma expectativa muito grande a respeito de redes neurais, o que fez com que o interesse nas mesmas disparasse, mas como não foram capazes de atender às altíssimas expectativas de computadores super inteligentes, foram perdendo espaço lentamente até a década de 80, quando ocorreu um novo *boom*. Nesse período, novas técnicas surgiram e expandiram consideravelmente o potencial das ANN. Outro fator relevante para o grande interesse em ANN a partir da década de 90 foi o grande aumento da capacidade de processamento computacional disponível em todos os computadores, em especial àqueles aos quais a maioria dos cientistas tinha acesso: os comuns. A indústria dos jogos também foi uma grande responsável pelos avanços dessa época, uma vez que estimulou a criação de GPUs (Unidades de Processamento Gráfico) mais potentes.

A menor unidade de uma rede neural é neurônio. Analogamente a uma rede neural biológicas, as ANN possuem três camadas: o input (entrada), a *hidden layer* (camada secreta) e a *output* (saída). Cada conexão entre neurônios de diferentes camadas possui um *weight* (peso) responsável por balancear a importância entre os diferentes neurônios e camadas para a redução do erro das predições (ALZUBI; NAYYAR; KUMAR, 2018). Um neurônio artificial é tão somente uma função que ativa determinados *outputs* em função dos *inputs* recebidos. Um perceptron é ligeiramente diferente de um neurônio artificial, pois passa a aceitar *inputs* e *outputs* numéricos, bem como associar cada conexão de entrada, em cada neurônio, a um *weight*, responsável por contabilizar a influência que aquela conexão tem naquele neurônio. Perceptrons apresentam algumas desvantagens, como a incapacidade de resolver alguns problemas simples e serem incapazes de produzir uma regressão logística. Contudo, muitas dessas dificuldades podem ser contornadas empregando uma série de perceptrons em sequência, em camadas. Esse conjunto de camadas é denominado MLA(do inglês *Multi-Layer Perceptron*) e é um tipo de ANN.

Os principais hiperparâmetros (parâmetros relacionados à própria estrutura ou taxa de aprendizado) da ANN são:

- Deepness (profundidade) da *hidden layer* (número de camadas);
- Wideness (largura), a quantidade de neurônios em cada camada dentro da camada oculta;
- A taxa de aprendizado (normalmente igual a $1x10^{-3}$), que é o valor usado para influenciar o quanto a rede modifica o conjunto de pesos a cada interação com base no erro.

3.4.2 Aplicação em bioprocessos

Bioprocessos naturais são, em geral, extraordinariamente complexos e, sob alguns aspectos, impossíveis de modelar e prever. Quando termo “bioprocessos” se refere a processos produzidos pelo homem, contudo, a descrição é um pouco diferente. Nesse contexto, bioprocessos são processos que empregam seres vivos de qualquer natureza ou produtos produzidos por eles (células, enzimas, vírus, etc) para a produção, modificação ou tratamento de determinada substância ou conjunto

de substância e que foi realizado propositalmente por ação humana. Dentre os bioprocessos, destacam-se os processos fermentativos, presentes na produção de diversos bens de consumo (como iogurtes, cervejas e queijos) e tratamento de efluentes (LIM et al., 2023). A representação de bioprocessos através de modelos matemáticos é um desafio que vem sendo superado já há algum tempo (DORAN, 2013), mas ainda existem modelos muito complexos ou microrganismos cuja cinética de crescimento, consumo de substrato ou geração de produto não é representada de forma simples pelas equações comumente empregadas e suas variantes. Os modelos matemáticos que apresentam boa performance na representação desses processos podem ser empregados posteriormente na simulação do processo, com o intuito de reduzir custos, melhorar o impacto ambiental ou reduzir o tempo de produção. Uma alternativa ao uso de modelos matemáticos é o emprego de ANN para a predição das variáveis de saída de interesse (como concentração de células ou de produto) com relação às variáveis de entrada (como o tempo decorrido desde o início do experimento). Contudo, essa alternativa é bastante custosa por causa das exigências em relação à quantidade e qualidade dos dados citada anteriormente.

Diversos modelos de ML já foram aplicados nos mais variados bioprocessos, mas muitos não conseguiram predizer adequadamente as variáveis de interesse (LIM et al., 2023). Por conta disso, os métodos de simulação numérico computacionais continuam sendo considerados uma ferramenta bastante robusta e até mesmo a mais adequada para diversos casos, uma vez que não dependem da disponibilidade de *big data*. Contudo, métodos numéricos podem depender significativamente das estratégias de discretização empregadas, e é praticamente impossível dissociar o método de solução numérico do método de discretização. Assim, é difícil comparar diferentes métodos numéricos sem levar em consideração a estratégia de discretização adotada. Outro ponto relevante é que o uso de malhas muito finas (aumentar o número de pontos de discretização) nem sempre resolvem o problema encontrado em malhas menos finas, bem como podem acarretar em grandes tempos de execução do programa – reflexo do grande esforço computacional necessário. É nesse cenário que surgem as *Physics-Informed Neural Networks*, um híbrido entre redes neurais e solução de modelos matemáticos.

3.5 Physics-Informed Neural Network

PINN (Physics-Informed Neural Network) nasce num contexto onde diversas técnicas de ML não conseguem convergir a resultados apropriados devido à falta de *big data*. Introduzido à comunidade científica em 2018, o framework aproveita a otimização de derivadas graças ao uso de GPUs por ANN para aplicá-las à derivação de equações e expressões matemáticas (RAISSI; PERDIKARIS; KARNIADAKIS, 2019). A funcionalidade de aproximadores universais, capazes de representar qualquer função com pontos de entrada e de saída de ANN é explorada para gerar as variáveis de saída desejadas e suas respectivas derivadas.

PINNs adotam uma abordagem diferente das ANN. A geração de dados se dá através do uso de modelos matemáticos. Uma função loss (que representa o erro) é minimiza em cada etapa de iteração (chamada de *epoch* em vários algoritmos). A função consiste na soma dos desvios entre os valores preditos pela rede neural e aqueles que seriam os corretos (obtidos através do equacionamento matemático). É através de iterações subsequentes que a rede é capaz de aprender os valores do sistema de equações e resolvê-lo. A partir da minimização do erro das derivadas, portanto, o sistema é capaz de obter também as variáveis. Na prática, é equivalente a um sistema de integração numérico, mas possui duas grandes vantagens. A primeira é não depender fortemente da discretização dos pontos no tempo ou no espaço. A segunda vantagem é que, uma vez que o modelo tenha sido treinado e otimizado (o que pode levar um tempo considerável), ele pode ser reutilizado infinitas vezes a um custo computacional ínfimo. Isso poderia permitir, por exemplo, a otimização de modelos em computadores de alta capacidade, mas a execução dos modelos já prontos em computadores comuns. A função de perda (loss) é definida abaixo.

$$L = \sum w_N \left\| \left(\left. \frac{dN}{dt} \right|_{calculado} - \left. \frac{dN}{dt} \right|_{previsto} \right) \right\|^2 \tag{1}$$

onde L é a *total loss* e w_N é o *weight* da variável N em relação à perda. Reparar que o *weight* aqui se refere ao peso que a variável N compõe para a determinação da *loss*, e não ao peso empregado entre neurônios, que é um outro conceito.

Diversos fenômenos já foram representados através de PINNs, como fluxos aerodinâmicos (MAO; JAGTAP; KARNIADAKIS, 2020) e processos de adsorção (SANTANA et al., 2022). Embora a aplicação de modelos de ML a biorreações e biorreatores, bem como estações de tratamento de efluentes (ANDRADE CRUZ et al., 2022; BAGHERZADEH et al., 2021; MATEO PÉREZ et al., 2021; MEY et al., 2021) seja bastante estudada, a aplicação de PINNs a bioprocessos ainda não é tão difundida. Assim, novos esforços, como este, são bem-vindos para identificar os principais méritos e dificuldades do método na representação de bioprocessos.

3.6 DeepXDE

DeepXDE é uma biblioteca em linguagem de programação Python e que emprega outras bibliotecas de Machine Learning e Inteligência Artificial (como Tensorflow e Pytorch) para a solução de sistemas de equações diferenciais através de PINNs. As derivadas das variáveis de interesse são avaliadas através de *back-propagation*, uma técnica onde o valor das derivadas é avaliado em uma etapa reversa, logo após a avaliação do valor das variáveis em si, em uma etapa normal (*forward*). A DeepXDE é usada ao longo de todo este trabalho para a construção e avaliação dos modelos PINN descritos. A construção da DeepXDE sistema é detalhada no artigo original (LU et al., 2021).

3.7 Engenharia de Processos

A Engenharia de Processos surge da necessidade da Engenharia Química de melhor integrar, otimizar e modelar processos. É uma área interdisciplinar por natureza, abordando conceitos de Ciências básicas (como Biologia e Química), Fundamentos (compreensão de fenômenos e modelagem matemática) e Engenharia de Equipamentos (projeto e modelagem de equipamentos como reatores, trocadores

de calor e colunas de adsorção) (PERLINGEIRO, 2018). É a Engenharia de Processos que permite o projeto de processos integrados, conectando blocos de processos e equipamentos que outrora seriam modelados e avaliados separadamente. Da mesma forma que representa um grande potencial por possibilitar a geração sistemática de sistemas baseados em variáveis de interesse, a área também apresenta muitos desafios, pois se faz necessário conhecimento em áreas distintas, integração dos mesmos, boa compreensão de conceitos e fundamentos básicos e boa organização das ideias e dos algoritmos empregados.

3.7.1 Processos estacionários e transientes

Processos de qualquer natureza podem ser estacionários ou transientes. São considerados estacionários aqueles processos ou equipamentos cujas propriedades (como concentração, pressão, temperatura ou volume) não variam ao longo do tempo. Já o estado transiente representa aqueles processos onde há variação de qualquer propriedade ao longo do tempo (DORAN, 2013). São exemplos de processos estacionários um frasco com água pura e perfeitamente isolado do exterior, não sofrendo ou realizando qualquer tipo de interação, e um tanque em estado contínuo, onde a matéria e energia que entram e que saem do volume de controle são constantes, mantendo portando as propriedades internas também constantes. São exemplos de processos ou sistemas transientes reatores em batelada e os mais diversos fenômenos da natureza, como a variação da água em um lago devido à evaporação, chuvas e à corrente de um rio próximo.

3.7.2 Dimensionamento e Simulação de Equipamentos

O dimensionamento de um equipamento consiste na determinação das propriedades e variáveis que, em conjunto, permitirão a construção e/ou simulação dos equipamentos. Para tanto, se fazem necessários 2 subconjuntos de variáveis:

as condições conhecidas (como temperatura na superfície externa do reator, se o equipamento é isolado ou não do meio em que está inserido) e as metas de projeto, que determina quais variáveis devem ser priorizadas durante a proposta da solução (PERLINGEIRO, 2018). No projeto de um reator contínuo, a minimização do volume a fim de reduzir gastos é uma meta de projeto, e a produção fixada de produto em kg/h é uma condição conhecida. A simulação consiste em empregar as variáveis vindas das correntes de entrada (sejam elas de matéria ou energia) e as dimensões do equipamento para simular seu comportamento.

As operações de dimensionamento e simulação, aplicadas à otimização, podem ser empregadas para melhorar o design, segurança e lucratividade de processos químicos, mas requerem uma boa bagagem teórica dos fundamentos e fenômenos envolvidos no processo de interesse. A simulação pode, ainda, ser empregada como uma ferramenta auxiliar para diagnosticar problemas em uma indústria já existente e propor soluções (JANA, 2011).

3.8 Ácido Lático

O Ácido Lático (LA) é uma molécula de grande importância econômica. Empregado nas indústrias alimentícia, farmacêutica, cosmética e de síntese. É empregado ainda em impressão 3D, na fabricação do polímero PLA (Poli-Ácido Lático), polímero estudado há anos (DATTA et al., 1995; LEE et al., 1998) e que pode substituir o PET (Poli etileno) em algumas aplicações. Foi descoberto em 1780 pelo químico Scheele. Em 1857, Pasteur determinou que não era uma substância presente naturalmente no leite, mas fruto do metabolismo de microrganismos. Até os dias atuais a rota de produção fermentativa é amplamente estudada. Isso se dá pelo LA possuir dois isômeros: L(+) e D(-) Ácido Lático (KOMESU; MACIEL; FILHO, 2017). A síntese química gera uma mistura racêmica, e a fermentativa pode favorecer expressivamente um dos isômeros a depender das condições e do microrganismo empregado. O emprego de açúcares refinados como fonte de carbono é relativamente caro, então outras alternativas estão sendo estudadas, bem como fontes de nitrogênio menos financeiramente dispendiosas (ALTAF; NAVEENA; REDDY, 2007).

Estima-se que a produção mundial de LA alcançará 1960 mil toneladas em 2025 (LÓPEZ-GÓMEZ et al., 2019), com a indústria alcançando um valor de 8,7 bilhões de dólares americanos (DIN et al., 2021) com cerca de 90% sendo obtido por via fermentativa. A produção de LA costumeiramente ocorre em modo batelada, embora existam modelos contínuos e batelada-alimentada em operação. Alguns processos empregam ainda um reciclo de células para maximizar a conversão do substrato. O processo em batelada costuma ter maior percentual de conversão do substrato (ou seja, o substrato é melhor aproveitado) mas apresenta menor produtividade. Assim, como muitos substratos apresentam valor econômico considerável, a escolha do processo em batelada em detrimento do contínuo, muitas vezes, se dá pela otimização dos custos através do maior aproveitamento possível da matéria-prima (KOMESU; MACIEL; FILHO, 2017). Fica claro, portanto, que o custo da matéria-prima é um ponto importante na determinação do preço de venda e da margem de lucro no mercado de LA, e que são necessários esforços para a descoberta de substratos de custos reduzidos, com menor impacto ambiental e com melhor apelo de marketing.

Muitas dos esforços para redução de custos na indústria do LA envolvem a busca por novos microrganismos, engenharia genética e novas fontes de matérias-primas para serem usadas como substratos (como rejeitos ou subprodutos industriais). O Quadro 1- Substratos empregados na produção de LA exibe algumas matérias-primas empregadas na produção de LA em diversos estudos. Na Linha Método, o termo “B” indica o emprego de organismos microbiológicos, e o termo “Q” o emprego de síntese de natureza química.

Quadro 1 - Substratos empregados na produção de LA

Substrato	**Método de Produção**	**Microrganismo ou tecnologia química empregado**	**Referência**
Cassava Flour (Farinha de Mandioca)	B	*Lactobacillus brevis*	(QUINTERO et al., 2012)
Glycerol (glicerina)	Q	Conversão hidrotérmica	(ARCANJO; FERNANDES; SILVA, 2015)
Potato Starch (Amido de Batata)	B	*Thermotoga neapolitana*	(PRADHAN et al., 2021)

Whey (Soro de leite)	B	*Lactobacillus casei*	(ALTIOK; TOKATLI; HARSA, 2006)
Xarope de açúcares (subproduto da produção de cenouras)	B	*Rhizopus oryzae* e *Rhizopus arrhizus*	(SALVAÑAL et al., 2021)
Sugar cane juice (Suco da cana-de-açúcar)	B	*Lactobacillus delbrueckii*	(DEY; PAL, 2013)
Glicose	B	*Rhizopus oryzae*	(HAMAMCI; RYU, 1994)
Amido	B	*Lactobacillus amylophilus*	(ALTAF; NAVEENA; REDDY, 2007)

Fonte: Autoria Própria (2022)

Os substratos tradicionais, como açúcares refinados, competem diretamente com a indústria alimentícia e são denominados como 1G (primeira geração). Os substratos advindos de rejeitos ou subprodutos de outras indústrias, como o soro de leite (ALTIOK; TOKATLI; HARSA, 2006) ou um xarope derivado de cenouras descartadas (SALVAÑAL et al., 2021) oferecem custos reduzidos e maior apelo ambiental, e são determinados 2G (segunda geração). Embora em um primeiro momento a produção a partir de matéria-prima 2G pareça necessariamente mais vantajosa, muitas vezes não é o que é observado. Como essas matérias-primas são mais complexas e quase sempre não são refinadas, é necessário lidar com as variações de safra ocasionadas por diversos fatores, muitos fora do controle do homem – como o clima no local da produção. Além disso, incluem contaminantes como vanilina e furfural, que precisam ser separadas posteriormente e podem elevar o custo do processo de purificação (DIN et al., 2021).

A maioria dos microrganismos empregados na produção de Ácido Lático são bactérias, embora também existam processos que empreguem fungos. As LAB (Bactérias Produtoras de Ácido Lático, do inglês *Lactic Acid Bacteria*) são em sua grande maioria *cocci*, e os pontos ótimos de operação de reatores que as empregam variam de 25 a 45ºC (temperatura) e 5 a 7 (pH) (KOMESU; MACIEL; FILHO, 2017). Em geral, as LAB também são microrganismos anaeróbios facultativos e toleram pHs relativamente ácidos. Contudo, em pHs muito ácidos, a produção de LA, que é o grande objetivo do processo, é fortemente prejudicada, então algumas estratégias devem ser adotadas para manter o pH numa faixa apropriada. A adição de bases ou álcalis e/ou o emprego de soluções tampão no meio são descritas na literatura como

procedimentos apropriados pela literatura. Esses procedimentos são essenciais porque, tendo em vista que o LA é um ácido orgânico, sua a conversão de substrato em produto naturalmente reduz o pH do meio, o que por sua vez pode reduzir a produtividade.

3.8.1 Purificação

O processo de purificação tradicional do Ácido Lático se dá por precipitação química, mas consome quantidades apreciáveis de reagentes (Ácido Sulfúrico e Hidróxido de Cálcio) e gera grande quantidade de rejeitos (estima-se que até 1 tonelada de gesso seja gerada para cada 1 tonelada de LA produzida (DATTA et al., 1995)). Assim, houve um estímulo à busca por processos mais sustentáveis, que empreguem matérias-primas menos agressivas ao meio ambiente e gerem menos efluentes ou descartes. Para muitos autores, o emprego de colunas de troca iônica é uma das alternativas mais eficientes. Além de ser um processo relativamente simples e com boa fundamentação na literatura científica, reduz consideravelmente os problemáticos rejeitos e é possível de ser implementado em escala industrial. O processo também pode reduzir o tempo de purificação e os custos com mão de obra. Como a tecnologia de troca iônica possui muitas especificidades, a resina e grupo ligante devem ser selecionados levando-se em conta o meio reacional, pH, temperatura e eluente a ser empregado (DIN et al., 2021).

3.9 Modelagem matemática

3.9.1 Tanque

A modelagem de um tanque é um simples balanço de massa, que emprega que a variação interna da propriedade *m* é igual à entrada menos a saída de tal propriedade, conforme indicado nas equações a seguir:

$$[acúmulo]=[entrada]-[saída]+[geração] \quad (2)$$

Caso o termo de geração/consumo seja considerado nulo (implicando que no tanque não ocorrem reações químicas, interações ou variações de temperatura, pressão e volume significativas) a equação pode ser expressa como:

$$\frac{dm}{dt}=\frac{dm_{\text{in}}}{dt}-\frac{dm_{out}}{dt} \quad (3)$$

Onde m é a propriedade, m_{in} é a propriedade na corrente de entrada do tanque e m_{out} é a propriedade na corrente de saída do tanque. A equação pode ser expandida, com *m* se aplicando a cada propriedade de cada componente das correntes de entrada e saída.

3.9.2 Reatores CSTR e Batelada

A modelagem de um volume de controle qualquer, que servirá de base para a representação de diferentes reatores, é relativamente parecida com a apresentada no tópico 3.9.1 Tanque, sendo a principal diferença a existência de um termo de geração ou consumo de matéria, o que acaba por implicar também na possibilidade de termos para o consumo ou geração de energia em virtude das reações químicas que ocorrem. O balanço de massa pode ser resumido como segue:

$$[corrente\,entrada]-[corrente\,saída]+[variação\,por\,reações]=[acúmulo] \quad (4)$$

O que por sua vez pode ser representado como símbolos:

$$F_{j0}-F_j+\int^{V} dV=\frac{dN_j}{dt} \quad (5)$$

Onde F representa a vazão molar da substância “j”, N representa o número de mols, V o volume e t o tempo. O subscrito “0” em F_{j0} indica se tratar da corrente localizada na entrada do reator, ao passo que F_j representa a corrente de saída. A equação 5 é generalista, e pode ser empregada tanto para a modelagem de reatores batelada quanto contínuos.

Um CSTR (Reator perfeitamente agitado, do inglês *Countinuous stirred-tank reactor*) é um tipo de reator muito empregado no meio industrial. Dentre suas vantagens, destacam-se a simplicidade de projeto e modelagem, uma vez que as propriedades são consideradas como uniformes em todo o interior do reator (FOGLER, 2018). Enquanto reatores CSTR podem operar em estacionário ou transiente, reatores em batelada são sistemas essencialmente transientes. Mesmo que não seja observada variação de massa durante determinado período de operação, outras propriedades como a concentração e número de moles ou massa de cada substância ou microrganismo presente pode mostrar dependência temporal, além de variações de energia, temperatura, volume e pressão (DORAN, 2013).

3.9.2.1 Variáveis de Processo

Comumente, um processo químico pode ser descrito empregando três tipos variáveis: *input* (entrada), *output* (saída) e variáveis de estado (JANA, 2011). As variáveis de entrada e de saída são, frequentemente, os pontos de “comunicação” entre um equipamento ou processo e os demais. Por exemplo, as correntes de entrada e saída na Figura 5 são variáveis de *input* e *output*, respectivamente. Já uma variável de estado é aquela que é capaz de descrever o *estado* do sistema em determinado ponto do espaço e do tempo, e aparecem com frequência nos termos de acúmulo de equações de balanço. Por exemplo, a composição química é uma variável de estado capaz de descrever o balanço de massa do sistema em um determinado ponto, e é uma variável de estado.

3.9.2.2 Modelo

O modelo de um processo é uma abstração matemática capaz de descrevê-lo. O conjunto de equações usados não representa o modelo físico no qual ele é baseado de forma completa, mas apenas o suficientemente adequada para manter a fidelidade da representação (JANA, 2011). O fenômeno é mais comum em reações químicas com resistência à transferência de massa: por vezes o processo de determinação dos coeficientes e do equacionamento do fenômeno difusivo e convectivo é tão complexo que torna impeditiva uma modelagem tão detalhista. Contudo, uma modelagem que ignora esses efeitos e os inclui dentro das taxas de reação costuma ser uma aproximação boa o suficiente para representar o sistema de forma apropriada.

4 METODOLOGIA

Nesta seção são descritas as estratégias, equacionamento e variáveis aplicadas nas etapas de simulação e otimização, necessárias à reprodutibilidade e melhor compreensão do estudo.

4.1 Adimensionalização

A adimensionalização é um processo em que variáveis são convertidas para variáveis adimensionais através do uso de fatores de adimensionalização, que serão chamados de *scalers* ao longo do trabalho. Esse scalers podem reduzir ou aumentar significativamente o erro no sistema de equações que está sendo resolvido e podem, portanto, gerar informações valiosas sobre como melhor o modelo computacional através do emprego de adimensionalização no sistema de equações (ALHAMA MANTECA; SOTO MECA; ALHAMA, 2012). O conjunto de equações adimensionais ou de dimensões reduzidas produzido pode ser capaz, portanto, de necessitar de menos recursos computacionais, produzir melhores resultados ou, por conta do processo de adimensionalização, ser incapaz de convergir a uma solução. Como PINNs podem ser bastante sensíveis aos valores das variáveis, derivadas e *loss*, as equações foram inteiramente adimensionalizadas para avaliar o impacto que o procedimento teria nos resultados da simulação.

Uma variável qualquer (representada por “N”) pode ser adimensionalizada através do uso de um coeficiente:

$$N = N_A * N_S \tag{6}$$

onde N é uma variável qualquer, N_A é a variável adimensional e N_s é o scaler (fator) de adimensionalização. O uso do subscrito A denota, portanto, a variávle adimensional, ao passo que o uso do subscrito S representa o scaler de adimensionalização de N. Essa convenção de nomenclatura é mantida ao longo de todo o trabalho, salvo quando for explicitamente indicado o contrário.

4.2 Modelo Matemático da Cinética de Reação

O modelo de produção de Ácido Lático por *Lactobacillus casei* a partir de lactose do soro de leite aqui descrito foi proposto e validado em um outro trabalho (ALTIOK; TOKATLI; HARSA, 2006) usando dados experimentais de um reator em batelada. A reação se dá atráves do consumo de substrato (lactose do soro de leite) para crescimento e manutenção de biomassa (*L. casei*) e geração de produto (ácido lático). As concentrações de biomassa, produto e substrato são representadas, respectivamente, por X, P e S. Neste trabalho, o experimento número 2, com concentração inicial de lactose de 21.4 g/L e duração de 9 horas, é usado como um estudo de caso. Os parâmetros são mostrados na Tabela 1.

Tabela 1 - Parâmetros empregados na equação cinética

Parâmetro	Símbolo	Valor	Unidade
Concentração inicial de biomassa	X_o	1,15	g . L^{-1}
Concentração inicial de produto	P_o	6	g . L^{-1}
Concentração inicial de substrato	S_o	21,4	g . L^{-1}
Concentração inibitória de biomassa	X_M	8	g . L^{-1}
Concentração inibitória de produto	P_M	90	g . L^{-1}
Velocidade máxima de crescimento celular	μ_{max}	265	h^{-1}
Constante de Monod	K_S	0,72	g lactose . L^{-1}
Coeficiente de geração de produto associada ao crescimento celular	α	3,0	g ácido lático . g^{-1} biomassa
Coeficiente de geração de produto não associada ao crescimento celular	β	0,06	g ácido lático . g^{-1} biomassa h^{-1}
Coeficiente de rendimento do produto	Y_{PS}	0,682	g ácido lático . g^{-1} lactose
Coeficiente de manutenção celular	m_s	0,03	g lactose g^{-1} biomassa h^{-1}

Coeficiente exponencial de inibição por biomassa	f	0,1	adimensional
Coeficiente exponencial de inibição por produto	h	0,3	adimensional
Tempo do experimento	t_{XP}	9	h

Fonte: Adaptado de (ALTIOK; TOKATLI; HARSA, 2006)

A produção de LA por *L. casei* é notoriamente controlada por efeitos de inibição e pode ser representada pela equação clássica de Monod, desde que sejam feitos alguns ajustes. O primeiro é o ajuste que representa o efeito de inibição por produto, representado por $(1 – P/P_M)$. O segundo é o efeito de inibição por biomassa, representado por $(1 – X/X_M)$. Assim, o modelo conta com termos de inibição por biomassa, produto e substrato. A equação que representa a variação da concentração de biomassa, r_X (g . L^{-1}. h^{-1}), é descrita a seguir:

$$r_X = \frac{\mu_{max} S}{K_S + S} X \left(1 - \frac{X}{X_M}\right)^f \left(1 - \frac{P}{P_M}\right)^h \quad (7)$$

onde r_x é a taxa de reação da biomassa, μ_{max} é a velocidade máxima de crescimento, S é a concentração de substrato, K_S é a constant de Monod e f e h são coeficientes e representam os fatores de inibição por biomassa e por produto, respectivamente.

A geração de LA (r_P) pode ser representada satisfatoriamente pelo modelo cinético de Luedeking-Piret e depende linearmente do crescimento celular (r_X) e da concentração de biomassa (X) e é representado a seguir:

$$r_P = \alpha r_X + \beta X \quad (8)$$

onde r_P é a taxa de geração de produto, α é o coeficiente de geração de produto associada ao crescimento celular (relaciona-se à variação da concentração de biomassa), e β é o coeficiente de geração de produto não associado ao crescimento celular (relaciona-se apenas à concentração de biomassa).

A cinética de consumo do substrato (r_S) é dada em função da taxa de produto produzida (r_P) e da quantidade de substrato empregada na manutenção celular:

$$r_S = \frac{-1}{Y_{PS}} r_P - m_s X \tag{9}$$

onde Y_{PS} é o coeficiente de rendimento do produto, r_S é a taxa de consumo do substrato e m_S é o coeficiente de manutenção celular.

As equações 7 a 9 podem ser adimensionalizadas, como descrito na seção 4.1, como segue:

$$\frac{X_S}{t_S} r_{X_A} = \frac{\mu_{max} S_A S_S}{K_S + S_A S_S} X \left(1 - \frac{X_A X_S}{X_M}\right)^f \left(1 - \frac{P_A P_S}{P_M}\right)^h \tag{10}$$

$$\frac{P_S}{t_S} r_{P_A} = \alpha \frac{X_S}{t_S} r_{X_A} + \beta X_A X_S \tag{11}$$

$$\frac{S_S}{t_S} r_{S_A} = \frac{-1}{Y_{PS}} \frac{P_S}{t_s} r_{P_A} - m_s X_A X_S \tag{12}$$

4.3 Dados Experimentais

Os dados experimentais para a cinética de crescimento celular, consumo de substrato e geração de produto para um reator batelada com 3 litros de volume útil foram obtidos de um outro estudo (ALTIOK; TOKATLI; HARSA, 2006) e são descritos na Tabela 2. Na tabela, t representa o tempo da coleta dos dados, X a concentração de biomassa (g/L), P a concentração de ácido lático (g/L) e S a concentração de lactose (g/L).

Tabela 2 - Dados experimentais de Altiok (2006)

t (h)	X (g/L)	P (g/L)	S (g/L)
0	1,45	5,50	21,40
1	1,80	5,80	19,00
2	2,09	6,10	16,90

3	2,42	7,50	14,50
4	3,07	9,50	10,00
5	3,80	12,00	6,50
6	4,65	16,00	2,50
7	5,10	17,80	0,10
8	4,84	18,50	0,03
9	4,70	18,60	0,03

Fonte: Adaptado de (ALTIOK; TOKATLI; HARSA, 2006)

4.4 Modelo Matemático do reator

O balanço de volume de líquido no reator é dado por:

$$\frac{dV}{dt} = f_{\text{in}} - f_{\text{out}} \tag{13}$$

onde V é o volume, t é o tempo, f_{in} é a vazão volumétrica na corrente de entrada e f_{out} é a vazão volumétrica na corrente de saída.

A equação 13 pode ser adimensionalizada como:

$$\frac{V_S}{t_S}\frac{dV_A}{dt_A} = f_{\text{in}} - f_{\text{out}} \tag{14}$$

O balanço adimensional para a concentração de uma espécie qualquer (substância ou biomassa), representada por N, é representado a seguir:

$$V_A V_S \frac{N_S}{t_S}\frac{dN_A}{dt_A} = V_A V_S \frac{N_S}{t_S} r_{N_A} + f_{\text{in}} N_{\text{in}} - f_{\text{out}} N_A N_S \tag{15}$$

onde N é a concentração de cada substância ou biomassa no reator, V é o volume de líquido no interior do reator, t é o tempo, f_{in} é a vazão volumétrica na corrente de entrada do reator, N_{in} é a concentração de N na corrente de entrada do reator, f_{out} é a vazão volumétrica na corrente de saída do reator, N_{out} é a concentração de N na corrente de saída do reator. Os subscritos S e A foram definidos na seção 4.1.

O modelo matemático descrito pela equação 15 pode ser usado para representar um reator operando em modo batelada, batelada alimentada ou CSTR. Para a operação em batelada, f_{in} e f_{out} são iguais a zero. Para a operação em batelada alimentada, apenas f_{out} é zero. Para a operação CSTR, tanto f_{in} quanto f_{out} são maiores ou iguais a zero e $f_{in} = f_{out}$ no estado estacionário. No caso do CSTR, a corrente pode ser zerada quando o *start-up* do reator, com enchimento de toda a capacidade de volume, é simulada, e não tão somente o estado estacionário.

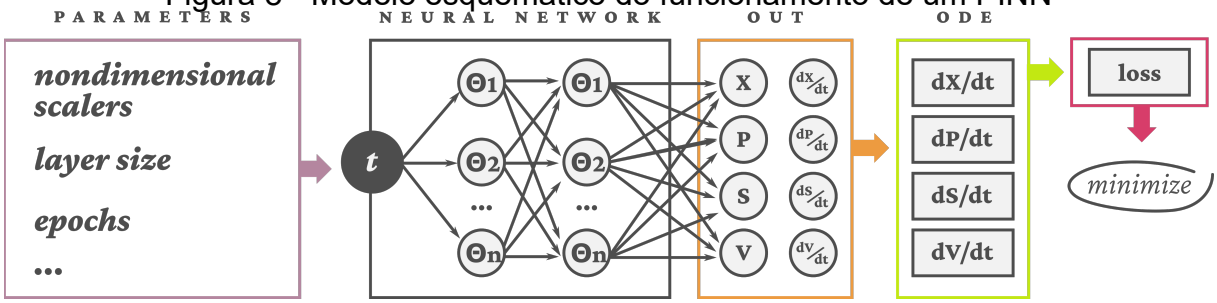
4.5 Simulação usando *Physics-Informed Neural Networks*

Um dos grandes pontos de destaque de PINNs é associar a disponibilidade de redes neurais para simulação de processos com poucos ou sem dados, através do uso de modelos matemáticos. Na prática, isso torna PINNs alternativas às técnicas tradicionais de simulação e modelagem, incluindo integração numérica através dos métodos de Euler e Runge-Kutta, por exemplo. A função loss, que é usada para minimização do erro e melhoria sequencial da rede neural, foi descrita pela equação 1, na seção 3.5. A variável genérica, N, representa X, S, P ou V. Serão os valores das derivadas em função do tempo dessas variáveis que serão empregados para avaliar o erro e consequentemente promover a convergência a uma solução adequada através da função loss.

Para testar diversas configurações para os PINNs aqui estudados, uma *grid search* e um loop de repetição foram criados. Um *grid search* é uma forma de testar diversas combinações entre hiperparâmetros de interesse, para verificar quais combinações proveem os melhores resultados para o sistema estudado (GÉRON, 2017). A representação do PINN (sem o loop de repetição) é representada na Figura 8. Os hiperparâmetros e parâmetros das equações de balanço e de cinética são fixados no início de cada loop de repetição. Na sequência, a rede neural (do PINN) é alimentada com os dados obtidos através do sistema de equações. As saídas do sistema (marcadas como OUT na figura) são as variáveis de interesse (X, P, S e V) e suas respectivas derivadas ao longo do tempo. A função loss, portanto, é calculada comparando as derivadas em cada ponto com os valores teóricos que deveriam

possuir (obtido a partir dos modelos matemáticos). Para isso, é feita a subtração entre o valor da derivada prevista pela NN em relação ao valor calculado, conforme indicado na equação 1.

Figura 8 - Modelo esquemático do funcionamento de um PINN



Fonte: Autoria Própria (2022)

No trabalho que introduziu o conceito, aplicação e validação de PINNs (RAISSI; PERDIKARIS; KARNIADAKIS, 2019), foram levantados alguns questionamentos, como o quão *deep* (profunda, se refere à quantidade de camadas na *hidden layer*) ou *wide* (larga, se refere relação à quantidade de neurônios por camada) deve ser a NN para que represente adequadamente os modelos matemáticos estudados, a relevância da normalização e/ou adimensionalização e como os pesos da função *loss* podem impactar a própria função loss e a perfomance do modelo produzido após o treino. Como essas questões ainda não foram substancialmente respondidas e mais estudos são necessários (SANTANA et al., 2022), alguns desses questionamentos também foram avaliados neste trabalho. Para isso, foram comparados um método de solução numérica tradicional e simples (Método de Euler), dados experimentais (ALTIOK; TOKATLI; HARSA, 2006) e as soluções geradas pelos diversos modelos PINN estudados. Em especial, são discutidas estratégias para a redução do erro (loss), melhoria da performance e é avaliada a complexidade dos modelos propostos e três modelos de operação de um mesmo reator: batelada, batelada alimentada e CSTR.

4.6 Configurações de Simulação

As principais questões a serem respondidas ao longo dos experimentos computacionais são:

- É possível obter soluções com boa precisão para equações simples (equações diferenciais ordinárias, ODEs) com modelos PINNs usando poucas camadas e um baixo número de neurônios por camada? A equação de Burgers, considerada relativamente simples, precisou de pelo menos 20 neurônios em algumas camadas (LU et al., 2021);
- Os pesos aplicados na função loss para cada variável impactam significativamente na precisão e no número de iterações necessárias na etapa de treino da NN para produção de resultados aceitáveis?
- Quais os impactos da adimensionalização da variável independente (tempo) na função loss?
- Como a adimensionalização das variáveis dependentes (X, P, S, V) afeta o sistema? A adimensionalização de algumas delas é mais relevante que das demais?
- Como o mesmo modelo matemático está sendo usado para os três casos estudados (batelada, batelada alimentada e CSTR), a melhor configuração (conjunto de parâmetros e hiperparâmetros) para o modelo batelada também é capaz de produzir modelos PINN que representem adequadamente as operações batelada alimentada e CSTR?
- É possível encontrar um padrão entre hiperparâmetros, pesos da função loss e scalers (fatores de adimensionalização) capazes de melhorar a grid search tradicional, que é de natureza um tanto randômica, e substituí-la por uma determinação de hiperparâmetros e parâmetros ótimos mais racional e previsível?

Para a realização dos testes, a operação de cada reator tem um estado inicial, volume inicial e valores de entrada e saída diferentes. Todas as concentrações inciais, bem como todas as concentrações nas correntes de alimentação são iguais aos seus respectivos valores iniciais (X_o, P_o e S_o) indicados na Tabela 1. Os valores dos parâmetros matemáticos para cada reator são representados na Tabela 3. V_o é o volume em cada reator em t = 0 e t_{sim} é o tempo ao longo do qual o sistema foi simulado. O reator batelada, por definição, tem saídas e entradas fechadas (matematicamente, equivale aos valores de vazão das

correntes serem iguais a zero). Ao reator batelada alimentada, permitiu-se que o volume de líquido crescesse indefinidamente, para que fossem avaliadas as possíveis implicações. O reator CSTR inicia em estado transiente e deve convergir a estado estacionário após algum tempo de operação. Para isso, a vazão de saída é regulada conforme indicada pela equação 16. A vazão de entrada f_{in} dos três modelos também foi mantida diferente. A do CSTR foi reduzida para propositalmente aumentar o tempo necessário para o alcance do estado estacionário e verificar, com possivelmente mais clareza e detalhes, a predição do modelo PINN ao longo do tempo.

Tabela 3 - Parâmetros matemáticos para cada reator

Símbolo	**Valor**		
	Batelada	**Batelada Alimentada**	**CSTR**
f_{in}	0	2 L/h	1 L/h
f_{out}	0	0	Equação 16
V_o	5 L	1 L	1 L
t_{sim}	10,6 h	10,6 h	96 h
X_{in}	X_o	X_o	X_o
P_{in}	P_o	P_o	P_o
S_{in}	S_o	S_o	S_o

Fonte: Autoria Própria (2022)

A função de saída do CSTR é representada abaixo. Ela foi criada de tal modo que simule uma rápida abertura da corrente de saída até que a vazão de saída se iguale a vazão de entrada, à medida em que o volume se aproxima do volume máximo útil do reator (5 L):

$$f_{\text{in}} * \left(V / V_{max} \right)^7 \tag{16}$$

onde V_{max} é o volume máximo útil do reator.

Todas as simulações foram executadas com os pesos da função loss (w_V, w_X, w_P, w_S) = 1, todos os scalers (fatores de adimensionalização) (t_S, X_S, P_S, S_S, V_S) = 1, número de pontos de treino = 800, número de pontos de teste = 1000, rede neural

com 3 camadas com 22 neurônios cada, epochs (número de iterações no algoritmo Adam (KINGMA; BA, 2017)) = 30.000, taxa de aprendizado l_r = 1×10^{-3}, exceto quando for explicitamente dito o contrário. Os modelos PINN foram produzidos e otimizados usando a biblioteca DeepXDE (LU et al., 2021). Como função de ativação foi empregada a função de ativação tangente hiperbólica. Por relacionar os pesos internos da rede entre neurônios com valores sempre entre -1 e 1 (por ser uma tangente hiperbólica), a função tende a tornar os fatores de contribuição mais ou menos normalizados e pode promover a convergência mais rapidamente (GÉRON, 2017; PANNEERSELVAM, 2021). A função tangente hiperbólica é dada a seguir:

$$\tanh(z)=\frac{2}{1+e^{-2z}}-1 \tag{17}$$

onde x representa o input(entrada) de dados para aquele neurônio, vindo dos neurônios da camada anterior.

5 RESULTADOS E DISCUSSÃO

Para buscar padrões e determinar a influência dos diversos fatores já citados (como pesos da função loss e fatores de adimensionalização), os testes de diversos hiperparâmetros e configurações de PINN foram feitos de forma planejada e um-a-um em vez de conduzir uma grid search randômica tradicional, onde uma miríade de parâmetros é selecionada, todas as combinações possíveis são iteradas e o resultado com o menor erro (menor valor de loss) é escolhido como o modelo ótimo. Todas as representações gráficas com a Loss empregaram a loss dos pontos de teste. Ou seja, o modelo foi treinado com os pontos de train e a loss foi obtida a partir dos pontos de teste, conforme explicado na seção 3.5.

Para melhor compreensão do estudo, foi adotada uma nomenclatura explicativa. A palavra “teste” se refere a cada teste que buscou determinar um valor ou responder a uma (ou mais) das perguntas destacadas na seção 4.6. A palavra “caso” se refere a cada subconjunto de parâmetros (configuração) presente naquele teste. Dessa forma, cada teste é um conjunto de casos (experimentos computacionais) capaz de fornecer uma melhor compreensão do assunto de interesse. O Quadro 2 explana quais parâmetros foram testados em cada teste e caso. A coluna “Parâmetros Padrão” exibe os parâmetros padrão para cada teste (os que serão usados caso não sejam indicados na coluna Parâmetros). A coluna “Parâmetros” indica os parâmetros, caso se aplique, que foram aplicados naquele caso e naquele teste. Todos os valores não citados são conforme os descritos na seção 4.6. Devido à dificuldade de convergência nos modelos batelada alimentada e CSTR, o teste *weight*, além da otimização através do número de iterações Adam definido, também passou por uma etapa prévia e outra etapa posterior (à iteração empregando o algoritmo Adam) de otimização pelo algoritmo L-BFGS (BYRD et al., 1995), disponível nativamente na DeepXDE.

Quadro 2 - Parâmetros de cada caso

Test Name	**Parâmetros Padrão**	**Caso**	**Parâmetros**
t_S	3 camadas com	case t_1	t_S = tempo do experimento (9 h)

	22 neurônios cada 30.000 epochs	case t_2	$t_S = 1 / (\mu_{max} * S_o / (K_S + S_o)$
		case t_3	$t_S = \alpha * S_o * (K_S + S_o) / \mu_{max}$
		case t_4	$t_S = (1/Y_{PS}) * \alpha * (K_S + S_o) / \mu_{max}$
		case t_5	$t_S = 1$
		case t_6	$t_S = 1 / \mu_{max}$
non_dim	3 camadas com 22 neurônios cada 80.000 epochs	n1	$V_S = 1, X_S = 1, P_S = 1, S_S = 1$
		n2	$V_S = f_{in} * 1h, X_S = 1, P_S = 1, S_S = 1$
		n3	$V_S = V_{max}$ (5 L), $X_S = X_o, P_S = P_o, S_S = S_o$
		n4	$V_S = V_{max}$ (5 L), $X_S = X_M, P_S = P_M, S_S = S_o$
		n5	$V_S = 1, X_S = 1, P_S = P_M, S_S = S_o$
		n6	$V_S = V_{max}$ (5 L), $X_S = X_M, P_S = 1, S_S = 1$
layer_size	$t_s = 1$ 30.000 epochs	t_lay1	6 camadas com 8 neurônios cada
		t_lay2	10 camadas com 8 neurônios cada
		t_lay3	14 camadas com 8 neurônios cada
		t_lay4	3 camadas com 16 neurônios cada
		t_lay5	4 camadas com 16 neurônios cada
		t_lay6	6 camadas com 16 neurônios cada
		t_lay7	3 camadas com 22 neurônios cada
		t_lay8	4 camadas com 22 neurônios cada
		t_lay9	5 camadas com 22 neurônios cada
		t_lay10	3 camadas com 32 neurônios cada
		t_lay11	4 camadas com 32 neurônios cada
		t_lay12	5 camadas com 32 neurônios cada
weight	5 camadas com 32 neurônios cada 45.000 epochs $w_V, w_X, w_P, w_S = 1$	W1	$w_V, w_X, w_P, w_S = 1$
		W2	$w_X = 3$
		W3	$w_P = 3$
		W4	$w_S = 3$
		W5	$w_V = 3$
		W6	$w_X = 3, w_V = 3$
		W7	$w_P = 3, w_S = 3$
		W8	$w_X = 2, w_S = 3, w_V = 5$
		W9	$w_X = 5, w_V = 10$

multiple_ config	5 camadas com 32 neurônios cada 80.000 epochs L-BFG-S	Batch	
		Fed-Batch	$V_S = V_{max}$ (5 L), $X_S = X_M$, $P_S = P_M$, $S_S = S_o$
		CSTR	$w_V = 3$ $V_S = V_{max}$ (5 L), $X_S = X_M$
pinn_numeri c_xp	Não se aplica	Experimental Data	Não se aplica
		Euler	Simulação usando o método de Euler com o tempo discretizado em 240 igualmente espaçados
		PINN	5 camadas com 32 neurônios 30.000 epochs

Fonte: Autoria Própria (2022)

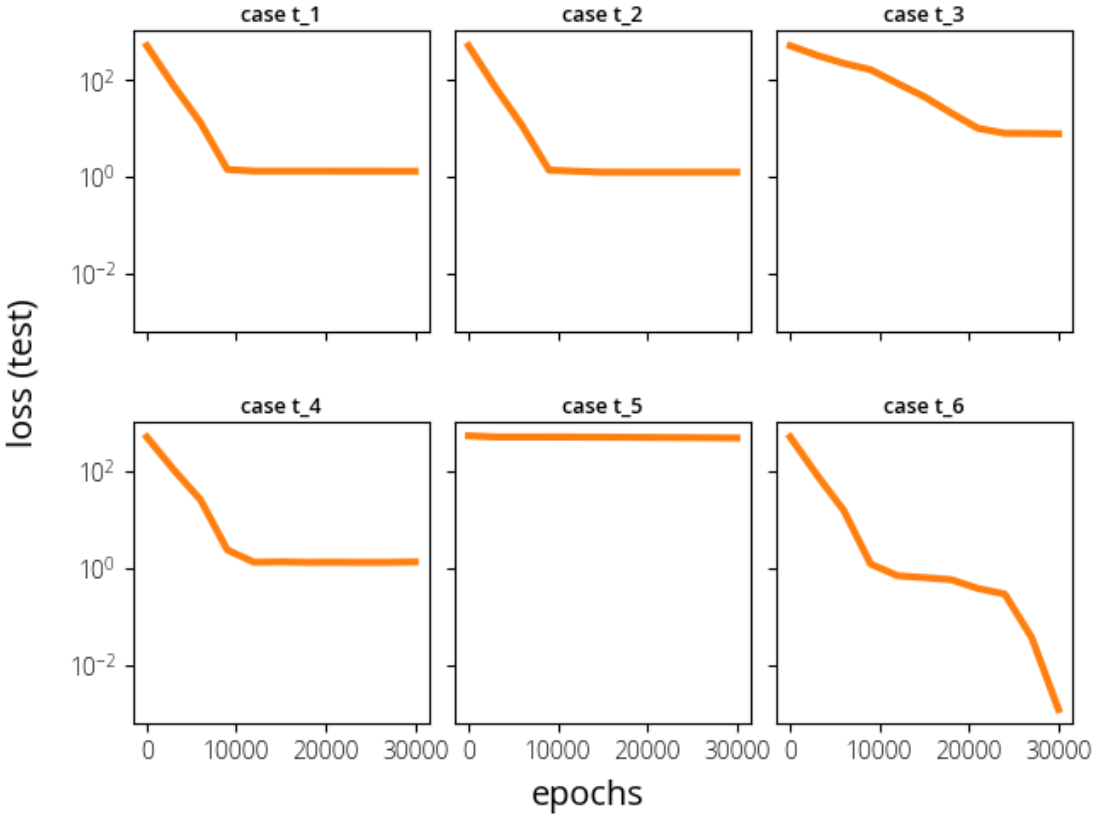
5.1 Testes de Adimensionalização (t_S e non_dim)

O emprego de scalers de adimensionalização pode melhorar substancialmente os resultados da simulação dos modelos CSTR e batelada alimentada, mas, de forma geral, fez com que o modelo batelada apresentasse piores resultados. A variação de t_S não só não melhorou significativamente a simulação no reator em operação batelada, como também aumentou a loss, conforme representado no Gráfico 1. De todos os valores testados para o scaler do tempo, o melhor foi encontrado no caso case ts_6, onde $t_S = 1$. Em outras palavras, o modelo que apresentou melhores resultados foi aquele que não adimensionalizou tempo. A adimensionalização do tempo, de forma geral, aumentou a função loss do modelo e, em diversos casos demonstrados no gráfico, fez com que a NN estagnasse em pontos próximos a loss = 1. Assim, a adimensionalização do tempo foi considerada inefetiva para todos os casos estudados. Deve ser ressaltado, contudo, que em todos os casos t_S era maior ou igual a 1. Assim, é possível que a investigação em outros valores de t_S, menores que 1, mesmo que escolhidos arbitrariamente e não dependendo de constantes (como aqui foi feito) possa resultar em melhores resultados para sistemas com múltipla dependência, uma vez que

várias variáveis dependem de pelo menos uma outra (P e S dependem diretamente de X, e X, P e S dependem de V).

A adimensionalização de X, P, S e V (teste “non_dim”) foi realizada com seis diferentes configurações. Para as configurações de camada e número de neurônios, foram testadas 5 camadas com 32 neurônios cada e 3 camadas com 22 neurônios cada. A loss do CSTR foi reduzida significativamente empregando a menor das duas camadas. Os melhores casos para o CSTR foram n1 e n6. O melhor caso para o batelada alimentada foi o n4, que apresentou loss < 10^{-2}. Apesar da boa performance, ainda ficou muito acima dos valores que normalmente representam bons modelos (loss < 10^{-3}) (SANTANA et al., 2022). Nenhuma das configurações foi capaz de representar adequadamente quaisquer reatores, mesmo com mais de 80.000 epochs. Os erros convergiram ainda antes de 40.0000 epochs e ficaram estagnados.

Gráfico 1 - Resultados do teste de adimensionalização do tempo para reator batelada



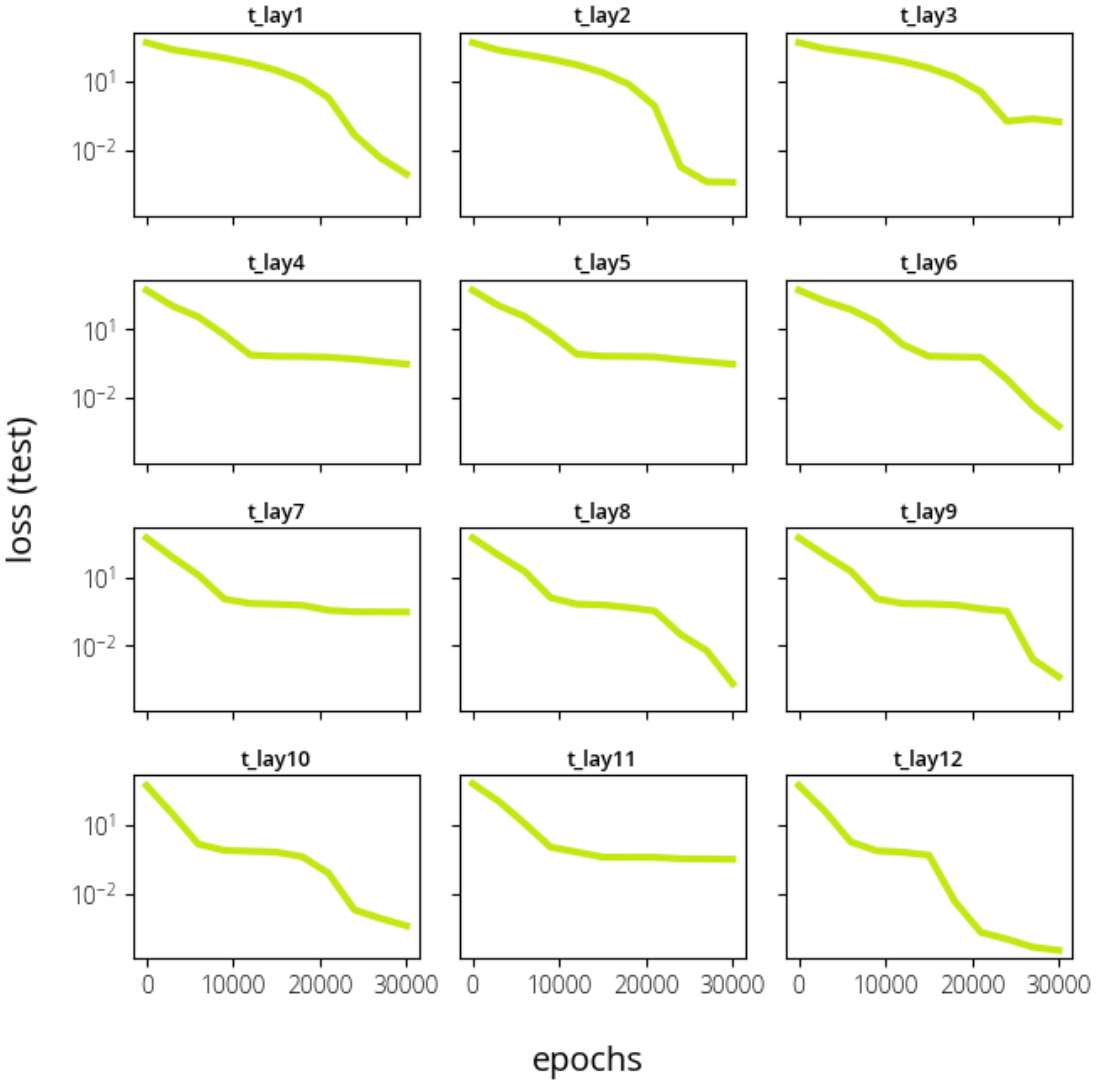
Fonte: Autoria Própria (2022)

5.2 Testes de Largura e Profundidade da Rede (layer_size)

O teste de configuração da camada (layer_size) foi realizado apenas para o reator batelada, e é exibido no Gráfico 2. Foram realizados 12 diferentes testes para averiguar a influência da deepness (profundidade; número de camada) e wideness (largura; número de neurônios por camada) da NN. Com um número constante de 22 neurônios por camada, os casos t_lay7, t_lay8 e t_lay9 variaram apenas o número de camadas em 3, 4 e 5, respectivamente. Enquanto há um grande progresso de 3 para 4 camadas (a zona de estagnação é superada muito mais rapidamente), o uso de 5 camadas não apresenta melhorias significativas, ao passo em que exige que 22 neurônios adicionais sejam treinados. Pode ser inferido, portanto, que um maior número de camadas pode exibir uma redução na função loss, mas pode necessitar de muitas mais epochs (iterações) para ser otimizado e, portanto, pode apresentar uma maior loss para um mesmo número de epochs. Em resumo, aumentar o número de camadas implicou em um custo computacional maior mas que não foi justificado pela performance do modelo produzido.

Os casos t_lay4, t_lay5 e t_lay6 empregaram 3, 4 e 8 camadas respectivamente. Eles mostram que, quando usando um baixo número de neurônios, aumentar a deepness é necessário para reduzir a loss e evitar a estagnação em mínimos locais. Os experimentos t_lay 1 a 3, que possuem relação Neurônios totais / número de camadas (N/L) < 1 produzem resultados melhores que razões N/L > 1 para 8 neurônios. Dos casos t_lay 4 a 6, pode-se inferir que N/L > 4 gera melhores resultados com o aumento do número de camadas, com N/L variando de 5.33 a 2.67. De t_lay 7 a 9 é possível observar que a loss é minimizada conforme N/L vai de 7.33 a 4.4. O mesmo padrão é observado ao se comparar t_lay11 (N/L = 10.67) com t_lay12 (N/L = 6.4). Há uma certa tendência, dentro dos dados estudados, de que N/L ≤ 3 pode gerar modelos melhores para uma quantidade maior de neurônios, mas a regra não pode ser generalizada. A prova disso é que a melhor performance (menor loss) foi alcançada pela t_lay12, o modelo com mais neurônios por camada (5 camadas com 32 neurônios, N/L = 6.4).

Gráfico 2 - Loss (teste) ao longo de epochs para o teste layer_size



Fonte: Autoria Própria (2022)

5.3 Testes de Pesos da Função Loss (weight)

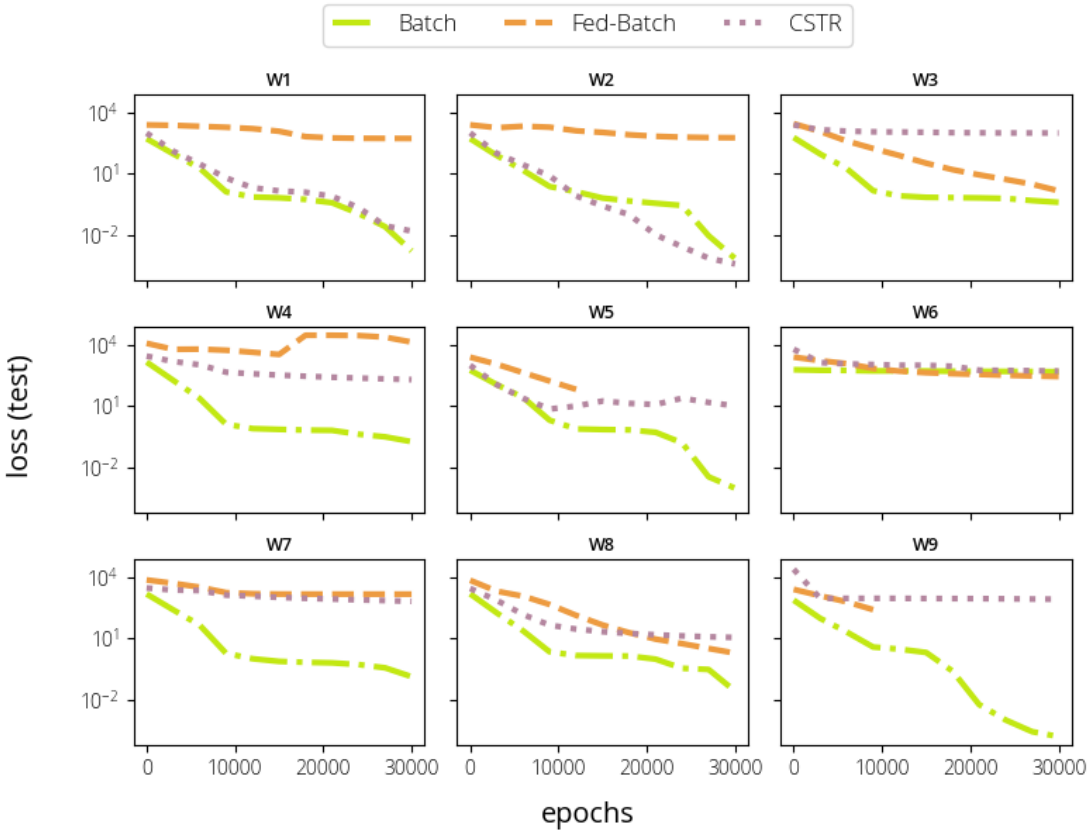
O teste de weights da função loss foram executados em diferentes configurações de NN (6 camadas com 8 neurônios cada, 3 camadas com 22 neurônios cada, 5 camadas com 32 neurônios cada e 3 camadas com 90 neurônios cada). A simulação do reator em batelada demonstrou bons resultados em

praticamente qualquer grupo de parâmetros, com exceção do caso W3, onde $w_P = 3$. O modelo batelada alimentada apresentou a pior performance dentro os três modos de operação em praticamente qualquer um dos casos, mas performou um pouco melhor em W2, com $w_X = 3$. Surpreendentemente, a loss do batelada alimentada foi menos na configuração mais profunda e menos larga (6 camadas com 8 neurônios), com um erro de $8{,}30\times10^{-1}$ após 120.000 epochs. Esse fenômeno pode indicar a presença de um problema do tipo *Vanishing* ou *Exploding gradients*. Esses fenômenos já foram observados empiricamente em outros trabalhos, e foram uma das razões pelas quais redes neurais, em seus primórdios, foram perdendo interesse (GÉRON, 2017). O *vanishing gradient* ocorre quando, devido à semi-normalização causada pela função de ativação, valores muito altos ou muito baixos acabam sendo representados pelos valores de topo e de fundo (no caso da função de ativação tanh [tangente hiperbólica] empregada, +1 e -1) (WANG et al., 2019). Assim, há uma saturação de valores nessa região e, na prática, o treinamento das camadas iniciais é tão lento e requer tantas iterações que pode, na prática, ser inviável ou quase impossível de obter a convergência devido ao custo computacional. A segunda possibilidade é o *exploding gradient*, em que os gradientes de atualização da rede neural oscilam e crescem sem que a NN atinja um mínimo global. Não raramente esses casos geram valores tão grandes que ultrapassam o valor numérico de ponto flutuante disponível (em Python representados, esses valores que não podem ser representados por número são apresentados como NaN – do inglês Not a Number, Não é um Número) (LI, 2022). Nessa etapa, diversos valores foram retornados como NaN durante a etapa de otimização, o que indica um problema do tipo *vanishing gradient* em diversos casos estudados.

O Gráfico 3 representa uma série de gráficos para todos os casos de weight estudados, com uma NN de 3 camadas com 22 neurônios cada. É possível observar uma frequente estagnação (com exceção do modelo batelada) que indica que provavelmente apresenta um problema do tipo *vanishing gradient*, onde as mudanças e atualizações na NN acontecem de forma tão gradual que praticamente estagnam em um mínimo local (W6, W7, W9). O Gráfico 4 revela, por sua vez, uma maior predisposição ao *exploding gradient*, onde o valor estagnado subtamente é alterado e em vários momentos exibe picos que aumentam a loss ao invés de diminuir. As descontinuidades (linhas interrompidas abruptamente) em ambos os

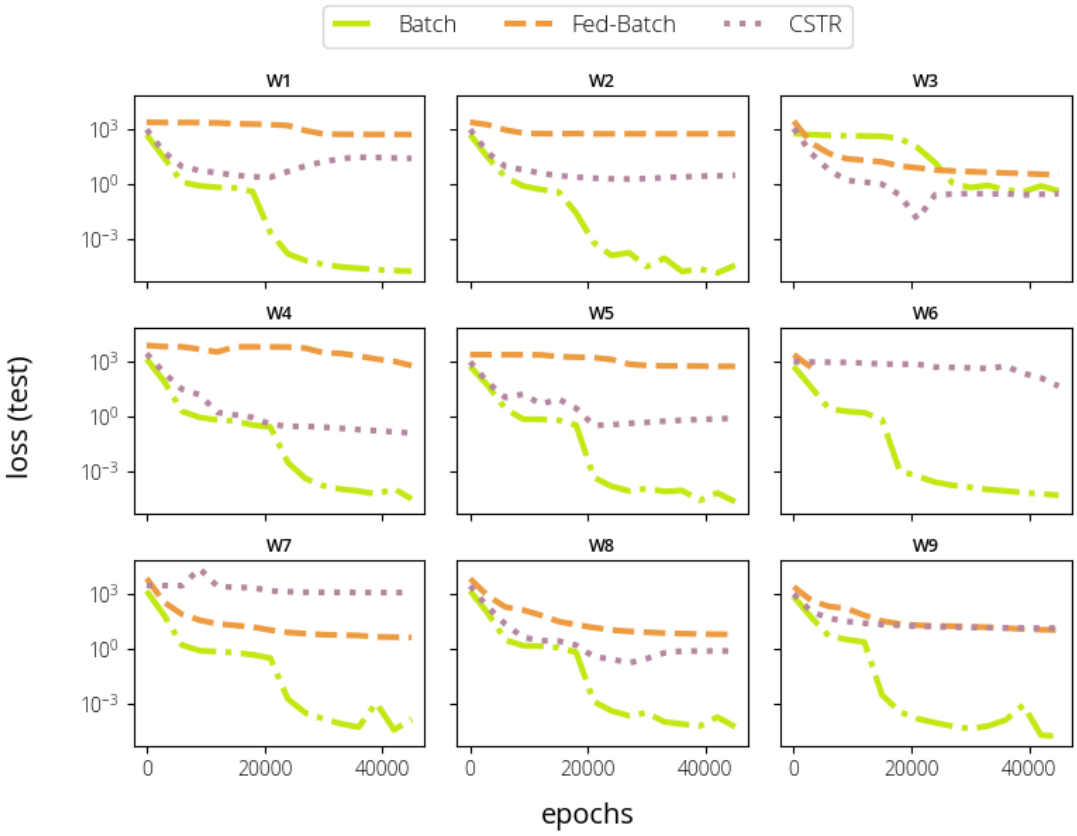
gráficos indicam o passo de iteração (epoch) a partir do qual um valor NaN foi alcançado e o sistema foi interrompido. É válido salientar a tendência, em todos os modelos, de que a simulação do reator em batelada apresente performance consideravelmente melhor do que o reator CSTR, que por sua vez se apresenta levemente melhor do que o reator batelada em diversos casos. Isso indica uma possível tendência à que a variação de volume torne o sistema mais difícil de ser resolvido (mesmo que todos tenham que prever o volume, uma vez que, por mais que o valor seja constante, ele ainda precisa ser previsto pela NN no modelo batelada). Outra tendência é de que a presença de um estado estacionário, em algum momento, no geral, torne o modelo mais fácil de ser resolvido pelo modelo PINN – evidenciado pela melhor performance do CSTR em comparação com o reator batelada alimentada em grande parte dos casos.

Gráfico 3 - Loss para teste weight com 3 camadas com 22 neurônios



Fonte: Autoria Própria (2022)

Gráfico 4 - Loss para test weight com 5 camadas de 32 neurônios



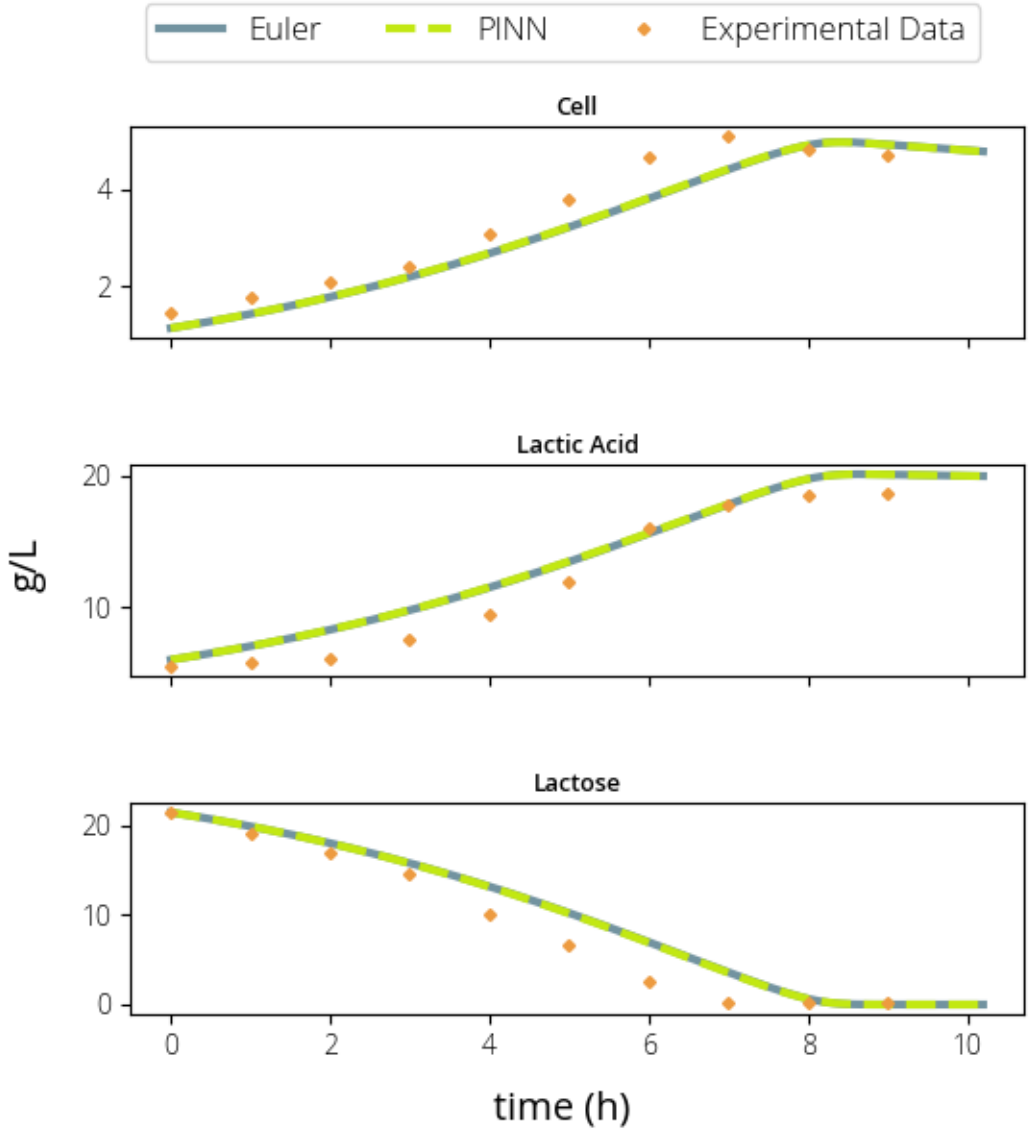
Fonte: Autoria Própria (2022)

O reator batelada é melhor representado por uma das configurações intermediárias de NN (5 camadas com 32 neurônios), apresentando um erro de 1.37×10^{-5} após 45.000 epochs. Em apenas duas simulações o CSTR apresentou performance melhor do que o batelada. A primeira foi com uma configuração onde o CSTR apresentou melhor performance (loss < 10^{-2}) em 3 camadas com 22 neurônios foi no caso W2, que emprega $w_X = 3$. A segunda foi com 32 neurônios e 5 camadas, no caso W3. A única configuração onde o CSTR foi simulado com menor loss do que os modelos batelada e batelada alimentada foi no emprego de 90 neurônios em 3 camadas, otimizado ao longo de 120.000 epochs, mas ainda gerou um erro apreciável (loss > 10). Excluindo esses casos, praticamente todas as outras combinações geraram resultados aceitáveis para o modelo batelada (loss < 10^{-3}), com loss maiores para o modelo CSTR e ainda maiores para o modelo batelada alimentada.

5.4 Teste Comparativo e de Validação (pinn_numeric_xp)

Para validar o modelo de PINN proposto para a representação do reator batelada, a simulação por PINN (5 camadas com 32 neurônios otimizadas por 60.000 epochs de Adam) foi comparada com a solução do sistema de equações diferenciais ordinárias pelo método de Euler (com 240 pontos de discretização do tempo, igualmente espaçados) e com dados experimentais (disponíveis na Tabela 2). O resultado é exibido no Gráfico 5. É possível concluir que o modelo PINN foi capaz de representar com grande fidelidade a cinética da reação estudada. Contudo, o gráfico não responde se o mesmo conjunto de configurações é capaz de gerar um modelo que seja capaz de representar adequadamente os casos com variação de volume, dado que no reator batelada dV/dt = 0. Os dados até aqui discutidos mostram que, de forma geral, um dos principais responsáveis pela dificuldade de convergência nos modelos com variação de volume é a própria variação de volume.

Gráfico 5 - Comparação entre PINN, Método Numérico e Dados experimentais: Reator Batelada



Fonte: Autoria Própria (2022)

5.5 Teste de múltiplas configurações (multiple_config)

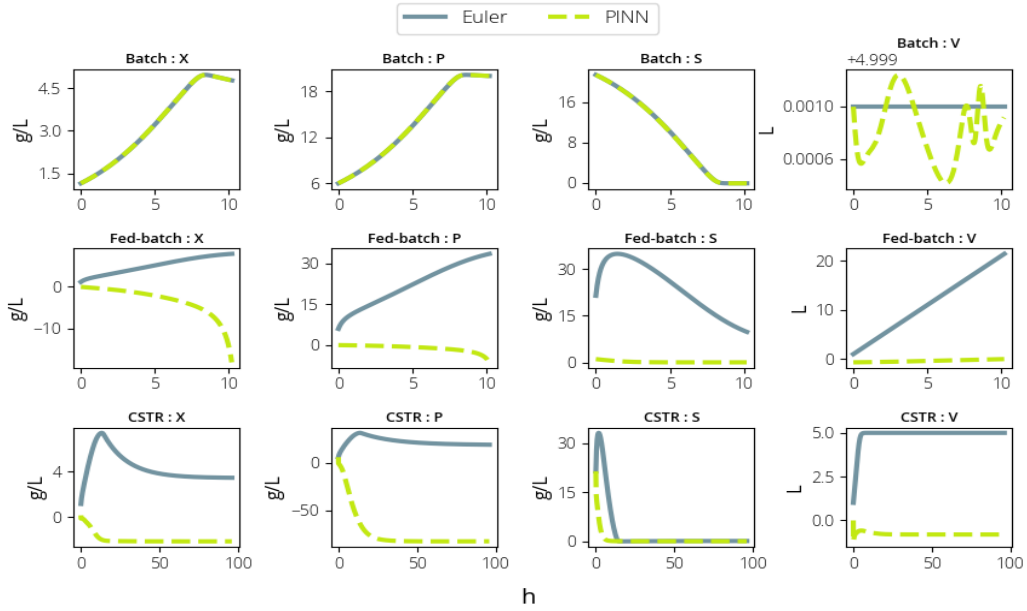
O reator CSTR, embora possua correntes de entrada e de saída e seja inicializado em estado transiente, eventualmente atinge o estado estacionário. Foi observada uma tendência geral de que a existência de variação permanente do volume (batelada alimentada) pode gerar um sistema bastante instável (conforme visto pela ausência de dados por NaN nos gráficos 3 e 4) e dificultar a convergência a um erro mínimo global. Como X depende de V e P e S dependem tanto de X

quanto de V, eles são significantemente afetados quando o sistema é incapaz de prever apropriadamente os valores de V e dV/dt. Os valores de X, P e S praticamente apresentam desvios muito grandes quando a loss do volume previsto é muito elevada. Assim, espera-se que o emprego de um w_V maior seja capaz de aumentar a relevância do volume na produção da função loss e, consequentemente, produzir resultados mais precisos. Contudo, como os pesos não foram capazes de produzir simulações com variação de volume com boa precisão (loss < 10^{-4}), é possível que esse problema, em específico, necessite de um número de neurônios maior do que o que foi testado (>90), bem como mais camadas (>5). Para um problema desse gênero (uma equação diferencial ordinária), contudo, corre-se o risco de gerar um modelo superajustado e que será incapaz de extrapolar valores adequadamente.

Nenhuma das combinações ou configurações apresentadas (disposição da NN, número de neurônios por camada, número de epochs, pesos da função loss e fatores de adimensionalidade) foi capaz de representar os reatores batelada alimentada e CSTR apropriadamente. Assim, foi feito um teste final, que além do algoritmo Adam (KINGMA; BA, 2017) aplicou uma etapa de L-BFGS (BYRD et al., 1995) prévia e outra postterior, combinando essa estratégia com a aplicação dos *weights* da função loss e fatores de adimensionalização que apresentaram os melhores resultados nos testes anteriores. A NN empregada possuía 5 camadas com 32 neurônios e foi otimizada ao longo de 120.000 epochs. O resultado é exibido no Gráfico 6. O nome de cada gráfico interno é construído usando o padrão de nomenclatura “Nome do Reator” : “Nome da Variável”. Batelada:V portanto, representa o volume do reator batelada. É visível que o volume foi previsto com boa precisão (pois deveria ser de exatos 5 litros e a variação é irrisória, pois é mais de 3 ordens de grandeza menor do que o volume total). X, P e S são representados com excelentes resultados em comparação com o modelo numérico. Contudo, ainda é necessário salientar a oscilação persistente ao longo de todos os valores de volume previstos. Em outros estudos, PINNs também apresentaram um comportamento semelhante, onde conseguiram prever duas variáveis adequadamente mas apresentaram ruídos ou erro na predição de uma terceira (MAO; JAGTAP; KARNIADAKIS, 2020). O modelo CSTR exibe um comportamento parecido, onde curiosamente apenas S pode ser representada adequadamente, ao passo em que

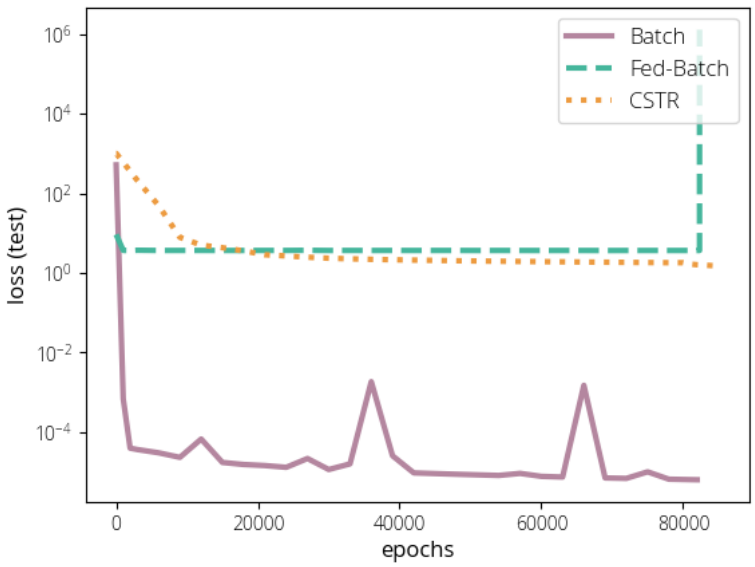
X, P e V apresentam desvios consideráveis. Além disso, os valores de loss para os reatores CSTR e batelada alimentada estagnam em torno de 10^1 (Gráfico 7).

Gráfico 6 - Perfis para Batelada, Batelada Alimentada e CSTR em NN com 5 camadas de 32 neurônios



Fonte: Autoria Própria (2022)

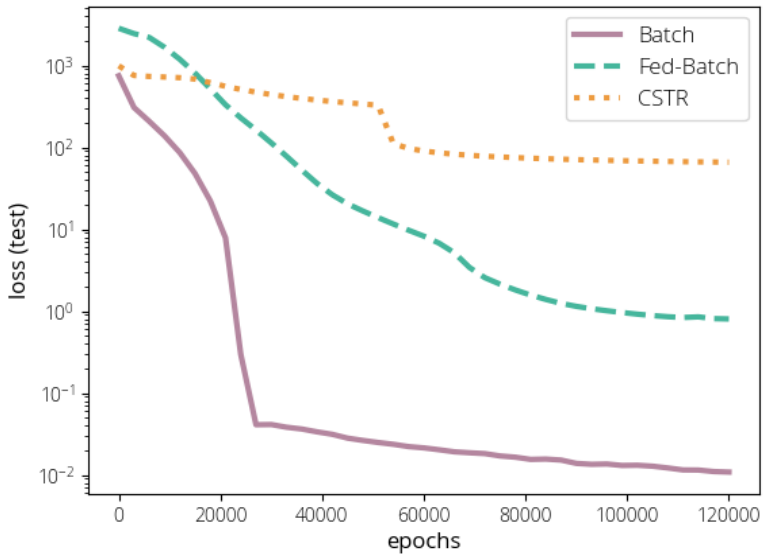
Gráfico 7 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 5 camadas de 32 neurônios



Fonte: Autoria Própria (2022)

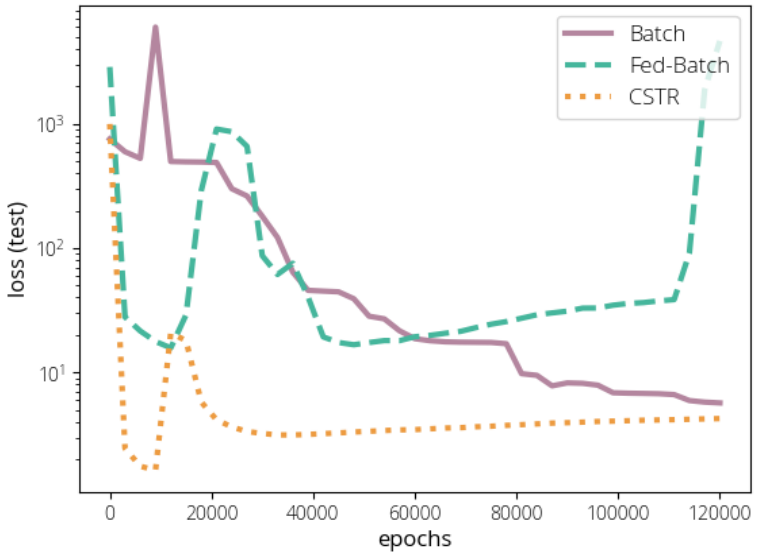
O reator batelada alimentada apresentou uma melhor performance usando redes mais simples e mais profundas (6 camadas com 8 neurônios, Gráfico 8), ao passo em que o reator CSTR apresentou melhor performance em redes mais largas (3 camadas com 90 neurônios, Gráfico 9) dentre todas as testadas. Em ambos os casos, foi possível simular apropriadamente os valores de volume (boa sobreposição com a simulação numérica), mas não necessariamente os das demais variáveis. É possível que o reator batelada alimentada possa ser modelado mais adequadamente em redes mais simples por ser superajustado (ainda que insuficientemente), e um fenômeno semelhante ser responsável pela melhor performance do CSTR em redes maiores e mais densas.

Gráfico 8 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 6 camadas de 8 neurônios



Fonte: Autoria Própria (2022)

Gráfico 9 - Loss para Batelada, Batelada Alimentada e CSTR em NN com 3 camadas de 90 neurônios



Fonte: Autoria Própria (2022)

Um outro ponto importante é que as expressões que representam os efeitos inibitórios de X e P na equação 7 possuem termos exponenciais não inteiros. Embora idealmente X nunca deva ultrapassar o valor de X_M nem P de P_M, é possível que durante o treino e ajuste de pesos internos, a NN faça predição de valores que não respeitem esses máximos. Assim, podem ser geradas expressões negativas elevadas a números não inteiros, o que também pode contribuir com o aparecimento de NaNs. Devido a isso, erros de flutuação de biomassa (X) podem rapidamente se propagar para as equações de P e S, desestabilizando todo o sistema. Muitos testes precisaram ser interrompidos previamente (não publicados neste trabalho) devido a problemas similares. Embora essa discussão possa indicar que o principal fator responsável pela desestabilização do sistema é X, a excelente performance do reator batelada em diversos modelos e que pode ser visualmente atestada no Gráfico 5 revela que X não é a causa pela incapacidade de convergência a uma solução viável nos demais modelos – ao menos, não a única. No modelo batelada, X, P e S são previstos apropriadamente e com erro desprezível. Assim, e com base em todos os outros fatores aqui abordados, é possível concluir que o fator de maior influência na performance do modelo a ser criado é o balanço de volume ao longo do tempo ou o balanço de biomassa, ou ambos. Apesar dessas conclusões, o emprego de w_X e w_V não foi capaz de permitir gerar modelos apropriados (loss < 10^{-}

^{3}) para representar os reatores batelada, batelada alimentada e CSTR a partir de um mesmo conjunto de parâmetros. O emprego dos fatores de X e V foi capaz, ainda, a depender do conjunto de parâmetros, de aproximar a predição dos valores iniciais (X, P, S e V em t = 0) dos valores corretos (X_o, P_o, S_o e V_o), mas nenhum padrão foi observado.

PINNs são uma técnica extremamente recente e muito interessante, e que abrem as portas do ML para muitas aplicações da ciência que não possuem uma miríade de dados disponíveis, mas possuem modelos matemáticos. Tendo em vista a excelente performance no modelo puramente cinético (batelada), é possível propor que uma melhor solução seria empregar os PINNs para modelar reações biológicas e outros fenômenos complexos e de difícil representação matemática, e representar as variáveis físicas ou com balanços bem estabelecidos e relativamente simples (como volume) através de métodos de integração tradicionais. Assim, o sistema proposto seria um híbrido, que empregava o PINN para a solução de interações (reações, adsorção e fenômenos associados) e um método numérico para a solução de balanços mais tradicionais (como volume e massa). O Quadro 3 resume os testes realizados, seus objetivos e conclusões.

Quadro 3 - Conclusões para cada teste

Teste	**Objetivos**	**Conclusão**
t_S	Avaliar o efeito do fator de adimensionalização do tempo, t_S	No geral, quanto maior t_S, maior o erro. O valor de menor erro foi t_S = 1, o que equivale ao sistema sem adimensionalização.
non_dim	Avaliar o efeito dos fatores de adimensionalização X_S, P_S, S_S e V_S	Os valores ótimos para os regimes batelada, batelada alimentada e CSTR variaram consideravelmente, e não foi possível determinar uma regra geral. X_S e V_S se mostram mais relevantes que os demais.
layer_size	1) Avaliar o efeito da profundidade e largura da *hidden layer* nos resultados produzidos pelo modelo 2) Verificar a possibilidade de	1) NNs complexas (mais de 4 camadas ou mais de 60 neurônios por camada) necessitam de muito mais etapas de iteração (epochs) para minimizar a loss e convergir a um valor mínimo, mas nem sempre conseguiram atingir valores aceitáveis. Algumas Nns mais simples (3 camadas com 32 neurônios e 4 camadas com 22 neurônios) foram

	racionalizar a taxa de neurônios por camada (N/L) em função da loss	capazes de produzir bons resultados para a simulação do reator em batelada. 2) Não foi encontrada nenhuma relação clara entre a loss, a fração N/L e/ou o número total de neurônios.
weight	Avaliar o impacto dos weights (pesos) da função loss em todos os modos de operação do reator	Não foi encontrado um conjunto de pesos capaz de produzir modelos de loss reduzida para todos os modos de operação. Assim como no teste non_dim, w_X e w_V se mostraram mais relevantes, sobretudo para os casos com variação de volume.
multiple_config	Avaliar se uma mesma configuração é capaz de produzir modelos que representem apropriadamente os modos de operação do reator.	Praticamente todas as configurações testadas conseguem produzir resultados de excelente fidelidade no reator batelada, e praticamente nenhuma foi capaz de reproduzir de modo aceitável (loss < 10^{-2}) os modelos com variação de volume.
pinn_numeric_xp	Validar o modelo proposto para o reator em batelada, comparando a solução em PINN com a solução pelo método de Euler e com dados experimentais	O modelo foi capaz de simular de forma excelente (loss < 10^{-4}) o reator em batelada. O gráfico não só se sobrepõe à solução numérica como é praticamente idêntico ao do trabalho original (ALTIOK; TOKATLI; HARSA, 2006).

Fonte: Autoria Própria (2022)

6 CONCLUSÃO

PINNs com Redes Neurais mais simples (menos neurônios por camada e menos camadsa) foram capazes de simular adequadamente (loss < 10^{-4}) o modelo de reator batelada. Contudo, esses modelos demonstraram má adequação para representar o equacionamento cinético e a variação de volume (modelos CSTR e batelada alimentada) simultaneamente. Dentre todos os modelos testados, o modelo que apresentou a menor loss para o reator batelada empregou 5 camadas com 32 neurônios e precisou de aproximadamente 30.000 epochs para convergir para um mínimo global (que se manteve até 120.000 epochs) e não necessitou de estratégias de adimensionalização. Em praticamente todos os casos em que dV/dt ≠ 0 A adimensionalização de X e V com $X_S = X_M$ e $V_S = V_{reator}$ foi capaz de reduzir a loss em mais de uma ordem de grandeza, mas a loss total ainda se mostrou muito superior à do reator batelada, que era menor que 10^{-3}. Como o modelo PINN se provou bastante robusto para a solução de cinétia de reação (batelada) e todos os PINNs empregados necessitaram de muitas iterações (>120.000 epochs) para os reatores CSTR e batelada alimentada e mesmo assim geraram erros apreciáveis, é possível propor que uma melhor alternativa seria o desenvolvimento de um modelo híbrido. Nele, PINNs seriam empregados para resolver conjuntos de equações químicas ou biológicas, que são intrinsecamente mais complexas, e um método numérico tradicional, como Euler ou Runge-Kutta, seria usado para resolver os fenômenos físicos do sistema, que no geral são mais bem estabelecidos e de modelagem mais simples.

PINNs foram capazes de simular apropriadamente uma reação biológica com termos complexos de inibição por produto, substrato e biomassa. Contudo, nenhum dos conjuntos de parâmetros testados foi capaz de representar simultaneamente X, P e S em um reator com variação de V. Para trabalhos futuros, a recomendação é verificar o impacto de t_S (fator de adimensionalização do tempo) menores que 1, ainda que o t_S proposto seja arbitrário e não baseado em constantes das equações. Para continuar explorando a possibilidade do emprego de PINNs em biorreações, é recomendado ainda, além de se destinar bastante atenção às variáveis independentes (neste trabalho, o tempo t) e às variáveis que influenciam outras

variáveis (neste trabalho, a concentração de biomassa, X, e o volume, V) uma vez que essa influência pode ser responsável pelo sucesso ou não do modelo proposto.

7 REFERÊNCIAS

ALHAMA MANTECA, I.; SOTO MECA, A.; ALHAMA, F. Mathematical characterization of scenarios of fluid flow and solute transport in porous media by discriminated nondimensionalization. **International Journal of Engineering Science**, v. 50, n. 1, p. 1–9, jan. 2012.

ALTAF, MD.; NAVEENA, B. J.; REDDY, G. Use of inexpensive nitrogen sources and starch for l(+) lactic acid production in anaerobic submerged fermentation. **Bioresource Technology**, v. 98, n. 3, p. 498–503, fev. 2007.

ALTIOK, D.; TOKATLI, F.; HARSA, Ş. Kinetic modelling of lactic acid production from whey by Lactobacillus casei (NRRL B-441). **Journal of Chemical Technology & Biotechnology**, v. 81, n. 7, p. 1190–1197, jul. 2006.

ALZUBI, J.; NAYYAR, A.; KUMAR, A. Machine Learning from Theory to Algorithms: An Overview. **Journal of Physics: Conference Series**, v. 1142, p. 012012, nov. 2018.

ANDRADE CRUZ, I. et al. Application of machine learning in anaerobic digestion: Perspectives and challenges. **Bioresource Technology**, v. 345, p. 126433, fev. 2022.

ARCANJO, M. R. A.; FERNANDES, F. A. N.; SILVA, I. J. Separation of Lactic Acid Produced by Hydrothermal Conversion of Glycerol Using Ion-Exchange Chromatography. **Adsorption Science & Technology**, v. 33, n. 2, p. 139–151, fev. 2015.

BAGHERZADEH, F. et al. Comparative study on total nitrogen prediction in wastewater treatment plant and effect of various feature selection methods on machine learning algorithms performance. **Journal of Water Process Engineering**, v. 41, p. 102033, jun. 2021.

BARRY, P. **Head first Python**. Second edition ed. Sebastopol, California: O'Reilly, 2017.

BYRD, R. H. et al. A Limited Memory Algorithm for Bound Constrained Optimization. **SIAM Journal on Scientific Computing**, v. 16, n. 5, p. 1190–1208, set. 1995.

DATTA, R. et al. Technological and economic potential of poly(lactic acid) and lactic acid derivatives. **FEMS Microbiology Reviews**, v. 16, n. 2–3, p. 221–231, fev. 1995.

DEBARROS, A. **Practical SQL: a beginner's guide to storytelling with data**. San Francisco: No Starch Press, 2018.

DEY, P.; PAL, P. Modelling and simulation of continuous L (+) lactic acid production from sugarcane juice in membrane integrated hybrid-reactor system. **Biochemical Engineering Journal**, v. 79, p. 15–24, out. 2013.

DIN, N. A. S. et al. Lactic acid separation and recovery from fermentation broth by ion-exchange resin: A review. **Bioresources and Bioprocessing**, v. 8, n. 1, p. 31, dez. 2021.

DORAN, P. M. **Bioprocess engineering principles**. 2nd ed ed. Amsterdam ; Boston: Elsevier/Academic Press, 2013.

FOGLER, H. S. **Essentials of chemical reaction engineering**. Second edition ed. Boston: Prentice Hall, 2018.

GÉRON, A. **Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems**. First edition ed. Beijing ; Boston: O'Reilly Media, 2017.

GUILHERME, A. D. A. OTIMIZAÇÃO DA PRODUÇÃO DE ÁCIDO LÁTICO POR Lactobacillus casei NRRL B-442 EM SUCO DE CAJU CLARIFICADO. [s.d.].

HAMAMCI, H.; RYU, D. D. Y. Production of L(+)-lactic acid using immobilized rhizopus oryzae reactor performance based on kinetic model and simulation. **Applied Biochemistry and Biotechnology**, v. 44, n. 2, p. 125–133, fev. 1994.

HELLER, S. R. et al. InChI, the IUPAC International Chemical Identifier. **Journal of Cheminformatics**, v. 7, n. 1, p. 23, 30 maio 2015.

HETLAND, M. L. **Python algorithms: mastering basic algorithms in the Python language**. Second edition ed. New York City, NY: Apress, 2014.

JANA, A. K. **Chemical process modelling and computer simulation**. 2. ed ed. New Delhi: PHI Learning, 2011.

JANOSKA, A.; BUIJS, J.; VAN GULIK, W. M. Predicting the influence of combined oxygen and glucose gradients based on scale-down and modelling approaches for the scale-up of penicillin fermentations. **Process Biochemistry**, v. 124, p. 100–112, jan. 2023.

JOHNS, W. Computer-Aided Chemical Engineering. Em: JOHN WILEY & SONS, INC. (Ed.). **Kirk-Othmer Encyclopedia of Chemical Technology**. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2011. p. 0315131620012525.a01.pub3.

KINGMA, D. P.; BA, J. **Adam: A Method for Stochastic Optimization**. arXiv, , 29 jan. 2017. Disponível em: <http://arxiv.org/abs/1412.6980>. Acesso em: 23 jan. 2023

KOMESU, A.; MACIEL, M. R. W.; FILHO, R. M. Lactic Acid Production to Purification: A Review. p. 20, 2017.

KRISHNA, A. **Object-Oriented Programming in Python**. Disponível em: <https://www.freecodecamp.org/news/object-oriented-programming-in-python/>. Acesso em: 6 dez. 2022.

LEE, E. G. et al. Lactic acid recovery using two-stage electrodialysis and its modelling. **Journal of Membrane Science**, v. 145, n. 1, p. 53–66, jun. 1998.

LI, B. et al. Application of mechanistic modelling and machine learning for cream cheese fermentation pH prediction. **Journal of Chemical Technology & Biotechnology**, v. 96, n. 1, p. 125–133, jan. 2021.

LI, K. (YI). **Vanishing and Exploding Gradients in Neural Network Models: Debugging, Monitoring, and Fixing**. Disponível em: <https://neptune.ai/blog/vanishing-and-exploding-gradients-debugging-monitoring-fixing>. Acesso em: 15 fev. 2023.

LI, Y.; XU, J. A PDF discretization scheme in wavenumber-frequency joint spectrum for simulating multivariate random fluctuating wind fields. **Probabilistic Engineering Mechanics**, p. 103422, jan. 2023.

LIM, S. J. et al. Opportunities and challenges of machine learning in bioprocesses: Categorization from different perspectives and future direction. **Bioresource Technology**, v. 370, p. 128518, fev. 2023.

LÓPEZ-GÓMEZ, J. P. et al. A review on the current developments in continuous lactic acid fermentations and case studies utilising inexpensive raw materials. **Process Biochemistry**, v. 79, p. 1–10, abr. 2019.

LU, L. et al. DeepXDE: A Deep Learning Library for Solving Differential Equations. **SIAM Review**, v. 63, n. 1, p. 208–228, jan. 2021.

MAO, Z.; JAGTAP, A. D.; KARNIADAKIS, G. E. Physics-informed neural networks for high-speed flows. **Computer Methods in Applied Mechanics and Engineering**, v. 360, p. 112789, mar. 2020.

MARTIN, R. C. (ED.). **Clean code: a handbook of agile software craftsmanship**. Upper Saddle River, NJ: Prentice Hall, 2009.

MATEO PÉREZ, V. et al. A Random Forest Model for the Prediction of FOG Content in Inlet Wastewater from Urban WWTPs. **Water**, v. 13, n. 9, p. 1237, 29 abr. 2021.

MCCORMACK, C. M.; CALDWELL, B. S. Learner-Centered Design of Online Courses: A Transdisciplinary Systems Engineering Case Design. Em: MOSER, B. R.; KOOMSAP, P.; STJEPANDIĆ, J. (Eds.). **Advances in Transdisciplinary Engineering**. [s.l.] IOS Press, 2022.

MEY, F. et al. Improving the performance of machine learning models for biotechnology: The quest for deus ex machina. **Biotechnology Advances**, v. 53, p. 107858, dez. 2021.

MONNUS, A. **Using OOP concepts to write high-performance Java code**. Disponível em: <https://raygun.com/blog/oop-concepts-java/>. Acesso em: 6 dez. 2022.

PANDEY, A. K. et al. Machine learning in fermentative biohydrogen production: Advantages, challenges, and applications. **Bioresource Technology**, v. 370, p. 128502, fev. 2023.

PANNEERSELVAM, L. **Activation Functions | What are Activation Functions**. **Analytics Vidhya**, 14 abr. 2021. Disponível em: <https://www.analyticsvidhya.com/blog/2021/04/activation-functions-and-their-derivatives-a-quick-complete-guide/>. Acesso em: 15 fev. 2023

PERLINGEIRO, C. A. G. **Engenharia de Processos: Análise, simulação, otimização e síntese de processos químicos**. 2. ed. São Paulo: Blucher, 2018.

PRADHAN, N. et al. Kinetic modeling of hydrogen and L-lactic acid production by Thermotoga neapolitana via capnophilic lactic fermentation of starch. **Bioresource Technology**, v. 332, p. 125127, jul. 2021.

QUINTERO, J. et al. fermentative process of cassava syrup using ion exchange resins. p. 14, 2012.

RAISSI, M.; PERDIKARIS, P.; KARNIADAKIS, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. **Journal of Computational Physics**, v. 378, p. 686–707, fev. 2019.

REMINI, A.; ROSATI, L. A Brief History of Information Architecture. **Journal Of Information Architecture**, v. 3, p. 14, 2011.

ROSENFELD, L.; MORVILLE, P.; ARANGO, J. **Information architecture: for the web and beyond**. Fourth edition ed. Sebastopol, CA: O'Reilly Media, Inc, 2015.

SALVAÑAL, L. et al. l-lactic acid production using the syrup obtained in biorefinery of carrot discards. **Food and Bioproducts Processing**, v. 127, p. 465–471, maio 2021.

SANTANA, V. V. et al. A First Approach towards Adsorption-Oriented Physics-Informed Neural Networks: Monoclonal Antibody Adsorption Performance on an Ion-Exchange Column as a Case Study. **ChemEngineering**, v. 6, n. 2, p. 21, 1 mar. 2022.

SANTANAM. **Let's get classy: how to create modules and classes with Python**. Disponível em: <https://www.freecodecamp.org/news/lets-get-classy-how-to-create-modules-and-classes-with-python-44da18bb38d1/>. Acesso em: 6 dez. 2022.

SULLIVAN, D. **NoSQL for mere mortals**. Hoboken, NJ: Addison-Wesley, 2015.

WANG, X. et al. ReLTanh: An activation function with vanishing gradient resistance for SAE-based DNNs and its application to rotating machinery fault diagnosis. **Neurocomputing**, v. 363, p. 88–98, out. 2019.