UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"
Câmpus de Presidente Prudente - SP

Diego Alecsander de Aguiar

Predição do Balanço de Energia na Dinâmica de Gotículas: Uma Abordagem com Redes Neurais Recorrentes

Presidente Prudente - SP

2024

Diego Alecsander de Aguiar

Predição do Balanço de Energia na Dinâmica de Gotículas: Uma Abordagem com Redes Neurais Recorrentes

Orientador: Prof. Dr. Cassio Machiaveli Oishi

Dissertação apresentada como parte dos requisitos para obtenção do título de Mestre em Ciência da Computação, junto ao Programa de Pós-Graduação em Ciência da Computação, do Universidade Estadual Paulista (UNESP) da Universidade Estadual Paulista "Júlio de Mesquita Filho", Câmpus de Presidente Prudente - SP.

Financiadora: DS/CAPES

Presidente Prudente - SP 2024

A282p

Aguiar, Diego Alecsander de
Predição do balanço de energia na dinâmica de gotículas: uma abordagem com redes neurais recorrentes / Diego Alecsander de Aguiar. -- Presidente Prudente, 2024
90 p. : il., tabs.

Dissertação (mestrado) - Universidade Estadual Paulista (UNESP), Faculdade de Ciências e Tecnologia, Presidente Prudente
Orientador: Cassio Machiaveli Oishi

1. Inteligência artificial. 2. Dinâmica dos fluidos. I. Título.

Sistema de geração automática de fichas catalográficas da Unesp. Biblioteca da Universidade Estadual Paulista (UNESP), Faculdade de Ciências e Tecnologia, Presidente Prudente. Dados fornecidos pelo autor(a).

Essa ficha não pode ser modificada.

UNIVERSIDADE ESTADUAL PAULISTA

Câmpus de Presidente Prudente

ATA DA DEFESA PÚBLICA DA DISSERTAÇÃO DE MESTRADO DE DIEGO ALECSANDER DE AGUIAR, DISCENTE DO PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO, DA FACULDADE DE CIÊNCIAS E TECNOLOGIA - CÂMPUS DE PRESIDENTE PRUDENTE.

Aos 24 dias do mês de abril do ano de 2024, às 09:00 horas, por meio de Videoconferência, realizou-se a defesa de DISSERTAÇÃO DE MESTRADO de DIEGO ALECSANDER DE AGUIAR, intitulada **Predição do Balanço de Energia na Dinâmica de Gotículas: Uma Abordagem com Redes Neurais Recorrentes**. A Comissão Examinadora foi constituida pelos seguintes membros: Prof. Dr. CÁSSIO MACHIAVELI OISHI (Orientador(a) - Participação Virtual) do(a) Departamento de Matemática e Computação / Faculdade de Ciencias e Tecnologia de Presidente Prudente, Prof. Dr. DANIEL CARLOS GUIMARÃES PEDRONETTE (Participação Virtual) do(a) IGCE / UNESP / Rio Claro (SP), Prof. Dr. DIEGO SAMUEL RODRIGUES (Participação Virtual) do(a) Faculdade de Tecnologia / Universidade Estadual de Campinas. Após a exposição pelo mestrando e arguição pelos membros da Comissão Examinadora que participaram do ato, de forma presencial e/ou virtual, o discente recebeu o conceito final:_ _ _Aprovado_ _ _ _ _ . Nada mais havendo, foi lavrada a presente ata, que após lida e aprovada, foi assinada pelo(a) Presidente(a) da Comissão Examinadora.

Prof. Dr. CÁSSIO MACHIAVELI OISHI

gov.br Documento assinado digitalmente
CASSIO MACHIAVELI OISHI
Data: 24/04/2024 11:17:18-0300
Verifique em https://validar.iti.gov.br

Agradecimentos

Eu gostaria de agradecer primeiramente ao meu orientador, Cassio M. Oishi, por ter me ajudado e me ensinado muito desde a minha graduação. O tempo que passei trabalhando com ele foi o que despertou meu interesse pelo meio acadêmico, espero que possamos continuar seguindo esse caminho.

Agradeço ao Fernando P. Martins que, embora não esteja diretamente envolvido com o projeto, me auxiliou em diversas etapas, principalmente na utilização do cluster do Laboratório de Simulação Numérica (LSN).

Agradeço também ao Hugo L. França por ter disponibilizado o código base para a execução das simulações numéricas.

Por fim, mas não menos importante, agradeço aos meus pais por todo o apoio emocional que me deram ao longo de todos esses anos. Sem eles, o caminho teria sido muito mais difícil.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

Resumo

Este estudo explora a aplicação de modelos Long Short-Term Memory (LSTM) para prever o balanço de energia (energias cinética, de superfície e de dissipação viscosa) no contexto de dois regimes de escoamento de fluidos distintos e sob diferentes efeitos de viscosidade e tensão superficial: gotículas com formatos iniciais variáveis impactando em superfícies sólidas e duas gotículas coalescendo após colidirem. Além disso, uma predição estática dos números adimensionais também foi explorada. A aplicação de redes neurais LSTM é proposta para lidar com as limitações observadas ao utilizar redes neurais *feedforward* simples para manipular dados transientes. Os modelos foram treinados usando números adimensionais e séries temporais geométricas extraídas de simulações numéricas. Adicionalmente, a necessidade de simular formas não convencionais motivou o desenvolvimento de um método inovador para extrair partículas de interfaces presentes em imagens digitais para a representação por Front-Tracking.

A avaliação de desempenho dos modelos incluiu métricas como o Coeficiente de Determinação, o Erro Quadrático Médio e o Erro Quadrático Médio Normalizado. A análise dessas métricas revela que as predições dos modelos estão profundamente alinhadas com seus valores ideais, destacando a adaptabilidade e precisão da arquitetura escolhida. Dado que os regimes explorados têm múltiplas aplicações significativas, como impressão por jato de tinta, aplicação de pesticidas agrícolas, operação de motores de combustão, transmissão de doenças respiratórias, entre outros, este trabalho abrir caminho para o desenvolvimento de novos modelos de aprendizado de máquina aproveitando o desempenho oferecido pelas arquiteturas LSTM para prever diferentes tipos de dados transientes em outros cenários de escoamento.

Palavras-chave: cálculo de energias, espalhamento de gotículas, colisão de gotículas, aprendizado de máquina.

Abstract

This study explores the application of Long Short-Term Memory (LSTM) models to predict the energy balance (kinetic, surface and viscous dissipation energies) within the context of two distinct fluid flow regimes under different viscous and surface tension effects: droplets with varying initial shapes impacting on solid surfaces and two droplets coalescing after colliding. Additionally, a static prediction of the dimensionless numbers was also explored. The application of LSTM neural networks is proposed to address the limitations observed when utilizing simple Feedforward neural networks for handling transient data. The models were trained using dimensionless numbers and geometric time series data from numerical simulations. Additionally, the need to simulate unconventional shapes prompted the development of a novel method for extracting Front-Tracking particles from interfaces present in digital images.

Evaluation of performance included metrics such as the Coefficient of Determination, Root Mean Square Error, and Normalized Root Mean Square Error. The analysis of these metrics reveals that predictions from the models closely align with their ideal values, emphasizing the adaptability and precision of the chosen architecture. Given that the explored regimes have multiple significant applications, such as inkjet printing, application of agricultural pesticides, operation of combustion engines, transmission of respiratory diseases, among others, this work could pave the way for the development of new machine learning models leveraging the performance offered by LSTM architectures to predict different types of transient data in other flow scenarios.

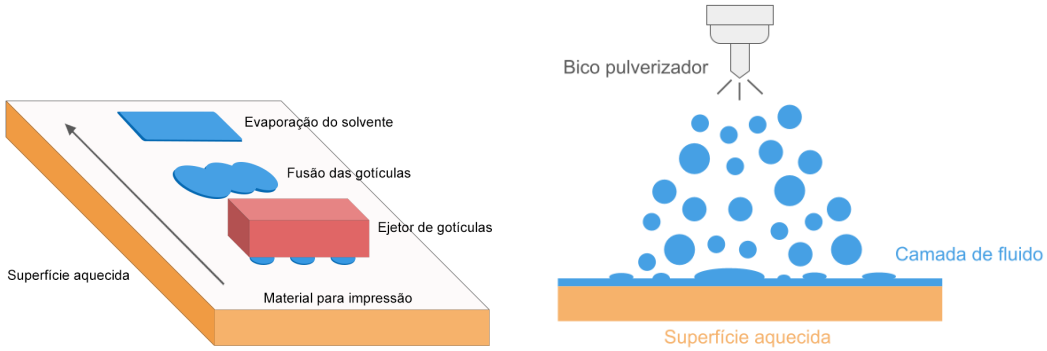
Keywords: energy calculation, droplet spreading, droplet collision, machine learning.

Sumário

1 Introdução

O estudo das dinâmicas de colisão entre gotículas e de espalhamento de gotículas impactando superfícies sólidas é importante para inúmeras aplicações em diferentes áreas da ciência e da engenharia, como a impressão a jato de tinta (Figura 1a), o resfriamento por pulverização (Figura 1b), a aplicação de defensivos agrícolas, o funcionamento de motores de combustão, a transmissão de doenças respiratórias, entre outras [1].

Em particular, a complexidade do processo de impressão a jato de tinta [2] destaca-se por envolver tanto a dinâmica de colisão das gotículas durante sua trajetória quanto o eventual impacto em uma superfície, ilustrando o vínculo existente entre diferentes fenômenos da dinâmica dos fluidos. Compreender o comportamento da colisão é essencial para que seja possível controlar a distribuição, o formato e o tamanho das gotículas de tinta durante a sua queda. Por outro lado, o estudo do espalhamento visa garantir o bom funcionamento de fatores relacionados ao contato com a superfície de impressão, como a velocidade e o ângulo de impacto, as propriedades das gotículas e as características da superfície.



(a) Funcionamento da impressão a jato de tinta. (b) Esquema de resfriamento por pulverização.

Figura 1 – Exemplos de aplicações práticas que envolvem a colisão e o espalhamento de gotas.

Ao longo do tempo, pesquisas sobre tais dinâmicas resultaram em diversas publicações que buscam compreender as particularidades dos diferentes regimes de escoamento. Entre elas, investigações experimentais foram adotadas para encontrar relações entre propriedades dos fluidos tanto no regime de espalhamento [3, 4, 5, 6, 7, 8] como no de colisão entre gotículas [9, 10, 11, 12, 13]. Outras contribuições notáveis são os artigos [14, 15, 16] que abordam, entre outros temas, a conexão entre o espalhamento máximo e os números adimensionais, a relação das energias de um sistema e a caracterização de diferentes regimes antes e depois do impacto de

uma gota.

Como a exploração experimental de problemas práticos nem sempre é viável, as simulações numéricas das dinâmicas mais complexas podem ser alternativas eficientes [17]. Em tais simulações, as equações governantes do escoamento de fluidos podem ser resolvidas através de esquemas de discretização clássicos, como o método dos elementos finitos [18], o método dos volumes finitos [19], o método das diferenças finitas [20], entre outros. Além da solução das equações, outro aspecto importante a considerar é o tratamento adequado das interfaces móveis que separam os fluidos. Esse tratamento envolve representar computacionalmente a interface e possibilitar que ela se mova a cada instante de tempo, além de mitigar eventuais inconsistências que possam surgir durante a simulação. Para isso, existem vários métodos descritos na literatura, incluindo representações explícitas, como o Front-Tracking [21, 22], representações implícitas, como o Volume-of-Fluid (VoF) [23], ou representações híbridas.

Em um trabalho anterior [24], explorou-se a aplicação do método VoF, que utiliza uma malha com frações de volume para representar as interfaces. Algumas vantagens do método incluem a sua implementação relativamente simples e sua capacidade de tratar as mudanças topológicas em sua formulação, sem que haja a necessidade de desenvolver verificações e modificações específicas para esse fim. No entanto, suas limitações incluem a capacidade de representar apenas dois fluidos em sua versão clássica e a necessidade de aplicar um método para reconstrução da interface à partir das frações de volume para visualizar a geometria original da gota.

Para o presente trabalho, decidiu-se utilizar um *solver* que implementa o método Front-Tracking, uma abordagem explícita que representa interfaces por meio de partículas interligadas. Seu principal ponto positivo é lidar com algumas das limitações observadas no VoF, permitindo a representação de mais de dois fluidos e não havendo a necessidade de um método para reconstrução da geometria da interface, uma vez que as próprias partículas podem ser utilizadas para tal. Apesar de sua eficiência, o método possui uma implementação mais complexa, exigindo também o desenvolvimento de uma adaptação para lidar com as mudanças topológicas. Para uma compreensão mais aprofundada sobre o Front-Tracking, uma implementação minimalista do processo de advecção, utilizando apenas o campo de velocidades, foi desenvolvida.

O uso de estratégias de aprendizado de máquina na área de dinâmica dos fluidos tem sido bastante explorado nos últimos anos graças ao surgimento de técnicas mais eficientes e de *fra-*

meworks e bibliotecas, como TensorFlow, Pytorch e MXNet, que simplificam o desenvolvimento, teste e otimização de diferentes arquiteturas de redes neurais. Como as simulações numéricas podem ter um alto custo computacional, a aplicação de aprendizado de máquina para predizer determinadas propriedades ou encontrar correlações entre parâmetros pode ser uma estratégia eficaz para acelerar o processo ou até mesmo eliminar a necessidade de executar uma simulação completa.

No contexto da dinâmica de gotículas, diferentes modelos de aprendizado de máquina têm sido adotados para predizer propriedades físicas ou geométricas observadas em escoamentos complexos, como por exemplo a predição do espalhamento máximo de uma gota através de números adimensionais e propriedades relacionadas a cada regime abordado [25, 26, 27, 28, 29, 30]. A ideia de aplicar propriedades relacionadas às dinâmicas para predizer outras propriedades também foi explorada em diversas outras situações, como na predição de morfologias de gotas a partir de dados de imagens [31], na classificação de impactos com ou sem respingo, utilizando imagens capturadas no momento em que metade da gota atinge a superfície [32], no uso de uma inteligência artificial explicável (XAI) para interpretar como o classificador citado funciona e descobrir uma possível correlação entre a propagação de uma gota e a força de impacto em uma superfície sólida [33], na predição da curvatura da interface através de partículas vizinhas no método de representação Front-Tracking [34], na predição do tamanho, velocidade e forma de gotículas de um jato de metal líquido sob demanda (DoD-LMJ) [35], na predição de características geométricas de jatos de fluido, como o ângulo de espalhamento e o comprimento laminar, através de fatores físicos, como o número de Reynolds (Re) e o número de Arquimedes (Ar) [36], o uso de uma rede neural LSTM na predição do escoamento de fluidos complexos através de medições de sensores que apresentam ruídos [37], etc.

Dentro do tema de correlação entre fatores, a conexão entre a dinâmica das gotículas e o balanço de energia tem sido extensivamente explorada. Quando uma gotícula se aproxima de uma superfície ou colide com outra gotícula, inicia-se um processo de transferência de energia que envolve componentes cinéticos, superficiais e dissipativos. A análise do balanço de energia aprimora não só a compreensão sobre os princípios fundamentais da dinâmica dos fluidos, mas também facilita o desenvolvimento de modelos analíticos baseados nele. Por exemplo, expressões derivadas da conservação de energia podem ser empregadas para construir equações que predizem o diâmetro máximo de uma gotícula se espalhando em uma superfície sólida. Publicações como [3, 38, 39, 8] exemplificam aplicações de tais princípios.

A exploração das conexões entre o balanço de energia e a dinâmica de gotículas tem uma grande importância prática, visto que diversas das aplicações citadas anteriormente requerem um controle preciso sobre a dinâmica de impacto das gotículas para guiá-las de acordo com o resultado esperado. Dentre trabalhos que exploram tais conexões, [40] investigou os efeitos de baixas e altas velocidades na energia superficial e na dissipação viscosa para gotículas impactando superfícies sólidas. Já a conexão entre a formação de respingos e a dissipação de energia foi abordada por [1], que utilizou resultados experimentais envolvendo uma gota de ferrofluido com formato assimétrico. Em relação aos estudos de colisão de gotículas, [41] explorou o balanço de energia durante a coalescência e a quebra após a colisão, [42] aplicou um esquema numérico com representação Front-Tracking para estudar o balanço de energia usando gotículas de tamanhos diferentes e [43] reavaliou a delimitação do regime de rebote do mapa proposto por [11], aplicando, por meio de experimentos, uma estratégia baseada em energia e colisões binárias de gotículas.

Apesar da grande quantidade de estudos, esses temas são normalmente abordados apenas no contexto de análises estáticas, havendo poucos trabalhos [44, 40] que lidam com as variações transientes das propriedades das gotículas, como o diâmetro durante o espalhamento, e do balanço de energia. A concentração na análise estática dos dados também pode ser observada em investigações mais recentes que usam algoritmos de aprendizado de máquina.

Neste trabalho, propomos modelos de aprendizado de máquina baseados em uma arquitetura de rede neural recorrente chamada de Long Short-Term Memory (LSTM) [45] para a predição não apenas de características estáticas, como números adimensionais, mas também de características transientes, como a variação temporal do balanço de energia. Para testar os modelos, geramos dois conjuntos de dados dos regimes comentados anteriormente: impacto de gotículas de formatos diferentes em superfícies sólidas e colisão entre gotículas. Embora nossos testes tenham empregado exclusivamente conjuntos de dados gerados por simulações numéricas, a versatilidade dessa metodologia não impossibilita a extensão da aplicação para investigações experimentais.

Para executar as simulações numéricas, o *solver* desenvolvido por [34], em linguagem de programação C, foi adaptado de forma a aceitar a entrada de gotículas com formatos diferentes e gravar os dados necessários. Para a implementação da versão minimalista do Front-Tracking, o método de extração de interfaces em imagens, a construção de arquiteturas de redes neurais e a geração das visualizações dos resultados, foi utilizada a linguagem de programação Python

com o auxílio das bibliotecas NumPy (estruturação dos dados), Matplotlib (visualização dos dados), OpenCV (processamento de imagens) e Tensorflow (implementação em alto nível das arquiteturas de redes neurais).

1.1 Objetivos

Neste trabalho, teve-se como objetivo explorar e desenvolver os seguintes temas:

- Representar interfaces através do método Front-Tracking, construindo malhas com classificação de células e tratando as mudanças topológicas;
- Extrair interfaces de gotículas presentes em imagens de experimentos;
- Definir o cálculo do balanço das energias presentes em um sistema (cinética, superficial e de dissipação viscosa);
- Apresentar conceitos de aprendizado de máquina, em específico o funcionamento de uma arquitetura de rede neural recorrente chamada Long Short-Term Memory (LSTM);
- Comentar os resultados da aplicação de redes neurais para predição das energias pelo tempo e/ou dos parâmetros adimensionais em dois conjuntos de dados gerados por simulações numéricas;
- Explorar as possibilidades de correlação entre propriedades físicas e geométricas.

No Capítulo 2, é elaborada uma descrição resumida da modelagem matemática do problema e da adimensionalização dos termos. No Capítulo 3, é definida a implementação de uma versão simplificada do método de representação Front-Tracking, sendo aplicada uma advecção por campo de velocidades para testá-lo. No capítulo 4, é descrito um método de extração de partículas de interfaces presentes em imagens digitais, destacando os problemas relacionados ao método e as possíveis soluções. No capítulo 5, é demonstrado o cálculo das energias de um sistema. No Capítulo 6, são apresentados os regimes selecionados para a geração de dois conjuntos de dados, além da esquematização do processo de extração e processamento dos dados. No Capítulo 7, é explicado o funcionamento geral da arquitetura de rede neural LSTM. No Capítulo 8, são apresentados os resultados obtidos através de duas abordagens que visam aplicabilidade no meio experimental. Por fim, no Capítulo 9, são realizadas as considerações finais sobre o projeto e levantados alguns dos possíveis próximos passos do estudo.

2 Modelagem matemática

A modelagem matemática do escoamento de fluidos e a estratégia de solução numérica adotadas neste projeto foram implementadas e definidas em detalhes no trabalho de [34], sendo adaptadas para gerar os dados necessários para o projeto.

Como o objetivo principal é a extração de alguns dados da simulação numérica e a utilização desses dados para realizar o treinamento e o teste de redes neurais, a modelagem matemática será definida de forma sucinta, citando as equações que regem o problema, as condições de contorno do domínio e trazendo um contexto sobre o que é tratado pelo *solver*.

2.1 Equações e adimensionalização

As equações que governam o comportamento do escoamento de fluidos isotérmicos e incompressíveis são a equação de momento (2.1) e a equação de continuidade (2.2).

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u})\right) = -\nabla p + \nabla \cdot \boldsymbol{\tau}, \tag{2.1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.2}$$

nas quais $\mathbf{u}$ e p são os campos de velocidade e pressão e ρ é a densidade do fluido. Levando em conta apenas fluidos newtonianos, o tensor tensão $\boldsymbol{\tau}$ assume a seguinte forma:

$$\boldsymbol{\tau} = 2\eta\dot{\boldsymbol{\gamma}}, \tag{2.3}$$

em que η é a viscosidade do fluido e $\dot{\boldsymbol{\gamma}} = \frac{1}{2}(\nabla\mathbf{u} + (\nabla\mathbf{u})^T)$ é o tensor de taxa de deformação com $\nabla\mathbf{u}$ sendo o gradiente de velocidade.

Em seguida, as expressões definidas anteriormente foram adimensionalizadas. Para tal, as variáveis que as compõe são escaladas da seguinte forma:

$$\bar{\mathbf{x}} = \frac{\mathbf{x}}{D}, \quad \bar{t} = \frac{tU}{D}, \quad \bar{\mathbf{u}} = \frac{\mathbf{u}}{U}, \quad \bar{p} = \frac{p}{\rho U^2}, \quad \bar{\boldsymbol{\tau}} = \frac{D\boldsymbol{\tau}}{\eta U}, \tag{2.4}$$

em que D e U são, respectivamente, o comprimento e a velocidade característicos. Para simplificar visualmente a adimensionalização das equações governantes (2.1) e (2.2), os termos definidos em (2.4) são utilizados sem as barras superiores, gerando:

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{u}\mathbf{u}) = -\nabla p + \frac{1}{Re}\nabla^2 \mathbf{u}, \tag{2.5}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2.6}$$

em que $Re = \frac{\rho U D}{\eta}$ é o número de Reynolds adimensional.

Além de satisfazer o sistema acima, a solução numérica precisa levar em conta um conjunto de restrições complementares impostas sobre o fluido para garantir uma solução única, chamadas de condições de contorno.

2.2 Condições de contorno

Para os testes executados, tem-se duas condições de contorno de maior importância: a condição para paredes sólidas, para definir o comportamento nas fronteiras do domínio, e a condição de superfície livre, para o contato entre o fluido e o ambiente externo (ar). Em resumo, uma condição *no-slip* é aplicada para contornos de paredes sólidas, enquanto a condição de superfície livre é definida como:

$$\begin{cases} \mathbf{n} \cdot (\mathbf{T} \cdot \mathbf{n}) = \gamma \kappa, \\ \mathbf{m} \cdot (\mathbf{T} \cdot \mathbf{n}) = 0, \end{cases} \tag{2.7}$$

na qual $\mathbf{n}$ e $\mathbf{m}$ são, respectivamente, os vetores normal e tangencial à superfície livre, γ é o coeficiente de tensão superficial, κ é a curvatura da interface e $\mathbf{T} = \boldsymbol{\tau} - p\mathbf{I}$ é o tensor tensão total. As formas adimensionalizadas de 2.7 podem ser escritas como:

$$\begin{cases} \mathbf{n} \cdot (\bar{\mathbf{T}} \cdot \mathbf{n}) = \frac{1}{We}\bar{\kappa}, \\ \mathbf{m} \cdot (\bar{\mathbf{T}} \cdot \mathbf{n}) = 0, \end{cases} \tag{2.8}$$

na qual

$$\bar{\kappa} = D\kappa$$
$$\bar{\mathbf{T}} = 2\bar{\dot{\boldsymbol{\gamma}}} - Re \cdot \bar{p}\mathbf{I}$$

e $We = \frac{\rho U^2 D}{\gamma}$ é o número de Weber. Além disso, um número adimensional adicional é adotado para o regime de colisão, o parâmetro de impacto $B = \frac{2b}{D_1+D_2}$, onde b é a distância relativa de centro a centro entre as gotículas e D_1, D_2 são os diâmetros de cada gotículas, conforme apresentado em [43].

2.3 Solução numérica

Para resolver numericamente as equações governantes do escoamento de fluidos isotérmicos e incompressíveis, foi aplicado o famoso método de projeção [46, 47]. De forma resumida, o método utiliza a decomposição de Helmholtz-Hodge para desacoplar a velocidade e a pressão das equações (2.5) e (2.6). Na decomposição de Helmholtz-Hodge, qualquer campo vetorial definido em uma região Ω com fronteira suave $\partial\Omega$ pode ser decomposto de forma única:

$$\tilde{\mathbf{u}} = \mathbf{u} + \nabla\psi, \tag{2.9}$$

em que $\mathbf{u}$ é um campo vetorial tal que $\nabla \cdot \mathbf{u} = 0$ e ψ é um campo escalar também definido em Ω.

Inicialmente, o método de projeção resolve a equação de momento 2.5 para um campo de velocidade intermediário $\tilde{\mathbf{u}}$. Para desacoplar a velocidade e a pressão, a p desconhecida é aproximada por algum campo escalar conhecido $\tilde{p}$, resultando na seguinte equação:

$$\frac{\partial\tilde{\mathbf{u}}}{\partial t} + \nabla \cdot (\tilde{\mathbf{u}}\tilde{\mathbf{u}}) = -\nabla\tilde{p} + \frac{1}{Re}\nabla^2\tilde{\mathbf{u}}, \tag{2.10}$$

com condições de contorno apropriadas. Após solucionar a equação 2.10 e obter o campo de velocidade intermediário $\tilde{\mathbf{u}}$, ainda precisamos ter certeza de que a equação de continuidade será satisfeita. Utilizando a decomposição de Helmholtz-Hodge e as equações 2.9 e 2.6, podemos escrever a seguinte equação de Poisson para um campo escalar ψ [48]:

$$\nabla^2\psi = \nabla \cdot \tilde{\mathbf{u}}. \tag{2.11}$$

Por fim, ao resolver as equações 2.10 e 2.11, nós obtemos $\tilde{\mathbf{u}}$ e ψ de tal forma que a decomposição 2.9 possa ser usada para obter um campo de velocidade final $\mathbf{u}$ que satisfaça tanto a equação de momento quanto a de continuidade. O processo de discretização é realizado utilizando o método de diferenças finitas para problemas de superfície livre, como discutido em detalhes em [49]. É importante mencionar que empregamos uma formulação axissimétrica para resolver o problema de impacto de gotículas em superfícies, enquanto as simulações de colisão entre gotículas foram realizadas utilizando um código bidimensional.

3 Método Front-Tracking

Nesta seção, é definida e explicada uma versão reduzida do método Front-Tracking, destacando como ele representa uma interface, como é construída a malha a partir das informações geradas e como são tratadas as mudanças topológicas. Para facilitar a compreensão do funcionamento prático deste método, uma implementação simplificada foi realizada na linguagem de programação Python, seguindo as ideias expostas na adaptação feita por [50] dos trabalhos de [51, 52].

3.1 Representação de interfaces

A interface formada pela área de contato entre dois fluidos é representada através de um conjunto discretizado de coordenadas, comumente chamadas de partículas, responsáveis por modelar a sua evolução temporal. Para representar computacionalmente essas partículas e permitir que elas sofram mudanças topológicas durante o escoamento, os principais componentes da representação Front-Tracking são:

Ponto: representa as partículas de uma interface. São armazenados em uma lista duplamente encadeada na qual cada coordenada da interface está interligada à coordenada anterior e à posterior. Para tal, cria-se uma classe Ponto que guarda os valores de x e y e aponta para o ponto anterior (ant) e o próximo (prox):

```
1:  class Ponto():
2:    def __init__(self, x, y, ant=None, prox=None):
3:          self.x = x
4:          self.y = y
5:          self.ant = ant
6:          self.prox = prox
```

A Figura 2 exemplifica a representação de uma interface por meio de um conjunto de dez pontos. Quanto maior for o número de pontos utilizados, mais precisa será a representação. Porém, como o processo de tratamento de mudanças topológicas pode ser bastante custoso para um grande conjunto de pontos, recomenda-se utilizar apenas o mínimo necessário.

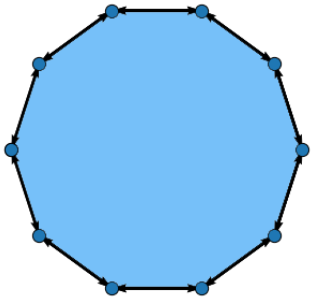


Figura 2 – Representação da interface por um conjunto de pontos interligados.

Curva: são formadas por um conjunto de pontos ordenados em um determinado sentido. O sentido pode ser horário ou anti-horário, sendo necessário manter um padrão para todas as curvas. A classe Curva armazena o nó inicial, o nó final, a região à sua esquerda e região à sua direita, sendo as regiões definidas de acordo com o sentido adotado.

```
1:  class Curva():
2:      def __init__(self, no_inicial, no_final,
3:                   regiao_esq=None, regiao_dir=None):
4:          self.no_inicial = no_inicial
5:          self.no_final = no_final
6:          self.regiao_esq = regiao_esq
7:          self.regiao_dir = regiao_dir
```

A Figura 3 ilustra um sistema com duas curvas distintas. Em casos que apresentam uma curva fechada, como mostrado no exemplo, é adicionado um novo ponto na mesma posição do nó inicial para fechar o ciclo. Isso é necessário para evitar problemas ao percorrer a lista de pontos das curvas.

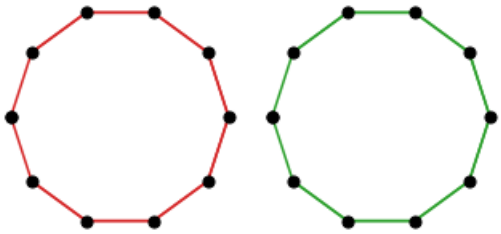


Figura 3 – Interface representada por duas curvas.

Região: são seções delimitadas pelas curvas do sistema. A classe Regiao armazena um número identificador da região, uma variável status necessária para o tratamento das mudanças topológicas, podendo receber os valores “INDEFINIDA”, “FISICA” ou “NAO_FISICA”, e a sua localização, podendo ser “INTERNA” (com um fluido) ou “EXTERNA” (com outro fluido).

```
1:  class Regiao():
2:      def __init__(self, numero, status="INDEFINIDA",
3:                   local="INTERNA"):
4:          self.numero = numero
5:          self.status = status
6:          self.local = local
```

Por fim, na Figura 4, é mostrado um exemplo de uma curva que define duas regiões. A primeira região engloba todo o ambiente externo (ausência do fluido azul), enquanto a segunda engloba a parte interna (com o fluido azul).

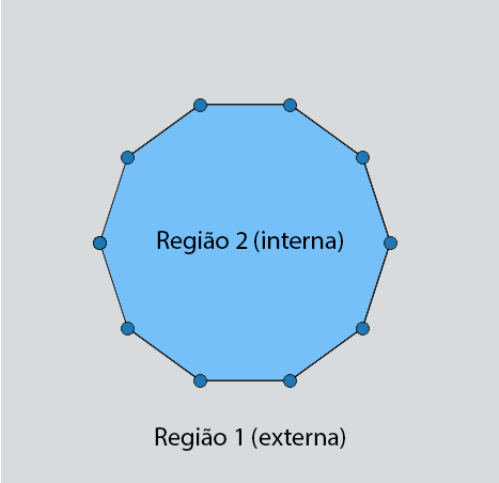


Figura 4 – Divisão de um sistema em duas regiões através de uma curva.

3.2 Construção da malha

Alguns processos, como a aproximação das curvaturas da interface, exigem a definição de uma malha com células classificadas como “Cheia” (totalmente preenchida), “Vazia” (sem o fluido) ou “Interface” (parcialmente preenchida). Para construir uma malha $N_x \times N_y$ e classificar suas células, são seguidas as seguintes regras:

Cheia (F): estão dentro de uma região que contém fluido, podendo incluir um ou mais pontos, e não apresentam células vazias em qualquer um de seus quatro vizinhos imediatos $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ e $(i, j+1)$;

Vazia (E): estão fora de regiões que contenham fluido e não possuem pontos em seu interior;

Interface (S): possuem ao menos um ponto em seu interior e são vizinhas imediatas de ao menos uma célula vazia.

Para realizar estas classificações, são executados os seguintes passos:

Passo 1: inicializar a malha com todas as células vazias (E);

Passo 2: caminhar todos os pontos armazenados e classificar as células em que se encontram como sendo de interface (S). Para encontrar a qual célula um ponto pertence, é utilizada uma função que associa as coordenadas do ponto aos índices da matriz através dos intervalos hx e hy:

```
1:  def encontrar_celula(hx, hy, ponto):
2:      i = int(ponto.y / hy)
3:      j = int(ponto.x / hx)
4:
5:      return i, j
```

na qual $h_x = \frac{x_{max} - x_{min}}{N_x}$ e $h_y = \frac{y_{max} - y_{min}}{N_y}$. Durante esta etapa, algumas células podem ser classificadas erroneamente como sendo de interface. Em um próximo passo, será explicado como esse problema pode ser resolvido.

Passo 3: definidas as células de interface, a próxima etapa envolve classificar todas as células de seu interior como sendo cheias. Para isso, é encontrado um ponto que esteja dentro do fluido e é aplicado o algoritmo *flood fill* (preenchimento por inundação) para preencher recursivamente as células vizinhas da região interna até alcançar as células de interface.

```
1:  def flood_fill(malha, Nx, Ny, i, j):
2:      # Caso base para parar a recursao
3:      if (j <= 0 or j >= Nx or
4:          i <= 0 or i >= Ny or
5:          malha[i][j] in ["S", "F"]):
6:            return
7:
8:      # Classifica a celula como cheia
9:      malha[i][j] = "F"
10:
11:     # Realiza recursao nas celulas adjacentes
12:     flood_fill(malha, Nx, Ny, i, j-1)
13:     flood_fill(malha, Nx, Ny, i-1, j)
14:     flood_fill(malha, Nx, Ny, i, j+1)
15:     flood_fill(malha, Nx, Ny, i+1, j)
16:
17:     return malha
```

Passo 4: percorrer a malha e verificar os vizinhos das células de interface. Caso uma delas não tenha uma célula vazia em sua vizinhança, ela é reclassificada como cheia. Esta etapa é essencial para corrigir as classificações incorretas feitas no passo 2.

Na Figura 5, é mostrada uma malha 200×200, construída pelo método descrito anteriormente, para uma circunferência de raio $r = 0.25$ centrada no domínio, com coordenadas x e y variando de 0 a 2.

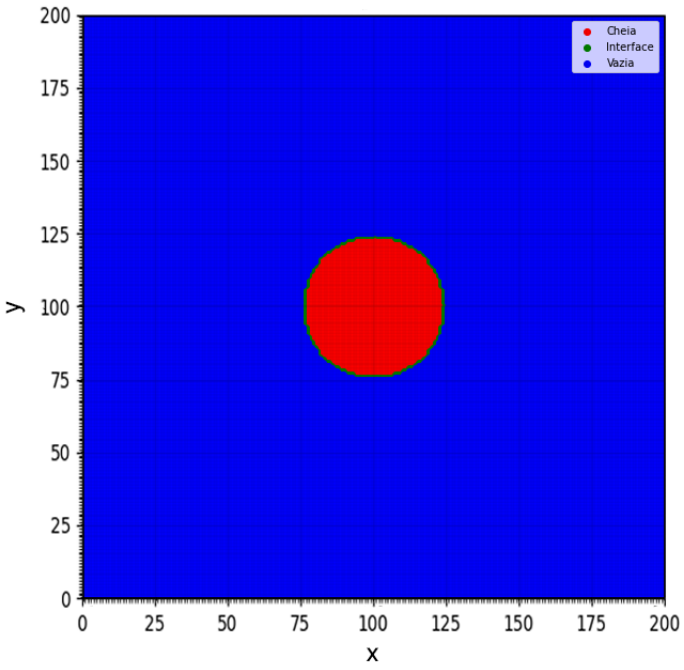


Figura 5 – Malha gerada pelo método aplicado a um exemplo.

3.2.1 Modificação

Dada uma malha já inicializada, é possível implementar uma modificação no algoritmo de construção da malha para otimizar a sua atualização, removendo a necessidade de gerá-la do zero a cada passo de tempo.

Primeiramente, buscam-se os índices das células que cada ponto da interface está ocupando no passo de tempo atual. Em seguida, são executadas as seguintes verificações para cada índice:

- Se a célula estava vazia no passo anterior e agora possui um ponto, ela é transformada em interface. Ao modificar uma célula (i, j), é preciso tratar também sua vizinhança 3×3. Para cada vizinho, verifica-se se nenhuma das quatro células adjacentes é uma célula vazia. Caso não sejam, esse vizinho é transformado em uma célula cheia.

```
1:  if (malha[i][j] == "E"):
2:      malha[i][j] = "S"
3:      for i in [i-1, i, i+1]:
```

```
4:          for j in [j-1, j, j+1]:
5:              if (malha[i-1][j] in ["S", "F"] and
6:                  malha[i][j-1] in ["S", "F"] and
7:                  malha[i+1][j] in ["S", "F"] and
8:                  malha[i][j+1] in ["S", "F"]):
9:                      malha[i][j] = "F"
```

- Se a célula era de interface e agora possui um ponto, verifica-se se ela deverá continuar como interface ou se deverá se tornar uma célula cheia. Se nenhum vizinho for uma célula vazia, a célula deve se tornar cheia. Para verificar cada vizinho, é necessário verificar não só a classificação anteriormente armazenada na malha, mas também a gerada no passo atual.

```
1:  if ((malha[i][j] == "S") and
2:      (malha[i-1][j] in ["S", "F"] or (i-1, j) in indices) and
3:      (malha[i][j-1] in ["S", "F"] or (i, j-1) in indices) and
4:      (malha[i+1][j] in ["S", "F"] or (i+1, j) in indices) and
5:      (malha[i][j+1] in ["S", "F"] or (i, j+1) in indices)):
6:          malha[i][j] = "F"
```

- Se a célula era cheia e agora possui um ponto, verifica-se se ela vai continuar cheia ou se vai se tornar de interface. Primeiramente, checa-se se a vizinhança 3×3 está correta, analisando se algum vizinho está classificado como interface sem possuir qualquer ponto no passo atual. Se houver algum vizinho com essa característica, transforma-o em uma célula vazia.

  Em seguida, faz uma verificação semelhante à mostrada anteriormente para testar se algum vizinho adjacente é uma célula vazia. Se for, transforma a célula em uma célula de interface.

```
1:  if (malha[i][j] == "F"):
2:      for iv in [i-1, i, i+1]:
3:          for jv in [j-1, j, j+1]:
4:              if (malha[iv][jv] == "S" and (iv, jv) not in indices):
5:                  malha[iv][jv] = "E"
6:      if ((malha[i-1][j] not in ["S", "F"] and
7:          (i-1, j) not in indices) or
8:          (malha[i][j-1] not in ["S", "F"] and
9:          (i, j-1) not in indices) or
10:         (malha[i+1][j] not in ["S", "F"] and
11:         (i+1, j) not in indices) or
12:         (malha[i][j+1] not in ["S", "F"] and
13:         (i, j+1) not in indices)):
14:             malha[i][j] = "S"
```

Como esta técnica percorre apenas as células que serão modificadas no passo de tempo atual, quanto maior for o tamanho da malha ou a complexidade da interface, mais significativo

será o tempo de execução economizado. É importante notar, porém, que grande parte do custo computacional do Front-Tracking encontra-se no tratamento das mudanças topológicas que, como será explorado posteriormente, conta com algumas etapas bastante custosas.

3.3 Tratamento das mudanças topológicas

Dado que o método Front-Tracking não trata as mudanças topológicas em sua formulação, é necessário desenvolver um método auxiliar para essa finalidade. No entanto, antes de definir tal método, é fundamental compreender melhor o que caracteriza uma mudança topológica.

Ao movimentar as curvas da interface durante a advecção do fluido, elas podem acabar se cruzando, fazendo com que sobreponham-se. Estes cruzamentos, também chamados de bifurcações, geram as chamadas mudanças topológicas da interface.

As bifurcações podem ocorrer naturalmente durante o escoamento ou artificialmente graças a erros de arredondamento ou instabilidades do método numérico utilizado. Independente da forma como surgem, as bifurcações embaraçam a interface e geram comportamentos anômalos na simulação. Para tratar o embaraçamento, são utilizados os métodos de tratamento de mudanças topológicas.

Ao detectar um embaraçamento na interface, existem duas possíveis situações:

União: duas curvas se encontram e se unem. Como resultado, é formada uma curva maior contendo as regiões das curvas que a produziram. Na Figura 6 é apresentado um exemplo de duas curvas em movimento que acabam colidindo, formando um embaraçamento e se unindo.

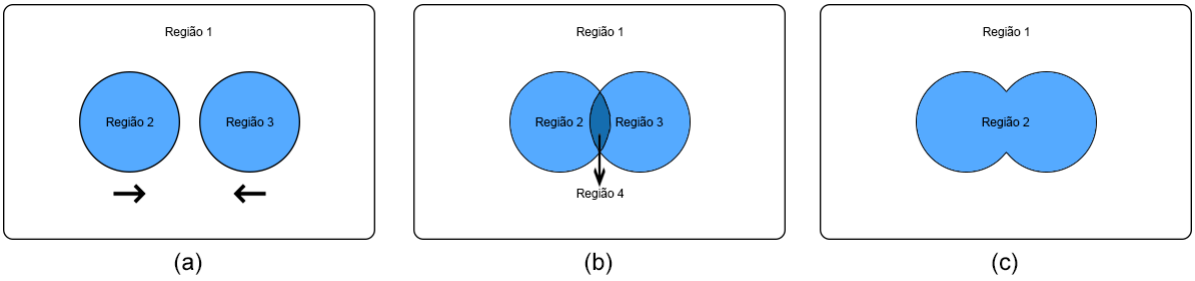


Figura 6 – Exemplo de mudança topológica de união.

Separação: a região de uma curva torna-se tão fina que acaba se quebrando, separando-a em curvas e regiões menores. A Figura 7 mostra um exemplo no qual é forçado um

embaraçamento em uma região fina da curva para que haja a separação.

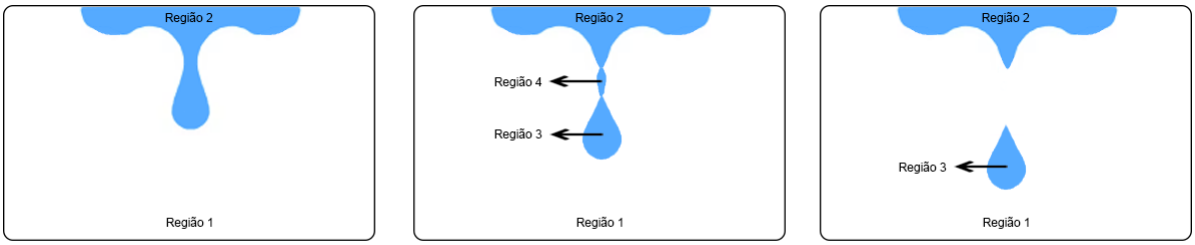


Figura 7 – Exemplo de mudança topológica de separação.

Estabelecida a definição do que é uma mudança topológica, o próximo passo é implementar um algoritmo capaz de tratá-la. Para facilitar a descrição das operações executadas pelo método, ele será dividido em diferentes etapas.

Para que o método funcione, porém, existem algumas condições que devem ser respeitadas:

1. Entre um passo de tempo t_i e um passo t_{i+1}, uma curva pode cruzar apenas **uma** outra curva. Caso uma curva cruze duas ou mais curvas em um único passo de tempo, o método não será capaz de aplicar o desembaraçamento;

2. As regiões não-físicas apresentam ao menos um ponto de embaraçamento em seu contorno. Regiões não-físicas são regiões que não condizem com o comportamento físico esperado do escoamento, sendo necessário excluí-las;

3. Para que haja embaraçamento, as curvas devem necessariamente se interceptar. Sendo assim, as curvas não são consideradas embaraçadas caso apenas tangenciem umas às outras;

4. Os pontos inicial e final de uma curva não podem cruzar uma outra curva durante a advecção. Apenas os demais pontos de uma curva podem estar envolvidos em um embaraçamento, caso contrário o método não funcionará adequadamente. Para mitigar esse problema, é necessário escolher os pontos inicial e final distantes das porções em que poderão ocorrer as mudanças.

Caso todas as condições sejam cumpridas, o método pode ser executado para detectar e tratar embaraçamentos de interfaces. São realizadas as seguintes etapas:

Etapa 1: verifica se há embaraçamento na interface e, se houver, localiza o nó onde ocorre a intersecção. Como cada curva é dada por segmentos de reta formados por pares de pontos, é preciso verificar para cada segmento se há algum outro segmento que o intersecta em algum ponto. Dados os segmentos s_1, formado pelos pontos $P_{11} = (x_{11}, y_{11})$ e $P_{12} = (x_{12}, y_{12})$, e s_2, formado pelos pontos $P_{21} = (x_{21}, y_{21})$ e $P_{22} = (x_{22}, y_{22})$, suas equações paramétricas são definidas da seguinte forma:

$$s_1 : (x, y) = (x_{11}, y_{11}) + \lambda_1(x_{12} - x_{11}, y_{12} - y_{11}), \quad \text{com } \lambda_1 \in [0, 1] \tag{3.1}$$

$$s_2 : (x, y) = (x_{21}, y_{21}) + \lambda_2(x_{22} - x_{21}, y_{22} - y_{21}), \quad \text{com } \lambda_1 \in [0, 1] \tag{3.2}$$

Para verificar se há um ponto de intersecção entre os segmentos, as expressões 3.1 e 3.2 são igualadas, fazendo com que:

$$(x_{11}, y_{11}) + \lambda_1(x_{12} - x_{11}, y_{12} - y_{11}) = (x_{21}, y_{21}) + \lambda_2(x_{22} - x_{21}, y_{22} - y_{21}) \tag{3.3}$$

É possível reorganizar essa igualdade para formar um sistema linear tal que:

$$\begin{bmatrix} x_{12} - x_{11} & -x_{22} + x_{21} \\ y_{12} - y_{11} & -y_{22} + y_{21} \end{bmatrix} \begin{bmatrix} \lambda_1 \\ \lambda_2 \end{bmatrix} = \begin{bmatrix} x_{21} - x_{11} \\ y_{21} - y_{11} \end{bmatrix} \tag{3.4}$$

Para verificar se há uma intersecção, o sistema linear 3.4 é resolvido, encontrando λ_1 e λ_2. Caso ambos estejam no intervalo $[0, 1]$, os segmentos apresentam uma intersecção. Para determinar em qual ponto ela ocorre, é aplicado um dos valores de λ em sua respectiva equação paramétrica.

O ponto de intersecção é chamado de nó de embaraçamento. A Figura 8 ilustra um exemplo de intersecção entre dois segmentos de reta de curvas diferentes, havendo a formação de um nó de embaraçamento no centro.

Figura 8 – Intersecção entre dois segmentos de reta.

Para coletar os nós de embaraçamento, é aplicada uma função que caminha cada uma das curvas e verifica quais delas intersectam alguma outra:

```
emb = []
for i in range(len(curvas)):
    c1 = curvas[i]
    s1 = c1.no_inicial
    while (s1 != c1.no_final):
        x11, y11 = s1.x, s1.y
        x12, y12 = s1.prox.x, s1.prox.y
        for j in range(len(curvas)):
            c2 = curvas[j]
            if (j >= i):
                s2 = c2.no_inicial
                while (s2 != c2.no_final):
                    x21, y21 = s2.x, s2.y
                    x22, y22 = s2.prox.x, s2.prox.y
                    A = np.array([[x12-x11, -(x22-x21)],
                                  [y12-y11, -(y22-y21)]])
                    b = np.array([[x21-x11], [y21-y11]])
                    try:
                        lmbd = np.linalg.solve(A, b)
                        if (lmbd[0] > 0 and lmbd[0] < 1 and
                            lmbd[1] > 0 and lmbd[1] < 1):
                            emb.append([s1, s2,
                                        x11+lmbd[0]*(x12-x11),
                                        y11+lmbd[0]*(y12-y11)])
                    except Exception:
                        pass
                    s2 = s2.prox
        s1 = s1.prox
```

Esta é a única etapa obrigatoriamente executada a cada passo de tempo. Caso seja encontrado um embaraçamento, as demais etapas do método são executadas. Caso contrário, a advecção do fluido continua para o próximo passo de tempo.

Etapa 2: cada curva presente no embaraçamento é dividida em duas novas curvas a partir do nó de embaraçamento. Para dividir as curvas, são realizados os seguintes passos:

1. O nó de embaraçamento é adicionado na interface;
2. Para cada curva que participa do embaraçamento, são extraídos os nós inicial e final;
3. Do nó inicial ao nó de embaraçamento, é gerada uma curva;
4. Do nó de embaraçamento ao nó final, é gerada outra curva.

Para exemplificar o processo, é exibida na Figura 9 a definição das novas curvas para o exemplo mostrado anteriormente.

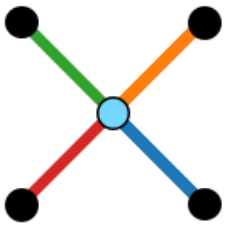


Figura 9 – Divisão das curvas anteriores em novas curvas.

A divisão de cada uma das curvas é feita seguindo os passos descritos no seguinte conjunto de instruções:

```
nos_embaracamento = []
for i in range(len(emb)):
    # Coleta a localizacao dos segmentos embaracados
    s11, s21 = emb[i][0], emb[i][1]
    s12, s22 = s11.prox, s21.prox
    x, y = emb[i][2][0], emb[i][3][0]

    # Cria os novos pontos
    p1 = Ponto(x, y, ant=s11)
    s11.prox = p1
    pontos.append(p1)
    p2 = Ponto(x, y, prox=s12)
    s12.ant = p2
    pontos.append(p2)
    p3 = Ponto(x, y, ant=s21)
    s21.prox = p3
    pontos.append(p3)
    p4 = Ponto(x, y, prox=s22)
    s22.ant = p4
    pontos.append(p4)

    # Procura quais curvas participam do embaracamento
    c1, c2 = None, None
    for curva in curvas:
        ponto = curva.no_inicial
        while ((ponto != None) and (c1 == None or c2 == None)):
            if (ponto == s11 or ponto == s12):
                c1 = curva
            elif (ponto == s21 or ponto == s22):
                c2 = curva
            ponto = ponto.prox

    # Cria novas curvas e atualiza as antigas
    c3 = Curva(p2, c1.no_final)
    curvas.append(c3)
    c4 = Curva(p4, c2.no_final)
    curvas.append(c4)
    c1.no_final = p1
    c2.no_final = p3

    nos_embaracamento.append(p1)
```

Após a divisão das curvas, será necessário definir para cada um dos nós de embaraçamento um vetor contendo todas as quatro curvas que os tocam. Além disso, também é preciso ordenar esses vetores de acordo com a ordem anti-horária de aparição das curvas. Esta ordem é determinada a partir do ângulo formado entre o segmento da curva e o eixo x, calculado usando a função arco tangente e considerando o quadrante em que a curva se encontra:

```
1:  def calc_ang(p1, p2):
2:      if (p2.y - p1.y > 0):
3:          # Quadrante 1
4:          if (p2.x - p1.x > 0):
5:              ang = np.arctan((p2.y-p1.y)/(p2.x-p1.x))
6:          # Quadrante 2
7:          else:
8:              ang = np.pi + np.arctan((p2.y-p1.y)/(p2.x-p1.x))
9:      else:
10:         # Quadrante 3
11:         if (p2.x - p1.x < 0):
12:             ang = np.pi + np.arctan((p2.y-p1.y)/(p2.x-p1.x))
13:         # Quadrante 4
14:         else:
15:             ang = 2*np.pi + np.arctan((p2.y-p1.y)/(p2.x-p1.x))
16:
17:     return ang
```

Seguindo o exemplo, são mostrados na Figura 10 os ângulos formados entre cada curva e o eixo x.

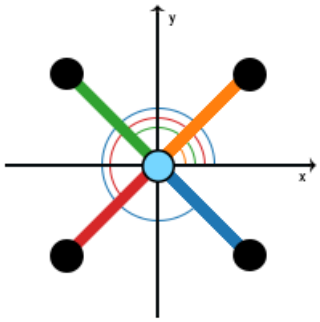


Figura 10 – Ângulos entre as curvas e o eixo x.

Feito isso, os elementos do vetor são ordenados a partir da ordem crescente dos valores dos ângulos calculados:

```
1:  embaracamentos = []
2:  # Encontra os angulos dos segmentos de reta de
3:  # cada curva e os ordena
4:  for n1 in nos_embaracamento:
5:      embaracamento = []
6:      for curva in curvas:
7:          n2 = curva.no_inicial
```

```
            n3 = curva.no_final
            if (n2.x == n1.x and n2.y == n1.y):
                embaracamento.append([curva, "s", calc_ang(n1,
                                      n2.prox), curva.no_final])
            elif (n3.x == n1.x and n3.y == n1.y):
                embaracamento.append([curva, "e", calc_ang(n1,
                                      n3.ant), curva.no_inicial])
        embaracamento.sort(key=lambda x: x[2])
        embaracamentos.append(embaracamento)
```

Etapa 3: levando em conta as curvas antes do embaraçamento mostrado no exemplo da Figura 8, existiam apenas as três regiões ilustradas na Figura 11.

Figura 11 – Curvas e regiões antes do embaraçamento.

Considerando que os pontos das curvas possuem ordenação anti-horária, determina-se que as regiões à esquerda e à direita da curva 1 (vermelha) são as regiões 2 e 1, respectivamente, e as regiões à esquerda e à direita da curva 2 (verde) são as regiões 3 e 1. Para as novas curvas adicionadas na etapa 2, porém, ainda não se tem quaisquer informações sobre suas regiões.

Para preencher quais são as regiões das novas curvas, é utilizado o vetor ordenado gerado na etapa anterior. Ao percorrer cada curva armazenada no vetor, são executadas as seguintes verificações:

- A curva atual e a anterior possuem uma região em comum. Para determiná-la, é analisado o sentido de cada curva em relação ao nó de embaraçamento:
  - Caso a curva atual esteja “entrando” no nó de embaraçamento, ela terá a região esquerda em comum com a curva anterior;
  - Caso esteja “saindo”, ela terá a região direita em comum.
- A curva atual e a próxima também compartilham uma região. Para verificar qual é essa região, os passos explicados anteriormente são seguidos com um sentido invertido:

- Caso a curva atual esteja “entrando” no nó de embaraçamento, ela terá a região direita em comum com a próxima curva;
- Caso esteja “saindo”, ela terá a região esquerda em comum.

A implementação dos passos retratados é apresentada no método “preencher_regioes”:

```
def preencher_regioes(embaracamento):
    for embaracamento in embaracamentos:
        for i in range(4):
            curva_ant = embaracamento[i-1 if i != 0 else 3][0]
            curva = embaracamento[i][0]
            curva_prox = embaracamento[i+1 if i != 3 else 0][0]

            # Determina uma das regioes da curva anterior atraves
            # da sua regiao em comum com a curva atual
            if (embaracamento[i][1] == "e"):
                regiao = curva.regiao_esq
            else:
                regiao = curva.regiao_dir

            # Verifica se a regiao em comum esta
            # na direita ou na esquerda
            if (curva_ant.regiao_dir == None and
                embaracamento[i-1 if i != 0 else 3][1] == "e"):
                    curva_ant.regiao_dir = regiao
            elif (curva_ant.regiao_esq == None):
                curva_ant.regiao_esq = regiao

            # Determina uma das regioes da proxima curva atraves
            # da sua regiao em comum com a curva atual
            if (embaracamento[i][1] == "e"):
                regiao = curva.regiao_dir
            else:
                regiao = curva.regiao_esq

            # Verifica se a regiao em comum esta
            # na direita ou na esquerda
            if (curva_prox.regiao_esq == None and
                embaracamento[i+1 if i != 3 else 0][1] == "e"):
                    curva_prox.regiao_esq = regiao
            elif (curva_prox.regiao_dir == None):
                curva_prox.regiao_dir = regiao
```

Ao concluir o preenchimento, é verificado se alguma curva ainda possui uma região indefinida. Se for o caso, isso indica que uma nova região deverá ser criada e atribuída ao lado da curva que ainda não possui uma. Após a atribuição,o método de preenchimento é executado novamente para que a nova região possa ser inserida em outras curvas que também estejam em contato com ela. O processo é repetido até que todas as curvas estejam com as regiões à esquerda e à direita definidas:

```
while (any(curva.regiao_esq == None or
           curva.regiao_dir == None for curva in curvas)):
```

```
3:      if (any(curva.regiao_esq == None or
4:              curva.regiao_dir == None for curva in curvas)):
5:          preencher_regioes(embaracamentos)
6:      for curva in curvas:
7:          if (curva.regiao_esq == None or
8:              curva.regiao_dir == None):
9:                  regiao = Regiao(len(regioes)+1)
10:                 regioes.append(regiao)
11:                 if (curva.regiao_esq == None):
12:                     curva.regiao_esq = regiao
13:                 elif(curva.regiao_dir == None):
14:                     curva.regiao_dir = regiao
15:                 break
```

Aplicando o processo no exemplo desenvolvido, o novo conjunto de regiões definidas é ilustrado na Figura 12.

Figura 12 – Regiões definidas na etapa 3.

Etapa 4: após definir as regiões das curvas, o próximo passo consiste em classificá-las como sendo físicas ou não-físicas. Como já discutido, uma região é chamada de não-física caso a sua existência não faça sentido no contexto da simulação (como ilustrado na Figura 13). Sendo assim, a classificação das regiões é uma etapa essencial para que seja possível localizar quais regiões permanecerão e quais serão excluídas.

Com base nas quatro condições de funcionamento do método, é possível elaborar um conjunto de observações que auxiliarão no processo de classificação das regiões:

1. Existem sempre duas curvas “entrando” e duas “saindo” em um nó de embaraçamento;
2. Apenas uma região ao redor de um nó de embaraçamento pode ser considerada não-física;
3. A região compreendida entre duas curvas, em que cada uma liga um nó de embaraçamento n_1 a nós n_2 e n_3, com $n_2 \neq n_3$, só será física se n_2 ou n_3 não for um nó de embaraçamento.

Dado que o algoritmo precisa determinar se um nó é ou não um nó de embaraçamento, é implementado um método que compara as coordenadas de ambos:

```
1:  def verif_no(nos_embaracamento, n1):
2:      for n2 in nos_embaracamento:
3:          if (n2.x == n1.x and n2.y == n1.y):
4:              return True
5:      return False
```

Seguindo o conjunto de observações comentadas, são estabelecidos os seguintes passos para a classificação das regiões:

1. Todas as regiões são marcadas como “INDEFINIDA”;
2. Executa os próximos passos enquanto ainda houver alguma região “INDEFINIDA”;
3. Percorre cada nó de embaraçamento;
4. Para cada nó, reúne as regiões ao seu redor;
5. Utilizando a terceira observação, tenta classificar algumas regiões;
6. Utilizando a segunda observação, verifica se já existem três regiões físicas. Se já houver, a última região será obrigatoriamente não-física;
7. Volta para o terceiro passo enquanto ainda houver nós de embaraçamento restantes. Caso não haja, volta para o segundo passo.

Os passos descritos são implementados da seguinte forma:

```
1:  while (any(regiao.status == "INDEFINIDA" for regiao in regioes)):
2:      for i, n1 in enumerate(nos_embaracamento):
3:          # Reune os setores (regioes ao redor
4:          # do no de embaracamento)
5:          setores = []
6:          for j in range(4):
7:              curva = embaracamentos[i][j][0]
8:              if (embaracamentos[i][j][1] == "e"):
9:                  curva.regiao_dir.status = "INDEFINIDA"
10:                 setores.append(curva.regiao_dir)
11:             else:
12:                 curva.regiao_esq.status = "INDEFINIDA"
13:                 setores.append(curva.regiao_esq)
14:
15:         # Classifica os setores
16:         for j in range(4):
17:             n2 = embaracamentos[i][j][3]
18:             n3 = embaracamentos[i][j+1 if j != 3 else 0][3]
19:             if ((n2.x != n3.x and n2.y != n3.y) and
20:                 (any(setor.status == "NAO_FISICA"
21:                  for setor in setores) or
22:                 not verif_no(nos_embaracamento, n2) or
```

```
23:                     not verif_no(nos_embaracamento, n3))):
24:                         setores[j].status = "FISICA"
25:
26:             # Caso ja existam tres setores fisicos,
27:             # o quarto e nao fisico
28:             if (sum(setor.status == "FISICA"
29:                     for setor in setores) == 3):
30:                 for j in range(4):
31:                     if (setores[j].status == "INDEFINIDA"):
32:                         setores[j].status = "NAO_FISICA"
33:                         break
```

Ao final das etapas, todas as regiões estarão classificadas como “FISICA” ou “NAO_FISICA”. Na Figura 13 é mostrada a classificação das regiões do exemplo.

Figura 13 – Classificação das regiões em “FISICA” ou “NAO_FISICA”.

Etapa 5: é aplicada a remoção de todas as regiões não-físicas encontradas e todas as curvas que as delimitam. Para isso, são seguidos os seguintes passos:

1. Verificar a classificação de cada uma das regiões. Para cada região “NAO_FISICA”, executar os próximos passos;
2. Buscar as duas curvas que delimitam a região;
3. Buscar a região física do outro lado da curva 1;
4. Buscar a região física do outro lado da curva 2;
5. Mudar o número dessas regiões para o número da região não-física;
6. Deletar a curva 1 e seus pontos;
7. Deletar a curva 2 e seus pontos.

Para a execução dos passos, é construído o seguinte método:

```
1:  remover = []
2:  for i in range(len(regioes)):
3:      if (regioes[i].status == "NAO_FISICA"):
4:          # Encontra as duas curvas que delimitam a regiao
```

```
5:            c1, c2 = None, None
6:            for curva in curvas:
7:                if (curva.regiao_esq == regioes[i] or
8:                    curva.regiao_dir == regioes[i]):
9:                    if (c1 == None):
10:                       c1 = curva
11:                   elif (c2 == None):
12:                       c2 = curva
13:                       break
14:
15:           # Obtem a regiao do outro lado da curva 1
16:           if (c1.regiao_esq == regioes[i]):
17:               r1 = c1.regiao_dir
18:           else:
19:               r1 = c1.regiao_esq
20:
21:           # Obtem a regiao do outro lado da curva 2
22:           if (c2.regiao_esq == regioes[i]):
23:               r2 = c2.regiao_dir
24:           else:
25:               r2 = c2.regiao_esq
26:
27:           # Muda o numero das regioes
28:           r1.numero = regioes[i].numero
29:           r2.numero = regioes[i].numero
30:
31:           # Deleta a curva 1 e seus pontos
32:           ponto = c1.no_inicial
33:           pontos.remove(ponto)
34:           while (ponto != c1.no_final):
35:               ponto = ponto.prox
36:               pontos.remove(ponto)
37:           curvas.remove(c1)
38:
39:           # Deleta a curva 2 e seus pontos
40:           ponto = c2.no_inicial
41:           pontos.remove(ponto)
42:           while (ponto != c2.no_final):
43:               ponto = ponto.prox
44:               pontos.remove(ponto)
45:           curvas.remove(c2)
46:
47:           remover.append(i)
48:
49:   for i in sorted(remover, reverse=True):
50:       del regioes[i]
```

Aplicando a remoção da região não-física e das curvas ao seu redor, é obtido, para o exemplo, o resultado apresentado na Figura 14.

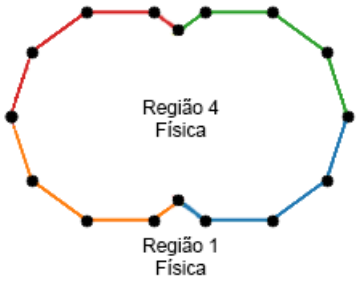


Figura 14 – Remoção da região não-física e das curvas que a formam.

Etapa 6 (opcional): para reduzir o número de curvas, é possível unir as curvas cujos nós finais estão na mesma posição que o nó inicial de alguma outra curva. Para o exemplo desenvolvido, a união de todas as curvas interligadas gera o resultado final apresentado na Figura 15.

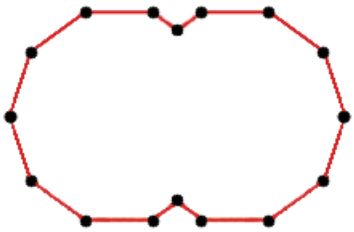


Figura 15 – União das curvas conectadas.

Com isso, o método Front-Tracking foi adaptado para que seja capaz de lidar com as mudanças topológicas, tornando-o válido para o processo de advecção.

Como já apontado, a implementação elaborada é apenas uma versão simplificada do método, levando em conta apenas o processo de advecção através de um campo de velocidades. O objetivo de sua implementação foi desenvolver um maior entendimento sobre a representação de interfaces através do Front-Tracking e sobre uma das possíveis técnicas de tratamento das mudanças topológicas.

4 Extração de interfaces em imagens

Um das limitações observadas na maioria dos *solvers* é a dificuldade de definir interfaces com formatos atípicos. No meio experimental, a criação de gotas esféricas ou elípticas que sejam estritamente simétricas é uma tarefa muito complexa. Porém, ao executar simulações numéricas para reprodução de tais experimentos, muitas vezes opta-se por utilizar desses formatos por apresentarem equações paramétricas que os definem facilmente.

Para que seja possível definir gotas com formatos semelhantes aos observados em experimentos reais, evitando a introdução de ruídos causados por aproximações no formato, foi desenvolvido um método capaz de extrair interfaces presentes em imagens digitais.

Para criar o método, foi escolhida a linguagem de programação Python devido ao seu alto nível e à variedade de funções de processamento de imagens disponíveis na biblioteca OpenCV. As seguintes etapas são realizadas para extrair as partículas de uma imagem:

Etapa 1: abrir uma imagem contendo a interface, preferencialmente delineada com uma cor mais escura que as demais para facilitar a detecção de contornos. Na Figura 16, é apresentado o formato da interface de exemplo que será extraída pelo método.

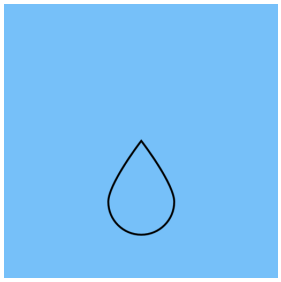


Figura 16 – Imagem original.

Etapa 2: converter a imagem para escala de cinza ou diretamente para preto e branco (recomendado apenas para casos que apresentam interfaces com menos detalhes e com pouco ruído). Para o exemplo, esta etapa produz o resultado ilustrado na Figura 17.

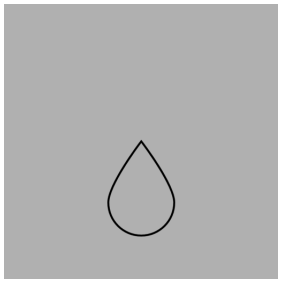


Figura 17 – Conversão da imagem original para escala de cinza.

Etapa 3: para imagens convertidas em escala de cinza, aplicar um limiar. O processo envolve converter os pixels da imagem para preto ou branco a partir de um valor de limiar L. Os pixels com valores inferiores ao limiar são transformados em preto (0), enquanto aqueles com valores iguais ou superiores ao limiar são transformados em branco (255). No exemplo, foi escolhido um limiar $L = 110$, gerando o resultado mostrado na Figura 18. Para automatizar o processo de escolha do limiar, a limiarização de Otsu, um método de segmentação de imagens, é uma possível estratégia.

Figura 18 – Aplicação do limiar na imagem em escala de cinza.

Etapa 4: extrair os contornos da imagem que representam interfaces. Como no exemplo há apenas uma interface contínua, foi encontrado apenas um contorno, conforme demonstrado na Figura 19.

Figura 19 – Contorno da interface detectado.

Etapa 5: aproximar o formato do contorno coletando um conjunto de pontos discretizados (i, j). Como os pontos coletados pela função são na verdade os índices dos pixels no

contorno da imagem, é necessário ajustá-los aos intervalos de interesse $x \in [x_{min}, x_{max}]$ e $y \in [y_{min}, y_{max}]$. Isso é realizado através das seguintes equações:

$$x = \frac{j(x_{min} - x_{max})}{m} + x_{max}$$
$$y = \frac{i(y_{min} - y_{max})}{n} + y_{max}$$

onde $n \times m$ representa as dimensões da imagem de entrada e (x, y) são as novas coordenadas para os pontos extraídos da interface. Como a lista de pixels retornada está ordenada, esse método facilita a criação das partículas para a representação pelo Front-Tracking. Dados os intervalos $x, y \in [0, \pi]$ aplicados no exemplo, são gerados os pontos ilustrados na Figura 20.

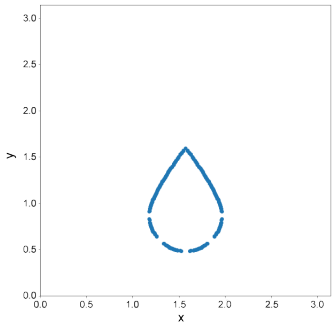


Figura 20 – Conjunto de pontos discretizados no contorno.

Etapa 6: caso a função retorne um número insuficiente de pontos ou gere pontos com um grande espaçamento (como observado no exemplo anterior), é possível aplicar uma interpolação linear para criar novos pontos intermediários.

Para tal, calcula-se a distância euclidiana entre cada par de pontos e, caso a distância seja maior que uma tolerância máxima escolhida, são gerados novos pontos intermediários. Ao aplicar a interpolação no exemplo com uma tolerância de 0.05, é gerado o novo conjunto de pontos mostrado na Figura 21.

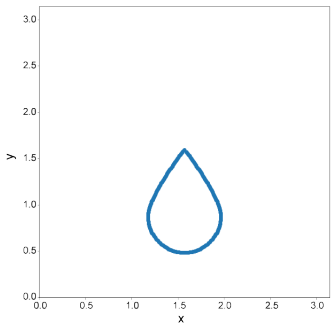


Figura 21 – Conjunto de pontos ao aplicar uma interpolação.

4.1 Teste do funcionamento da extração

Nos passos anteriores, foi demonstrada a extração de pontos através de um formato comumente observado em gotículas com um filamento alongado devido à ação da gravidade. Para verificar se a implementação da extração do contorno também funcionaria com interfaces mais complexas, foi selecionada uma interface formada por "hélices", conforme apresentada em [53]. Tal interface é retratada na Figura 22.

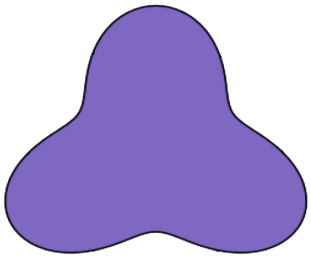


Figura 22 – Exemplo de um contorno mais complexo.

Para testar os pontos extraídos, utilizou-se a implementação em Python do método Front-Tracking que leva em conta apenas a advecção das partículas através de um campo de velocidades, desconsiderando a modelagem matemática do escoamento. Para o campo de velocidades, adotou-se o campo oscilatório descrito em [54], no qual:

$$
\begin{aligned}
u(x, y, t) &= 2 \cdot y \cdot cos\left(\frac{3 \cdot \pi \cdot t}{2}\right) \\
v(x, y, t) &= 2 \cdot x \cdot cos\left(\frac{3 \cdot \pi \cdot t}{2}\right)
\end{aligned}
\tag{4.1}
$$

O comportamento esperado para esse campo é que a interface seja esticada diagonalmente para a direita, volte para o centro e se estique diagonalmente para a esquerda, repetindo a oscilação com o tempo. Para a modelagem, adotou-se um domínio $x_d, y_d \in [-2, 2]$, com a gota contida aproximadamente em $x_g, y_g \in [-0.75, 0.75]$. A Figura 23 mostra a interface extraída da imagem.

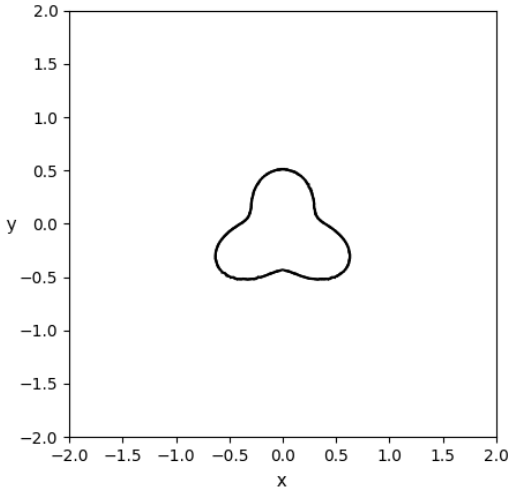


Figura 23 – Gota centrada no domínio $x_d, y_d \in [-2, 2]$.

Cada oscilação leva cerca de $1.33s$ para ser completada. O comportamento em cada etapa da oscilação pode ser observado na Figura 24.

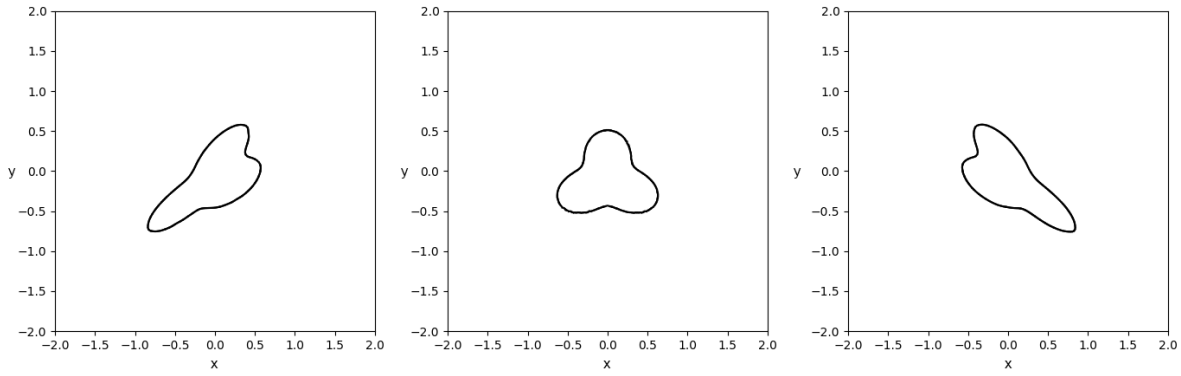


Figura 24 – Comportamento da oscilação.

Com isso, verifica-se que o método também é capaz de extrair pontos em formas mais complexas.

4.2 Limitações e algumas possíveis soluções

Como o método envolve a extração de pontos através de pixels dos contornos de uma imagem, a principal restrição observada é que a extração é limitada pela resolução da imagem. Quanto mais pixels uma imagem tiver, mais precisa será a interface extraída.

Além disso, como os pontos são extraídos do canto inferior esquerdo de cada pixel, é gerado um comportamento de serrilhamento como ilustrado na Figura 25.

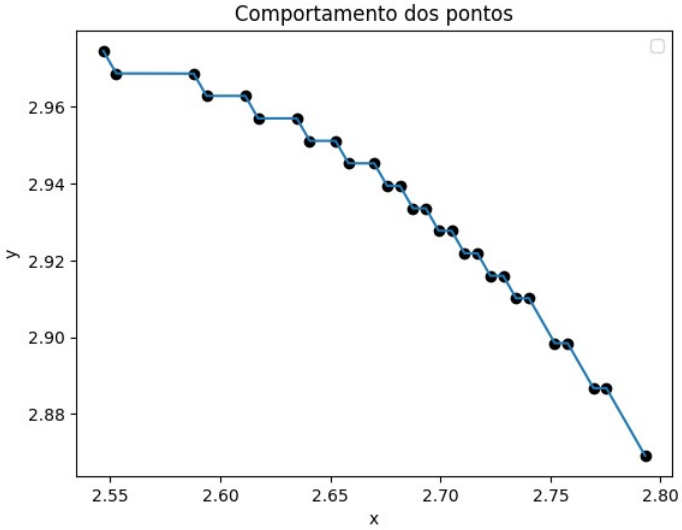


Figura 25 – Serrilhamento gerado ao extrair pontos dos pixels.

Esta perturbação na inicialização dos pontos pode gerar um comportamento atípico nos passos iniciais da simulação. Para mitigar o problema, foram explorados dois possíveis ajustes da estratégia.

O primeiro e mais simples ajuste consiste em agrupar pares de pontos e calcular o ponto médio de cada par tal que:

$$\begin{aligned} \bar{x_i} &= \frac{x_i + x_{i+1}}{2} \\ \bar{y_i} &= \frac{y_i + y_{i+1}}{2} \end{aligned} \tag{4.2}$$

para $i \in [0, n-1]$, sendo n o número de pontos extraídos pelo método. Desta forma, é feita uma suavização nos degraus observados anteriormente, gerando o comportamento exibido na Figura 26.

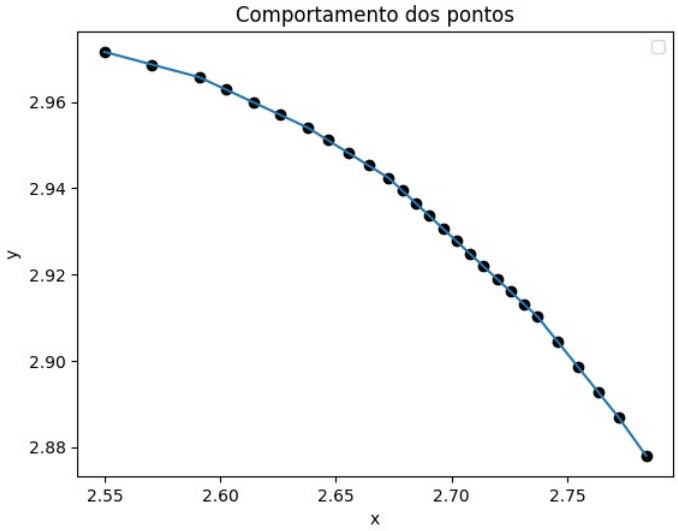


Figura 26 – Suavização do serrilhamento usando os pontos médios.

Embora seja eficiente para eliminar o serrilhamento, essa abordagem pode acabar removendo também as pontas de formatos com cantos de baixa angulação, como é o caso da primeira gota analisada (Figura 16). Além disso, a técnica não considera a minimização da perda de massa do fluido ao interpolar os pontos, o que acaba introduzindo um erro numérico significante já no primeiro passo de tempo.

A segunda opção envolve uma técnica aplicada diretamente no *solver* utilizado neste trabalho. Veremos a seguir que, apesar da técnica também aplicar uma suavização para eliminar os serrilhamentos, ela apresenta uma estratégia que reduz a perda de informações da interface original e, consequentemente, minimiza a perda de massa.

4.3 Método de suavização com conservação de massa - TSUR

O *Trapezoidal Sub-grid Undulations Removal* (TSUR), desenvolvido por [55], é um método de suavização com conservação de massa utilizado na redução de serrilhamentos e ondulações abruptas em curvas.

Dadas quatro partículas P_i, P_{i+1}, P_{i+2} e P_{i+3} (Figura 27), a técnica busca modificar os pontos P_{i+1} e P_{i+2}, gerando $\bar{P}_{i+1}$ e $\bar{P}_{i+2}$, de modo que formem um trapézio cuja área A_t seja igual

à do quadrilátero inicial A_q.

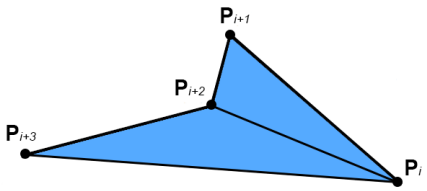


Figura 27 – Exemplo de um grupo de quatro partículas antes de aplicar o TSUR.

O trapézio gerado pelo deslocamento dos dois pontos centrais deve satisfazer as condições $L_1 = L_2 = L_3 = L$ e $h_1 = h_2 = h$ (Figura 28b). Para isso, primeiramente é necessário calcular as áreas A_q e At.

Como o quadrilátero pode ter formas diferentes, a área A_q é obtida através da fórmula de Gauss, dividindo o cálculo da área total em áreas dos triângulos $P_iP_{i+1}P_{i+2}$ e $P_iP_{i+2}P_{i+4}$ que compõe o quadrilátero total:

$$A_q = \frac{1}{2}\left|(x_iy_{i+1} - x_{i+1}y_i) + (x_{i+1}y_{i+2} - x_{i+2}y_{i+1}) + (x_{i+2}y_{i+3} - x_{i+3}y_{i+2}) + (x_{i+3}y_i - x_iy_{i+3})\right| \tag{4.3}$$

Para o cálculo da área do trapézio, tem-se dois triângulos de base L e altura h e um retângulo largura L e altura h:

$$A_t = \frac{Lh}{2} + Lh + \frac{Lh}{2} = 2Lh \tag{4.4}$$

Como $Aq = At = 2Lh$, tem-se que $h = \frac{A_q}{2L}$. Para deslocar os pontos, calcula-se:

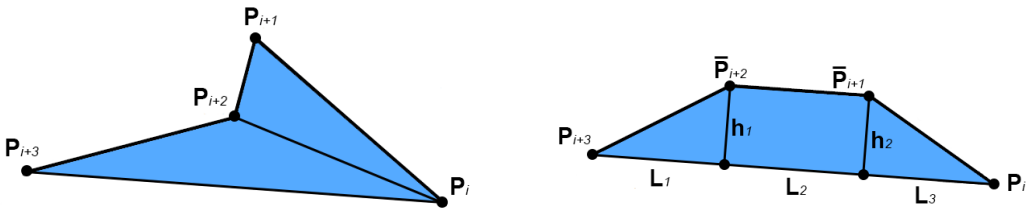
$$\tau = (\tau_x, \tau_y)^t = \frac{P_{i+3} - P_i}{\|P_{i+3} - P_i\|_2} \tag{4.5}$$

sendo τ um vetor unitário tangente a $P_{i+3} - P_i$ e $n = (\tau_y, \tau_x)^t$ um vetor unitário normal que aponta para fora da curva. Através desses vetores, as novas posições dos pontos centrais são definidas pela expressão:

$$(\bar{x}_{i+1}, \bar{y}_{i+1}) = (x_i, y_i) + L\tau + hn \tag{4.6}$$

$$(\bar{x}_{i+2}, \bar{y}_{i+2}) = (x_i, y_i) + 2L\tau + hn \tag{4.7}$$

Formando assim um trapézio que respeita as condições definidas anteriormente. Na Figura 28 é apresentada a comparação da posição dos pontos antes e depois de aplicar o método no exemplo da Figura 27.



(a) Antes. (b) Depois.

Figura 28 – Comparação da interface antes e depois da aplicação do método TSUR.

Além da utilidade do TSUR na remoção do serrilhamento inicial da interface, a técnica também ajuda a reduzir ruídos numéricos que possam aparecer durante a simulação, gerando uma aproximação fisicamente mais acurada.

Definido o método de extração e tratamento das interfaces iniciais, o próximo capítulo será responsável por apresentar o cálculo das energias, o principal dado de interesse das predições.

5 Cálculo do balanço de energias

Para a geração dos conjuntos de dados que foram utilizados no treinamento e no teste das redes neurais, a influência gravitacional foi desconsiderada. Sendo assim, o balanço de energias é dado apenas pela energia cinética, energia superficial e dissipação viscosa. A energia cinética E_k é definida como:

$$E_k = \int_V \frac{1}{2}\rho \, \|\mathbf{u}\|^2 \, dV, \tag{5.1}$$

em que ρ é a densidade. Para calcular a energia cinética em todo o domínio, somamos as contribuições de cada célula da malha preenchida com fluido, excluindo as células de interface. A exclusão se deve ao fato de que o cálculo do volume das células parcialmente preenchidas adicionaria um custo computacional considerável à simulação. Quanto à energia superficial E_s, tem-se:

$$E_s = \int_S \gamma dS, \tag{5.2}$$

sendo γ a tensão superficial. Por fim, a dissipação viscosa E_d é calculada através de:

$$E_d = \int_t \int_V 2\,\eta \, \left\|\dot{\boldsymbol{\gamma}}\right\|^2 dV \, dt, \tag{5.3}$$

em que η é a viscosidade e $\int_V 2\,\eta \, \left\|\dot{\boldsymbol{\gamma}}\right\|^2 dV$ é a chamada de taxa de dissipação viscosa, sendo $\dot{\boldsymbol{\gamma}} = \frac{1}{2}[\nabla\mathbf{u} + \nabla\mathbf{u}^T]$. A dissipação viscosa é dada pelo acúmulo de dissipação com o tempo, sendo calculada a cada passo através do somatório da dissipação em cada uma das células de fluido, excluindo, novamente, as de interface.

A lei da conservação da energia define que o somatório total das energias (E_t) no sistema deve manter-se constante pelo tempo de tal forma que:

$$E_t(t) = E_k(t) + E_s(t) + E_d(t) = E_t(0), \tag{5.4}$$

Para nosso trabalho, porém, os testes constataram haver uma diminuição na energia total com o tempo. Supõe-se que algumas das possíveis causas para esta perda sejam ruídos numéricos e/ou a escolha de não calcular a energia cinética e a dissipação viscosa nas células de interface.

Para avaliar se a redução na energia total ao longo do tempo é relevante, utilizamos um exemplo que será apresentado na Seção 8. Neste exemplo, duas gotas colidem, coalescem e oscilam ao longo de 50 segundos. Para visualizar a evolução das energias em relação ao primeiro passo de tempo, foi utilizado como base o gráfico apresentado em [44], gerando a Figura 29.

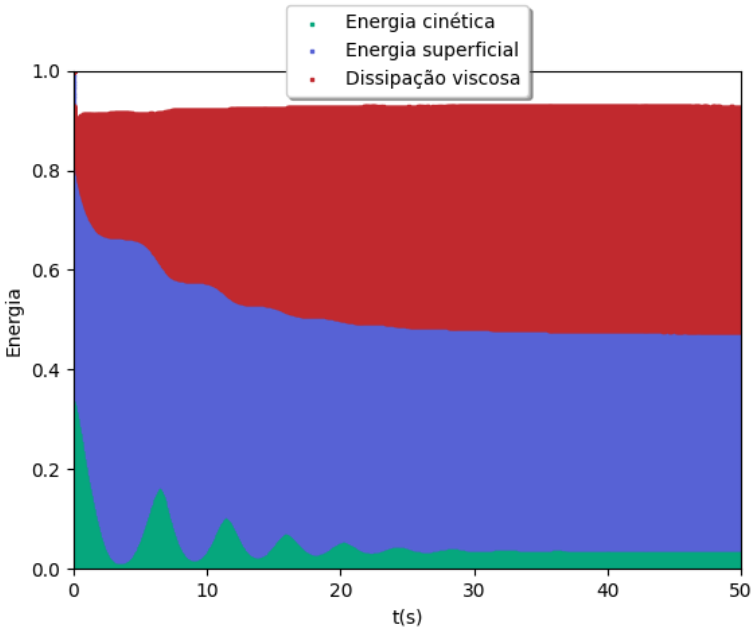


Figura 29 – Balanço de energia com o tempo.

Embora parte da energia superficial seja convertida em energia cinética durante os espalhamentos, parte da cinética seja convertida em superficial durante as retrações e outras partes sejam dissipadas, percebe-se que uma fração da energia total ainda é perdida ao longo do tempo. Em média, a redução equivale a pouco menos de 10% da energia inicial. Dada a natureza numérica do método, a perda é esperada e é considerada insignificante para os resultados dos testes.

Como o *solver* utilizado trabalha com parâmetros adimensionais (vide Seção 2), foram realizadas modificações nos cálculos das energias de forma a torná-las também adimensionalizadas. Para isso, foram utilizadas algumas das definições de termos adimensionais apresentados na Seção 2.1.

Para a energia cinética, define-se a seguinte modificação na equação 5.1 para separar um termo adimensional $\bar{E}_k$:

$$
\begin{aligned}
E_k &= \int_V \frac{1}{2}\rho \left\|\mathbf{u}\right\|^2 dV, \\
E_k &= \int_{\bar{V}} \frac{1}{2}\rho \left\|U\bar{\mathbf{u}}\right\|^2 d(D^3\bar{V}), \\
E_k &= \rho U^2 D^3 \int_{\bar{V}} \frac{1}{2} \left\|\bar{\mathbf{u}}\right\|^2 d\bar{V}, \\
E_k &= \frac{\eta\sigma D}{\rho U} ReWe \int_{\bar{V}} \frac{1}{2} \left\|\bar{\mathbf{u}}\right\|^2 d\bar{V}, \\
E_k &= \frac{\eta\sigma D}{\rho U} \bar{E}_k,
\end{aligned}
\tag{5.5}
$$

gerando $\bar{E}_k = ReWe \int_{\bar{V}} \frac{1}{2} \left\|\bar{\mathbf{u}}\right\|^2 d\bar{V}$. Para decompor a energia superficial da equação 5.2 e gerar $\bar{E}_s$ é feito o seguinte passo a passo:

$$
\begin{aligned}
E_s &= \int_S \sigma dS, \\
E_s &= \int_{\bar{S}} \sigma d(D^2\bar{S}), \\
E_s &= \sigma D^2 \int_{\bar{S}} 1 \, d\bar{S}, \\
E_s &= \frac{\eta\sigma D}{\rho U} Re \int_{\bar{S}} 1 \, d\bar{S}, \\
E_s &= \frac{\eta\sigma D}{\rho U} \bar{E}_s,
\end{aligned}
\tag{5.6}
$$

em que $\bar{E}_s = Re \int_{\bar{S}} 1 \, d\bar{S}$. Por fim, a adimensionalização da dissipação viscosa definida na equação 5.3 gera $\bar{E}_d$ através de:

$$
\begin{aligned}
E_d &= \int_t \int_V 2\,\eta \, \left\|\dot{\boldsymbol{\gamma}}\right\|^2 dV \, dt, \\
E_d &= \int_{\bar{t}} \int_{\bar{V}} 2\,\eta \, \left\|\frac{U}{D}\bar{\dot{\boldsymbol{\gamma}}}\right\|^2 d(D^3\bar{V}) \, d(\frac{D}{U}\bar{t}), \\
E_d &= \int_{\bar{t}} \int_{\bar{V}} 2\,\eta \, \frac{U^2}{D^2} \, D^3 \, \frac{D}{U} \, \left\|\bar{\dot{\boldsymbol{\gamma}}}\right\|^2 d\bar{V} \, d\bar{t}, \\
E_d &= \eta \, U \, D^2 \int_{\bar{t}} \int_{\bar{V}} 2 \left\|\bar{\dot{\boldsymbol{\gamma}}}\right\|^2 d\bar{V} \, d\bar{t}, \\
E_d &= \frac{\eta\sigma D}{\rho U} We \int_{\bar{t}} \int_{\bar{V}} 2 \left\|\bar{\dot{\boldsymbol{\gamma}}}\right\|^2 d\bar{V} \, d\bar{t}, \\
E_d &= \frac{\eta\sigma D}{\rho U} \bar{E}_d.
\end{aligned}
\tag{5.7}
$$

em que $\bar{E}_d = We \int_{\bar{t}} \int_{\bar{V}} 2 \left\|\bar{\dot{\boldsymbol{\gamma}}}\right\|^2 d\bar{V} \, d\bar{t}$. Para os demais testes apresentados neste trabalho, todas as energias utilizadas estarão adimensionalizadas pelas expressões 5.5, 5.6 e 5.7.

Visando buscar novas formas de diversificar os conjuntos de dados que serão gerados, pensou-se em investigar se as mudanças no formato inicial de uma gota são capazes de influenciar consideravelmente o comportamento das energias. Esta hipótese foi levantada após uma análise do trabalho de [1], no qual é explorado o impacto que gotas com diferentes formatos podem ter na dinâmica de espalhamento e colisão. Como será demonstrado nas seções seguintes, constatou-se que de fato a alteração do formato inicial de uma gota ao impactar uma parede sólida pode representar uma alternativa viável para a diversificação dos dados, resultando em mudanças tanto na magnitude das energias quanto na angulação de suas curvas pelo tempo.

6 Descrição dos conjuntos de dados

Para aplicar e testar a abordagem por redes neurais, é necessário, primeiramente, gerar datasets adequados. Para isso, são executadas simulações numéricas, utilizando uma versão adaptada do *solver* desenvolvido por [34], abrangendo dois regimes de escoamento diferentes: o impacto de uma gota com diferentes formatos iniciais em uma superfície sólida, para gerar uma variedade de comprimentos de espalhamento, e a colisão horizontal entre duas gotas, a fim de testar a capacidade do modelo em predizer características que oscilam com o tempo. Exemplos de ambos os tipos de simulações são ilustrados na Figura 30 (à esquerda). Além da geração de dados, é realizado um tratamento das instâncias, aplicando a normalização z-score e o fatiamento para gerar um conjunto maior de dados, como mostra a Figura 30 (à direita).

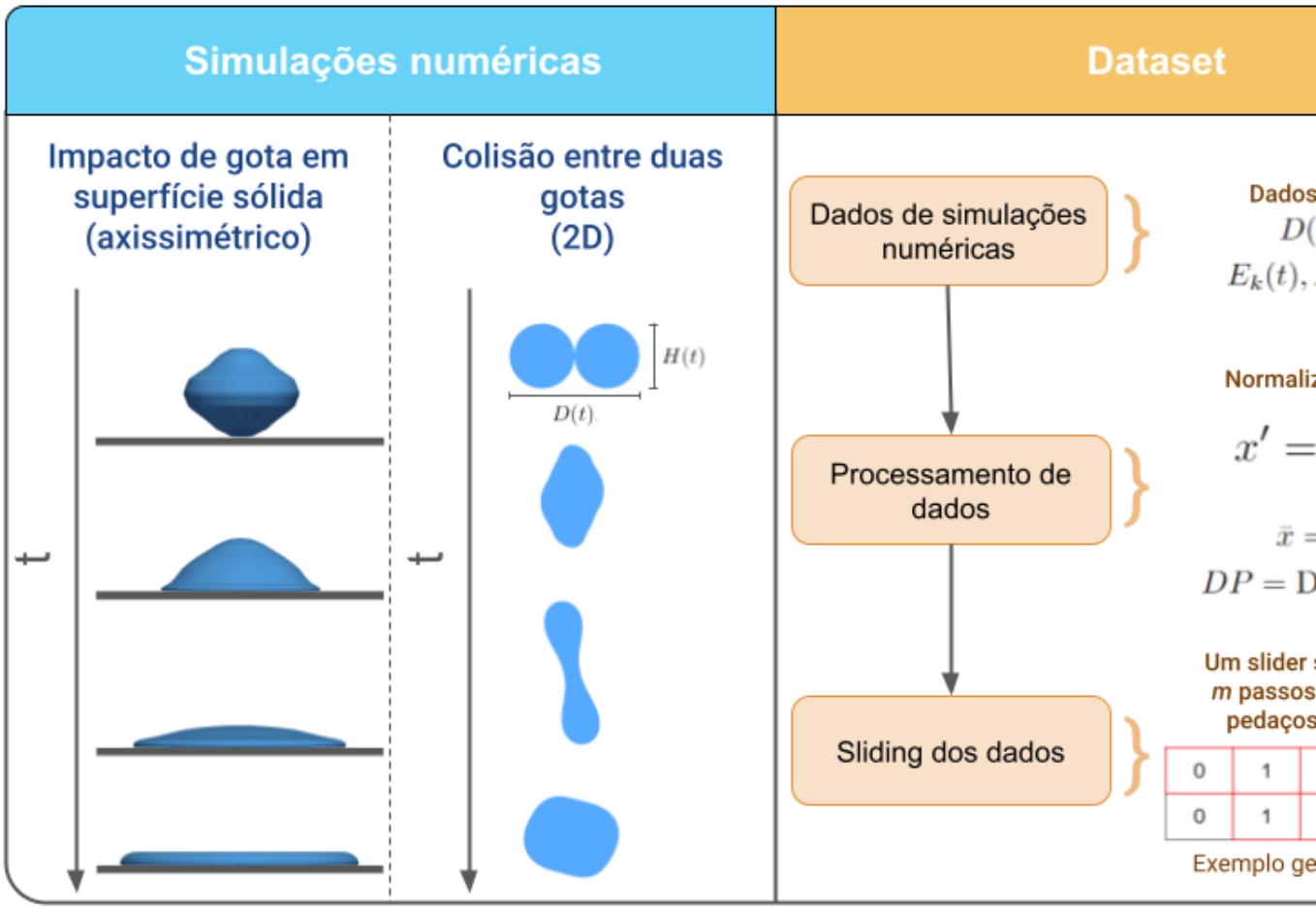


Figura 30 – Esquema de como os dados são extraídos das simulações numéricas.

Cada simulação foi rodada até o tempo $t_{\max} = 50$s, com um intervalo de tempo $\Delta t = 0.0001$, totalizando 500000 passos de tempo. Os dados geométricos e as três energias

definidas no Capítulo 5 são calculados a cada 500 passos de tempo, tendo cada instância um total de 1000 passos de tempo antes do fatiamento. Os parâmetros dos fluidos são diferentes para cada instância rodada, como será mostrado a seguir. No caso do impacto da gotícula em uma superfície sólida, o formato inicial da gotícula é variado utilizando interfaces presentes em imagens de gotículas experimentais [1], extraídas através do método apresentado no Capítulo 4. Para o dataset de colisão, o parâmetro de impacto B [43] foi adotado para introduzir variações. Em ambos os casos, também são modificados os parâmetros do fluido para gerar diferentes valores de números de Reynolds e de Weber.

Em relação aos dados coletados, eles podem ser estáticos (não mudam) ou transientes (mudam com o tempo):

estático: número de Reynolds (Re), número de Weber (We) e parâmetro de impacto (B);

transiente: energia cinética ($\bar{E}_k(t)$), energia superficial ($\bar{E}_s(t)$), dissipação viscosa ($\bar{E}_d(t)$), comprimento de espalhamento ($D(t) = max(x) - min(x)$) e altura de espalhamento ($H(t) = max(y) - min(y)$).

6.1 Dataset de espalhamento

No primeiro dataset, foram geradas gotas com formatos inspirados nos experimentos apresentados por [1]. Para testar a eficiência do modelo de aprendizado de máquina, os formatos foram escolhidos de tal forma que os subconjuntos de treinamento, validação e teste tivessem formas suficientemente variáveis, conforme mostrado na Figura 31.

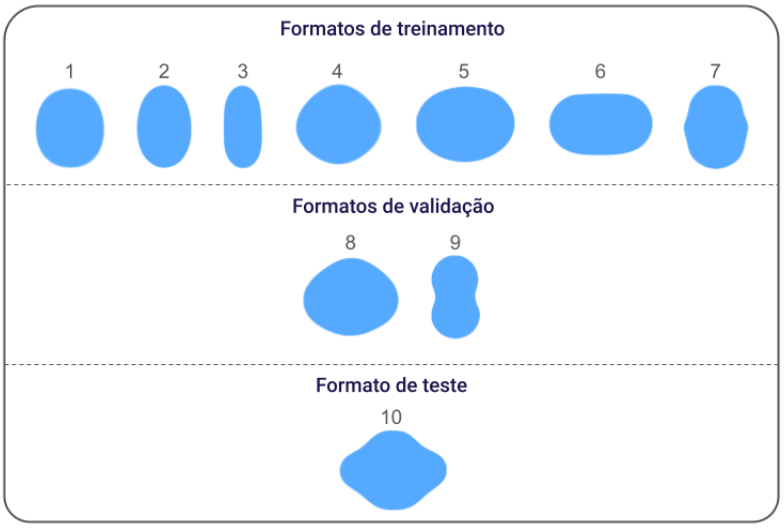


Figura 31 – Diferentes formas iniciais das gotas e a divisão do dataset em treinamento-validação-teste.

Visando extrair partículas desses formatos, o método de extração de interfaces descrito no Capítulo 4 foi utilizado. O comprimento e a altura inicial para cada um dos formatos, mostrados na Tabela 1, são escolhidos de tal forma que todas as gotas tenham aproximadamente o mesmo volume. Como aplicamos simulações axissimétricas para o caso de espalhamento, como comentado no Capítulo 2, isso é necessário para fazer com que cada gotícula tenha aproximadamente a mesma massa.

$Formato$	$D(0)$ (em mm)	$H(0)$ (em mm)
1	3.92	3.98
2	3.6771	4.64
3	2.62	4.64
4	4.64	3.6829
5	4.6086	3.1343
6	4.8143	2.5714
7	3.8229	4.24
8	4.5057	3.3857
9	3.5571	5.0086
10	4.6343	3.6229

Tabela 1 – Comprimento e altura para cada formato de gotícula.

Para cada um dos 10 formatos, foram executadas simulações numéricas com os conjuntos de parâmetros exibidos na Tabela 2. Para cada conjunto de parâmetros e formato inicial, uma simulação numérica é executada para gerar uma instância. Algumas combinações geram menos instâncias graças aos potenciais erros numéricos do *solver* utilizado, que resultam em falha

na geração de algumas instâncias. O número de Reynolds e de Weber foram arbitrariamente escolhidos variando a velocidade de impacto. No total, 132 instâncias foram extraídas com sucesso.

Dataset de espalhamento		
Formatos	Re	We
10	1.5063	240.625
10	4.5188	2165.625
10	5.875	212.5
10	10.2438	184.375
10	17.6250	1912.5
10	20.4375	118.75
10	24.8063	90.625
9	30.6313	53.125
10	30.7313	1659.375
7	35	25
10	61.3125	1068.75
9	74.4188	815.625
9	91.8938	478.125
8	105	225

Tabela 2 – Número de instâncias para cada grupo de parâmetros utilizados nas simulações de espalhamento.

Para visualizar o comportamento do espalhamento quando Re e We são variados, foi apresentado na Figura 32 a morfologia das gotículas e a evolução temporal do diâmetro para os formatos 1 e 10 da Figura 31), sendo seus comportamentos quantificados nos gráficos abaixo. Percebe-se que apesar das séries temporais terem trajetórias simples, há uma grande mudança no comportamento do espalhamento entre formatos diferentes.

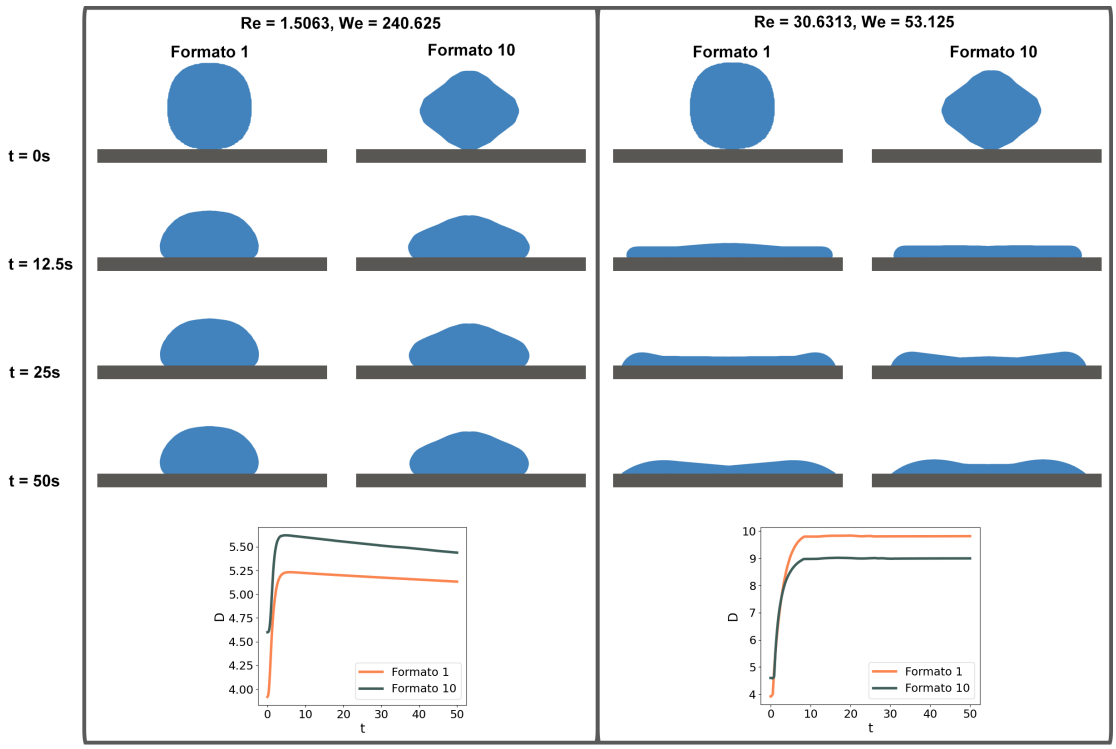


Figura 32 – Efeito da mudança de formato inicial (formatos 1 e 10) no impacto ao variar Re e We.

Além disso, é perceptível, ao analisar a Figura 33, como o comportamento das séries temporais extraídas ($D(t)$, $E_k(t)$, $E_s(t)$ e $E_d(t)$) também apresenta diferenças consideráveis ao modificar o formato da gota. Com isso, confirma-se a hipótese de que a forma inicial influencia significativamente o comportamento do escoamento.

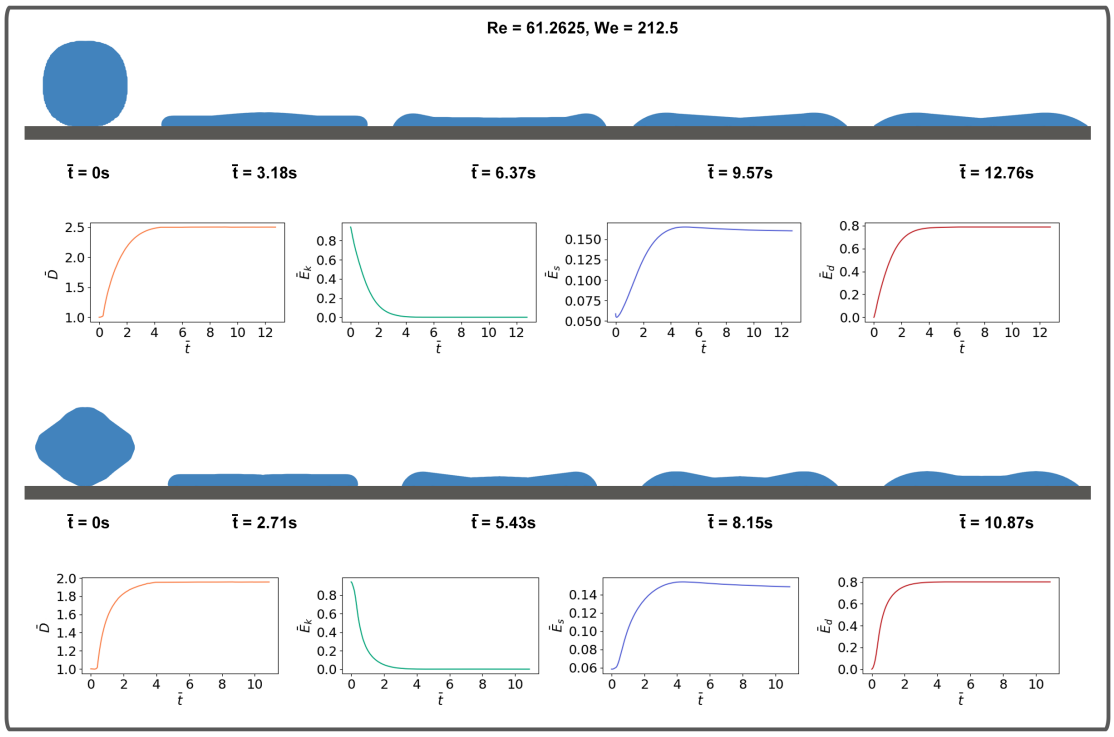


Figura 33 – Dados de séries temporais extraídos de simulações com os mesmos parâmetros, porém diferentes formatos de gota.

6.2 Dataset de colisão

O segundo dataset consiste em simulações de duas gotas com 1 mm de diâmetro colidindo horizontalmente uma com a outra. As gotas se fundem após a colisão e geram oscilações horizontais e verticais ao longo do tempo, ajudando a criar séries temporais mais complexas que podem ser utilizadas para testar a eficiência do modelo de aprendizado de máquina.

Para definir os parâmetros das simulações numéricas para cada instância, casos semelhantes aos apresentados no mapa experimental de coalescência de [43] foram selecionados variando a velocidade de impacto. Para cada conjunto de números de Reynolds e de Weber, 20 simulações são executadas variando o parâmetro de impacto B nos intervalos mostrados na Tabela 3. Novamente, algumas instâncias não rodaram com sucesso devido a erros no *solver*. No total, 72 instâncias foram geradas.

Dataset de colisão			
Instâncias	Re	We	B (intervalo)
20	74.451	22.0	[0.0, 0.4]
20	85.4788	29.0	[0.0, 0.4]
9	97.8478	38.0	[0.0, 0.2316]
5	104.0863	43.0	[0.0211, 0.1684]
7	109.9715	48.0	[0.0421, 0.1684]
6	115.5573	53.0	[0.0421, 0.2105]
5	120.8853	58.0	[0.0, 0.2316]

Tabela 3 – Número de instâncias para cada grupo de parâmetros utilizados nas simulações de colisão.

Para visualizar a diferença no espalhamento horizontal e vertical após o impacto causado pelas mudanças nos parâmetros físicos e no parâmetro de impacto, as capturas dos passos de algumas instâncias simuladas são mostradas na Figura 34. Enquanto os dois primeiros exemplos ilustram a dinâmica ao variar apenas Re e We, o primeiro e o último demonstram o efeito ao modificar apenas B.

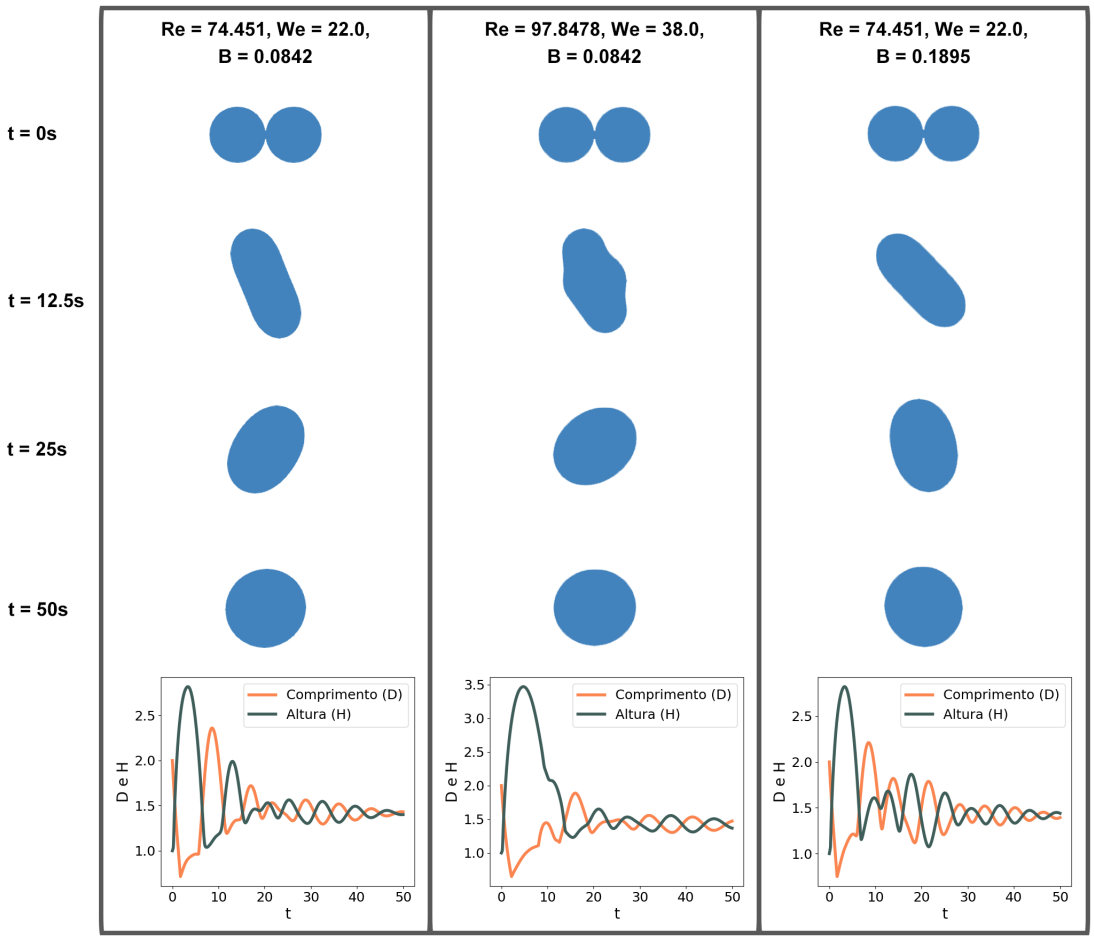


Figura 34 – Dinâmica de colisão de gotículas para diferentes parâmetros de impacto B variando Re e We.

Por fim, podemos visualizar na Figura 35 a diferença no comportamento das simulações ao alterar o parâmetro de impacto e como isso ajuda a diversificar as séries temporais do dataset.

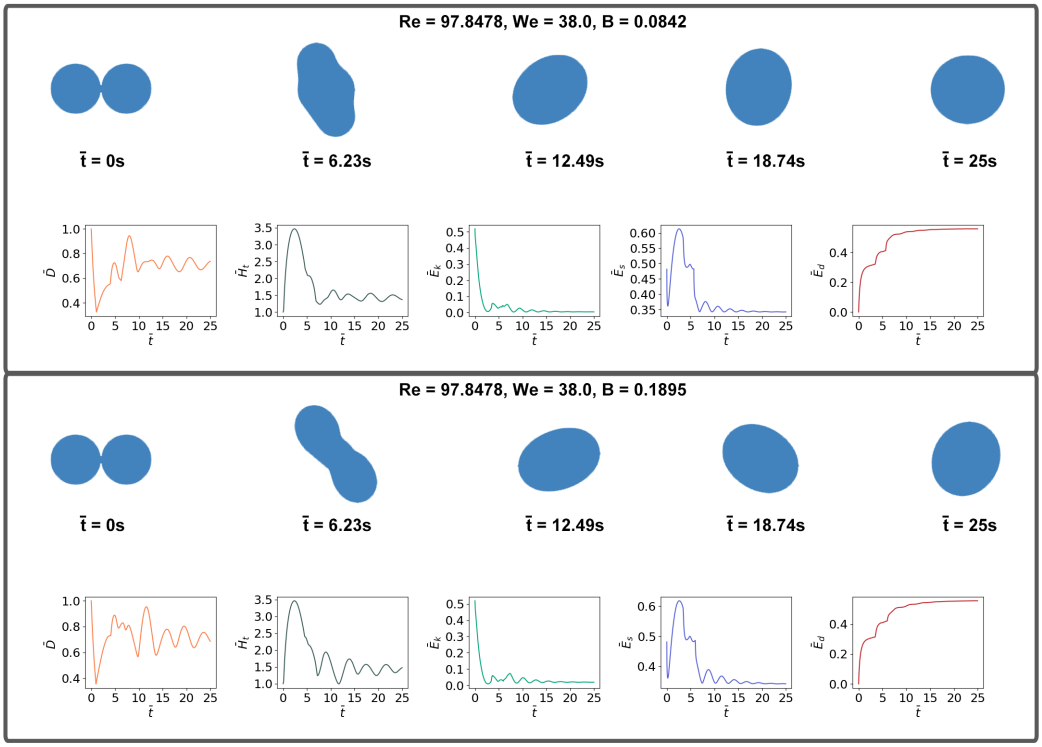


Figura 35 – Dados de séries temporais extraídos de simulações com diferentes parâmetros de impacto.

Dessa forma, foram gerados dois regimes diferentes nos quais determinadas características são variadas para gerar mudanças no comportamento das séries temporais, possibilitando assim o teste de eficiência da arquitetura de rede neural escolhida.

7 Aprendizado de máquina para predição de dados

Neste capítulo, são apresentados o funcionamento de uma arquitetura de rede neural recorrente chamada Long Short-Term Memory (LSTM), as predições de interesse que serão abordadas e uma estratégia desenvolvida para aumentar os conjuntos de dados, tornando-os ainda mais eficientes para o treinamento.

7.1 Long Short-Term Memory (LSTM)

A Long Short-Term Memory (LSTM) [56] é um tipo de rede neural recorrente (RNN) [57] que trás consigo dois novos mecanismos em relação à RNN tradicional: um canal de memória, responsável por transmitir informações de longo prazo, e três portões, encarregados de controlar quais informações devem ou não passar. Para compreender o funcionamento dessas modificações, é necessário, primeiramente, compreender como uma RNN comum funciona e quais são os motivos para adotar tais modificações.

Em conjuntos de dados nos quais as variáveis sofrem mudanças ao longo do tempo, também conhecidos como séries temporais, passos de tempo iniciais podem conter informações úteis para a definição de características de passos mais avançados e vice-versa. Essa relação temporal entre os dados não consegue ser capturada eficientemente ao utilizar uma rede neural *feedforward* comum, uma vez que a memória de passos de tempos anteriores não é carregada.

Para lidar com essa limitação, foram criadas as redes neurais recorrentes (RNNs), capazes de processar séries temporais de forma a transmitir informações de um passo de tempo para os seguintes. Na Figura 36 é mostrada a arquitetura básica de uma RNN, na qual x_t e $\bar{y}_t$ são, respectivamente, os dados de entrada e de saída para cada passo de tempo, e a célula RNN representa um bloco da rede responsável por gerar uma saída e passar as informações extraídas até o passo t para o tempo $t + 1$.

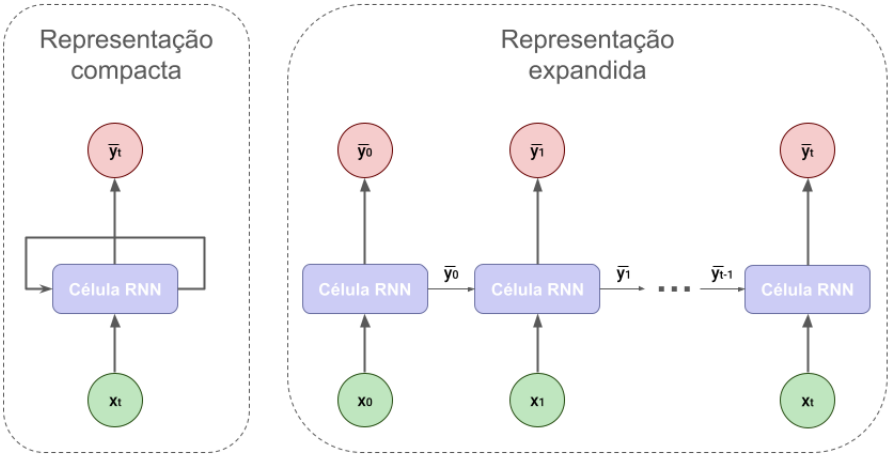


Figura 36 – Estrutura de uma rede neural recorrente.

A parte interna de uma célula, conforme ilustrada na Figura 37, contém uma camada de rede neural que aplica uma função de ativação (como a tangente hiperbólica, para o exemplo) na concatenação dos dados de entrada x_t com a saída da célula anterior $\bar{y}_{t-1}$, resultando em uma saída $\bar{y}_t$ que é transmitida para o próximo passo.

A estrutura da RNN tradicional permite a passagem de informações entre passos de tempo mais próximos. No entanto, essa arquitetura pode enfrentar problemas ao lidar com dependências temporais com muitos passos, nas quais a informação precisa ser transportada por diversas células RNN até chegar em um determinado tempo t.

O acúmulo de informações de muitos passos temporais anteriores pode gerar problemas com o gradiente [58], que é utilizado na atualização dos pesos durante o treinamento. A diminuição e o aumento excessivo do gradiente são conhecidos, respectivamente, como problema de dissipação do gradiente e problema de explosão do gradiente, costumando aparecer também em redes neurais *feedforward* muito profundas.

Ao gerar uma saída, a rede aplica uma função de *loss* para quantificar a discrepância entre o resultado obtido e o esperado. Através do valor gerado, é aplicado o algoritmo *backpropagation* para atualizar os pesos de cada camada com o objetivo de melhorar a acurácia da rede. Os problemas citados anteriormente ocorrem quando a derivada das funções de ativação, a arquitetura da rede e/ou os hiperparâmetros acabam causando um aumento (explosão) ou uma diminuição (dissipação) muito acentuada no valor do gradiente, causando distúrbios nos pesos e aumentando ou estagnando o *loss* nas próximas épocas. Uma explicação mais detalhada sobre problemas do gradiente, inclusive suas causas e consequências, pode ser encontrada no material de [59].

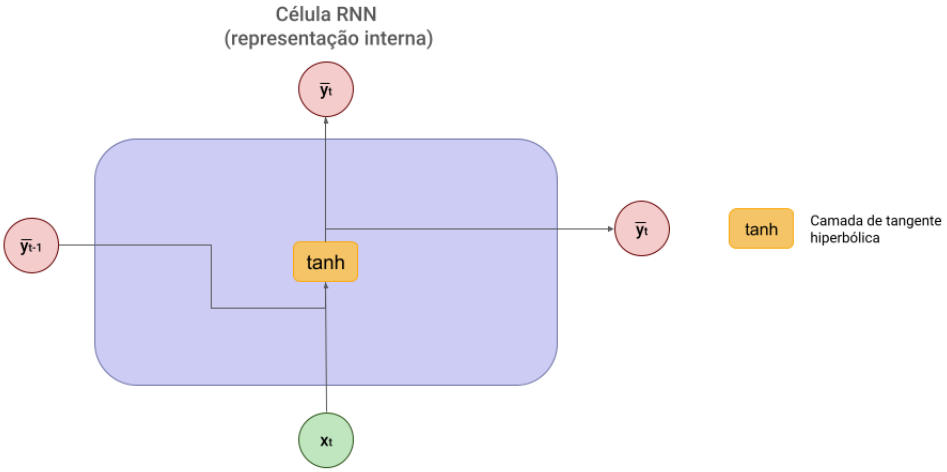


Figura 37 – Interior de uma célula RNN.

A arquitetura LSTM, apresentada na Figura 38, foi desenvolvida com o intuito de aprimorar as redes neurais recorrentes para que sejam capazes de carregar informações por mais passos temporais e evitar os problemas associados ao gradiente. Como as séries temporais utilizadas no presente trabalho envolvem um grande número de passos de tempo coletados, o uso de redes LSTM será fundamental para otimizar os resultados obtidos.

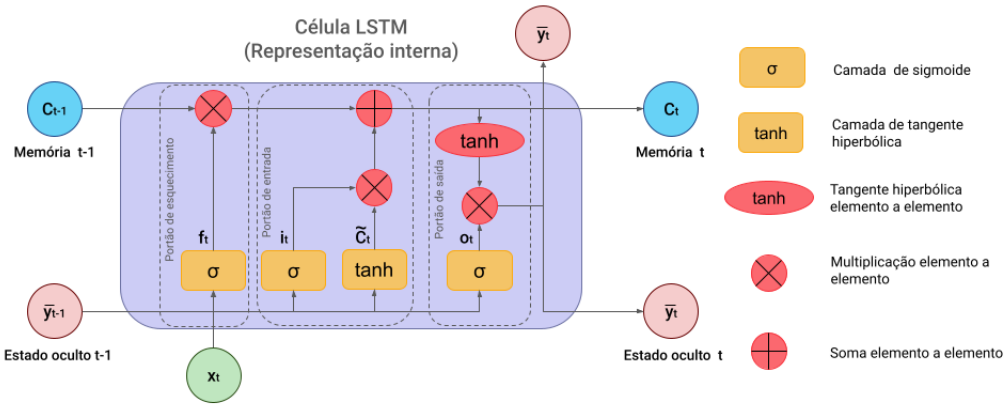


Figura 38 – Interior de uma célula LSTM.

Na Figura 38, os retângulos amarelos representam redes neurais com uma camada, onde σ é a função de ativação sigmoide e $tanh$ é a tangente hiperbólica, enquanto as elipses vermelhas representam operações elemento a elemento. Em relação à organização vista na RNN, há duas principais modificações: a criação de um novo caminho para a passagem de memória dos passos anteriores e um conjunto de portões que controlam a intensidade das informações que passam por eles.

O caminho da memória, como destacado na Figura 39, é uma passagem pela qual as

informações provenientes de passos anteriores podem transitar sem sofrer muitas transformações, sendo aplicadas apenas operações simples para adicionar novas informações extraídas de cada passo e apagar informações irrelevantes. Em contraste com o estado oculto, que carrega principalmente informações do passo de tempo anterior, o caminho da memória é capaz de transmitir dados através de várias células.

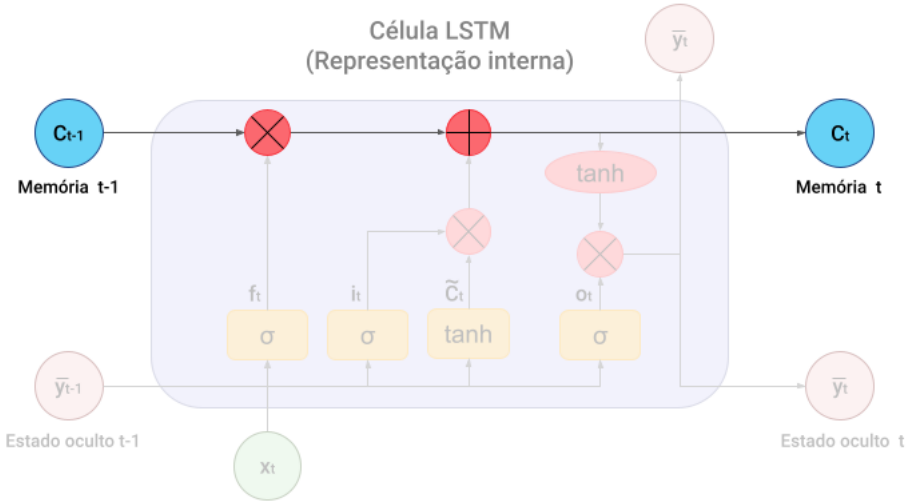


Figura 39 – Caminho pelo qual é transmitida a memória dos passos anteriores.

Já os mecanismos de portão, presentes em três partes da célula e realçados na Figura 40, são compostos por uma rede neural com função de ativação sigmoide cujas saídas, conhecidas também como “chaves”, são multiplicadas elemento a elemento pelos dados que passam pelos portões. Como a função sigmoide gera valores entre 0 e 1, as multiplicações realizadas pelos portões são responsáveis por controlar a intensidade com que os dados passarão por eles.

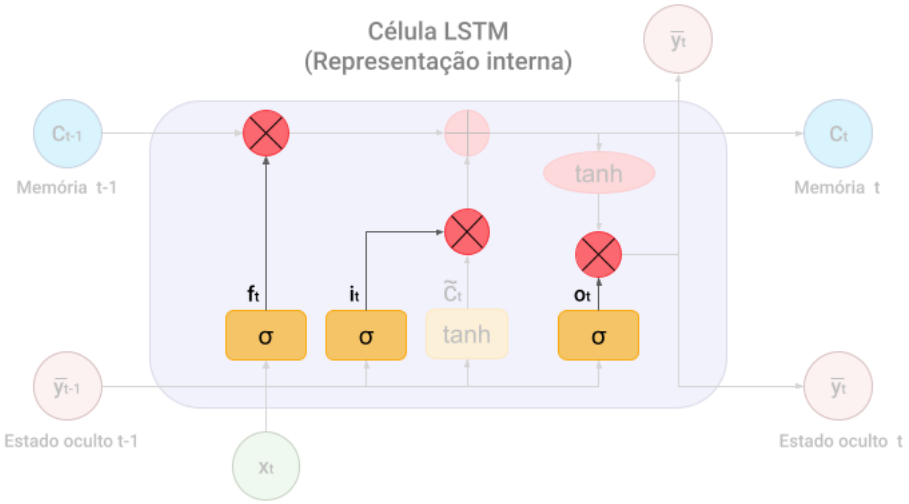


Figura 40 – Os três portões que controlam a passagem de informações.

Para explicar o funcionamento em conjunto dos componentes, será apresentado a seguir o passo a passo de como os dados chegam a uma célula t, geram a saída $\bar{y}_t$ e passam informações para a célula $t+1$.

No caminho inferior da célula, há uma concatenação dos valores da saída no tempo anterior $\bar{y}_{t-1}$ com os valores de entrada no tempo atual x_t. Esta concatenação passa por um mecanismo de portão conhecido como portão de esquecimento, como ilustrado na Figura 41, responsável por controlar a intensidade com que os dados da memória C_{t-1} serão passados adiante.

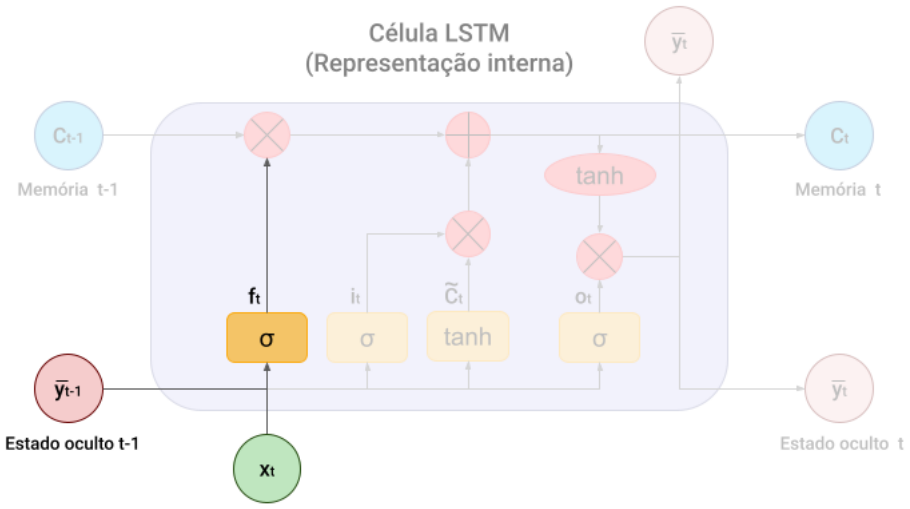


Figura 41 – Geração dos valores de controle para a passagem da memória C_{t-1}.

As chaves de controle f_t do portão de esquecimento são definidas como:

$$f_t = \sigma(W_f \cdot [\bar{y}_{t-1}, x_t] + b_f) \tag{7.1}$$

sendo W_f os pesos da rede, $[\bar{y}_{t-1}, x_t]$ a concatenação da saída da célula anterior com a entrada da célula atual e b_f um viés somado ao resultado da multiplicação dos pesos pelas entradas.

Seguindo o caminho, há uma bifurcação em que os dados são replicados e seguem dois caminhos diferentes. Um dos caminhos passa por mais um mecanismo de portão, chamado portão de entrada, para gerar chaves de controle i_t. Já o outro, passa por uma rede neural que aplica a função de ativação tangente hiperbólica para processar os dados e gerar novas informações $\tilde{C}_t$ a serem incorporadas na memória. Os valores de controle i_t são então aplicados na saída da rede

de processamento $\tilde{C}_t$ para decidir com que intensidade as novas informações entrarão no canal de memória.

Ilustrada na Figura 42, essa é a única etapa responsável por gerar novas informações, sendo as demais apenas responsáveis por controlar a intensidade dos valores passados adiante.

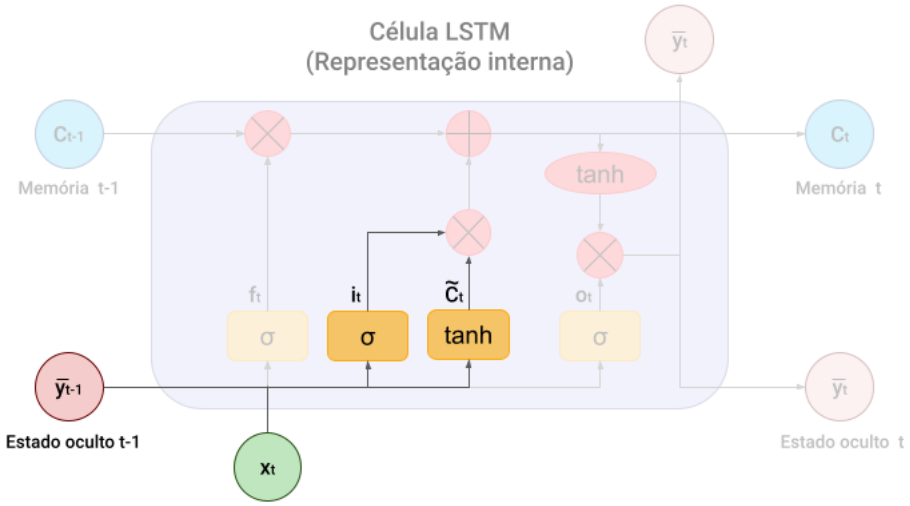


Figura 42 – Formação e controle de novas informações para a memória.

As novas informações geradas $\tilde{C}_t$ e as chaves i_t são definidas da mesma forma que f_t, mudando apenas a função de ativação para a $\tilde{C}_t$:

$$i_t = \sigma(W_i \cdot [\bar{y}_{t-1}, x_t] + b_i) \tag{7.2}$$

$$\tilde{C}_t = tanh(W_C \cdot [\bar{y}_{t-1}, x_t] + b_C) \tag{7.3}$$

Com isso, todos os mecanismos necessários para definir o novo estado do caminho C_t foram descritos. Enquanto o portão de esquecimento controla quanto dos dados do antigo estado continuará a passar, o portão de entrada controla quanto das novas informações será adicionado ao estado da célula atual.

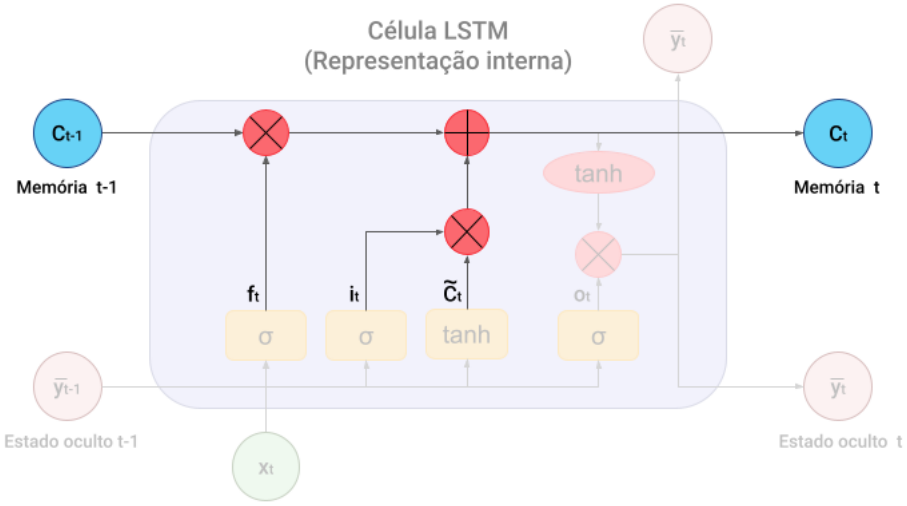


Figura 43 – Passagem de informação pelo caminho da memória.

Para atualizá-lo, o caminho da Figura 43 é seguido, resultando em:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{7.4}$$

Para gerar a saída $\bar{y}_t$ da rede, primeiramente os dados do estado oculto passam pelo terceiro e último mecanismo de portão, conhecido como portão de saída, gerando chaves de controle o_t. Em seguida, os dados do contexto C_t são transformados pela função de tangente hiperbólica. Por fim, o resultado da transformação é multiplicado elemento a elemento pelos valores o_t, gerando $\bar{y}_t$. Estes valores são replicados e enviados tanto para a saída da célula t quanto para a célula do tempo seguinte $t + 1$. Este passo está representado na Figura 44.

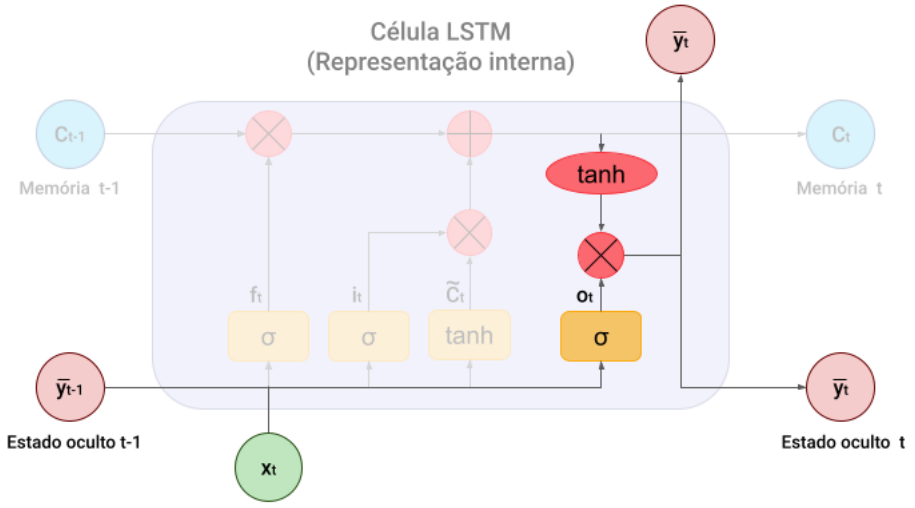


Figura 44 – Geração da saída no passo de tempo t ($\bar{y}_t$).

sendo o portão o_t e a saída $\bar{y}_t$ definidos como:

$$o_t = \sigma(W_o[\bar{y}_{t-1}, x_t] + b_o) \tag{7.5}$$

$$\bar{y}_t = o_t * tanh(C_t) \tag{7.6}$$

Para maiores detalhes sobre modificações e variantes existentes para esta arquitetura, vide o trabalho de [56].

7.2 Predições de interesse

As características mais interessantes de se prever, seja em um sentido numérico ou experimental, podem ser divididas em dois tipos de acordo com o comportamento das saídas ao longo do tempo. Elas podem ser estáticas, em que cada característica possui apenas um valor, ou transientes, onde cada característica possui valores que mudam ao longo do tempo. Nosso trabalho foca em previsões transientes, mas explorações de predições estáticas também foram realizadas no Capítulo 8.3.2 e no Apêndice A.

Para a entrada, porém, podem haver tanto características estáticas quanto transitórias ao mesmo tempo. Como as dimensões de entrada precisam ser iguais para cada característica, todos os dados estáticos são replicados n vezes, sendo n o número de passos de tempo, para que cada um tenha múltiplos passos temporais com o mesmo valor. Por exemplo, para a entrada do número adimensional Re, a repetição geraria $Re(1) = Re(2) = \cdots = Re(n)$.

Em relação às camadas da rede, é aplicado um certo número de camadas LSTM, uma camada Densa para gerar valores no intervalo esperado para as previsões e uma camada Reshape para controlar a forma da saída. Os esquemas para as relações entrada/saída exploradas e a estrutura geral das redes neurais definidas neste trabalho são ilustrados na Figura 45.

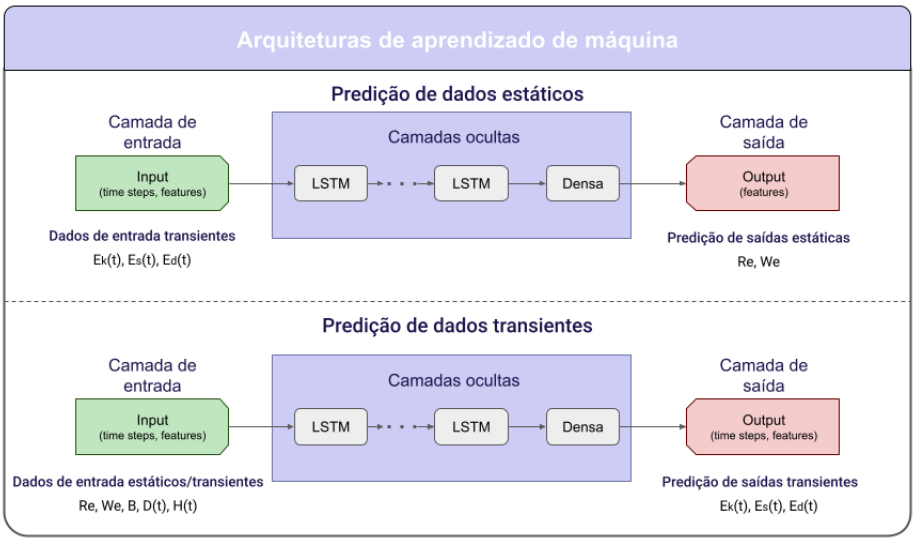


Figura 45 – Arquiteturas de aprendizado de máquina utilizadas nas predições estáticas e dinâmicas.

Um exemplo de representação visual da relação input/output é mostrado na Figura 46, sendo escolhido o caso de predição das energias no regime de colisão.

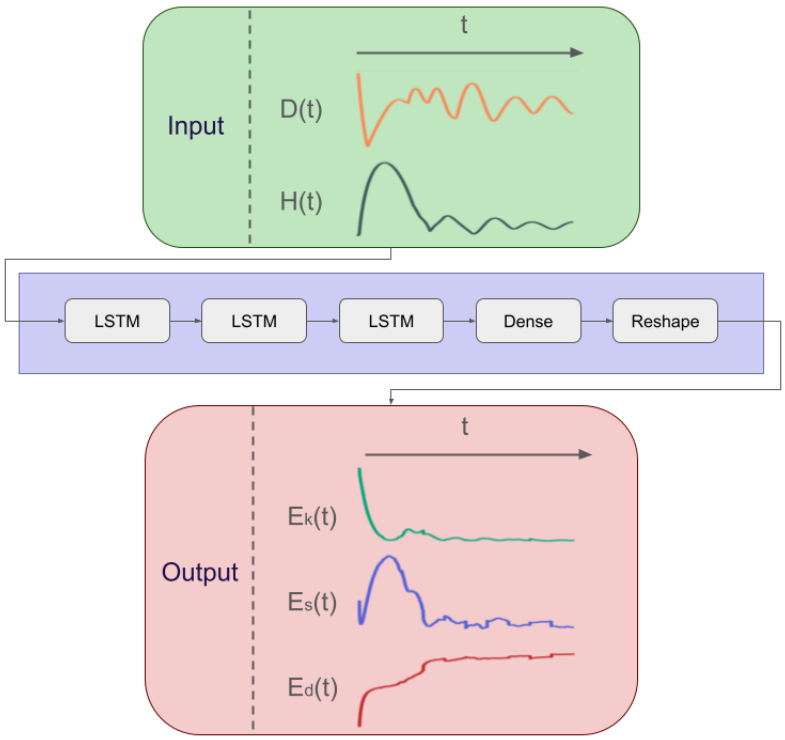


Figura 46 – Esquema para predição das energias usando dados geométricos no problema da colisão entre gotículas.

Para melhor organizar as predições exploradas, todos os casos de teste são mostrados na Tabela 4, caracterizados pelo tipo de output (estático ou transiente), o dataset utilizado, as variáveis de entrada as características de saída que serão preditas.

Casos testados			
Tipo	Dataset	Entradas	Saídas
Transiente	Espalhamento	$t, D(t)$, Re, We	$E_k(t), E_s(t), E_d(t)$
Transiente	Colisão	$D(t), H(t)$, Re, We, B	$E_k(t), E_s(t), E_d(t)$
Transiente	Colisão	$D(t), H(t)$	$E_k(t), E_s(t), E_d(t)$
Estático	Colisão	$E_k(t), E_s(t), E_d(t)$	Re, We

Tabela 4 – Listagem dos casos explorados.

7.3 Fatiamento dos dados

Apesar da alta capacidade da LSTM de manter a memória por muitos passos de tempo, a arquitetura ainda está sujeita a um número máximo ideal de passos para evitar os problemas do gradiente, visto que os dados que passam pelo caminho da memória são incrementados a cada passo de tempo.

Embora não haja regras que definam um limite de passos, geralmente é recomendado um máximo de 200 a 300 passos por instância, dependendo do dataset. Para evitar sobrecarregamentos das redes LSTM nos testes, foi aplicada uma técnica para separar os dados em fatias com m passos de tempo cada, sendo $m < 200$. Desta forma, além de diminuir a complexidade do treinamento, também são geradas novas instâncias que acabam aumentando o tamanho do dataset.

O fatiamento é feito através de um ponteiro que pula de n em n passos e coleta os m próximos passos de tempo de uma instância. Para ambos os conjuntos, escolheu-se um salto de $n = 50$ e a coleta de $m = 100$ passos, sendo o comportamento do fatiamento demonstrado na Figura 47.

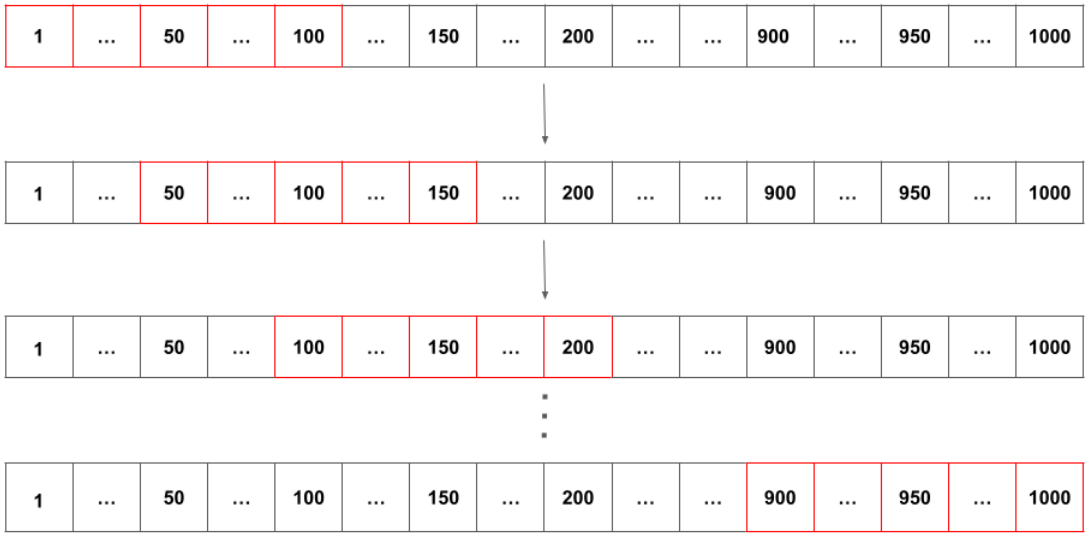


Figura 47 – Exemplo de funcionamento do fatiamento dos dados.

Ao final do fatiamento de uma instância com 1000 passos de tempo, tem-se a formação de 19 novas instâncias com 100 passos cada.

8 Resultados

Uma rede neural LSTM supervisionada foi aplicada para treinar modelos capazes de predizer o balanço de energia tendo como base dados facilmente extraídos de simulações numéricas ou de experimentos, como séries temporais com características geométricas de gotículas. Para testar a performance preditiva e a adaptabilidade da arquitetura escolhida, inicialmente foram treinados dois modelos, cada um considerando um dos datasets definidos anteriormente.

8.1 Performance dos modelos formulados

Primeiramente, como o ajuste de hiperparâmetros é uma etapa crucial para o desenvolvimento de modelos de aprendizado de máquina eficientes, no presente trabalho foi utilizado o framework de otimização de hiperparâmetros de código aberto *Optuna* [60]. Após investigações considerando os datasets obtidos através das simulações de gotículas com diferentes formatos iniciais impactando superfícies sólidas e de colisões entre gotículas, os hiperparâmetros foram definidos da seguinte maneira:

- Taxa de aprendizado de 0.0001;
- Máximo de 5000 épocas;
- Tamanho de lote de 8 para o espalhamento e de 32 para a colisão;
- 3 camadas ocultas LSTM com 32 neurônios cada;
- 1 camada de output Densa com $n \times m$ neurônios, onde n representa o número de passos de tempo e m representa o número de atributos;
- 1 camada Reshape para controlar o formato da saída (apenas para saídas transientes);
- Adam escolhido como algoritmo de otimização;
- Erro quadrático médio utilizado como função de *loss*.

Adicionalmente, uma parada antecipada baseada no *loss* de treinamento (colisão) ou no *loss* de validação (espalhamento) é aplicada. Para medir a capacidade preditiva de cada modelo, foram selecionadas algumas das métricas mais comumente utilizadas na avaliação de modelos de aprendizado de máquina. Essas métricas incluem o Coeficiente de Determinação - R^2 (8.1), a Raiz do Erro Quadrático Médio - $RMSE$ (8.2) e a Raiz do Erro Quadrático Médio Normalizada - $NRMSE$ (8.3), definidas como segue:

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2}, \tag{8.1}$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \tilde{y}_i)^2}, \tag{8.2}$$

$$NRMSE = \frac{RMSE}{y_{max} - y_{min}}, \tag{8.3}$$

onde N é o número de instâncias, y_{max} e y_{min} são, respectivamente, os valores mínimo e máximo de um atributo, y_i e $\tilde{y}_i$ representam, respectivamente, os valores esperado e predito, e $\bar{y}$ é a média de todos os valores esperados de um atributo.

O R^2 é aplicado para avaliar quão bem a variabilidade das variáveis dependentes (energias) pode ser explicada pela variabilidade das variáveis independentes (dados geométricos e números adimensionais). Uma pontuação próxima de 0 indica uma capacidade explicativa fraca, enquanto uma pontuação próxima de 1 significa uma forte capacidade explicativa para a variável dependente. Já o $RMSE$ e o $NRMSE$ são utilizados para avaliar quantitativamente a qualidade das predições. Enquanto o $RMSE$ gera um valor dependente da escala das variáveis preditas, o $NRMSE$ é utilizado para fornecer uma medida relativa de erro independente da escala dos dados. Em ambos os casos, um valor próximo de zero indica um modelo bem ajustado.

Enquanto as métricas R^2 (8.1) e $RMSE$ (8.2) são usadas para medir individualmente a eficiência de cada modelo, $NRMSE$ (8.3) é utilizada para comparar diferentes modelos e verificar a adaptabilidade da arquitetura escolhida.

Para o dataset de espalhamento, 10 formatos diferentes são separados em 70% treinamento (7 formatos, 95 instâncias), 20 % validação (2 formatos, 27 instâncias) e 10 % teste (1 formato, 10 instâncias). Como já comentado, para otimizar o processo de treinamento, o *loss* de validação foi utilizado como parâmetro de parada antecipada.

O *score* R^2 para cada instância, calculado de forma individual, varia de 0.98315 a 0.9994,

enquanto o $RMSE$ varia de 0.01122 a 0.05893. No geral, ambas as métricas estão dentro de intervalos de valores ideais e a maioria das predições caem em uma faixa de erro de ±10%. Além disso, observa-se que essas métricas permanecem bastante consistentes em todas as instâncias, fornecendo mais evidências de que o modelo está bem ajustado.

Para o dataset de colisão, 72 instâncias são divididas em 70% treinamento (50 instâncias) e 30 % teste (22 instâncias). Como não são separados dados de validação, o processo de parada antecipada é dado levando em conta o *loss* de treinamento.

Os *scores* R^2 para cada uma das 22 instâncias de teste variam de 0.95291 a 0.99997, enquanto os $RMSE$s variam de 0.00115 to 0.04618. Um dos motivos para o aumento nos intervalos de valores é a maior variabilidade no comportamento das energias, graças às oscilações causadas pela colisão. Outro fator significativo é a existência de menos instâncias no dataset de treinamento com maiores valores de Re e We, como mostrado na Tabela 3, resultando em uma acurácia menor para predições dentro desse intervalo. Apesar dessa diferença, ambas as métricas ainda estão próximas dos valores ótimos.

Anteriormente, as métricas foram analisadas para cada instância individualmente. Para avaliar a precisão geral das previsões dos modelos, a Tabela 5 compara as métricas de erro levando em conta todas as instâncias de treinamento e todas as de teste para ambos os datasets.

Métrica	Instâncias de treinamento		Instâncias de teste	
	Espalhamento	Colisão	Espalhamento	Colisão
R^2	0.9924	0.99996	0.99429	0.99559
$RMSE$	0.03677	0.00124	0.02957	0.0137
$NRMSE$	0.03674	0.0019	0.02961	0.02096

Tabela 5 – Comparação das métricas de avaliação para cada dataset.

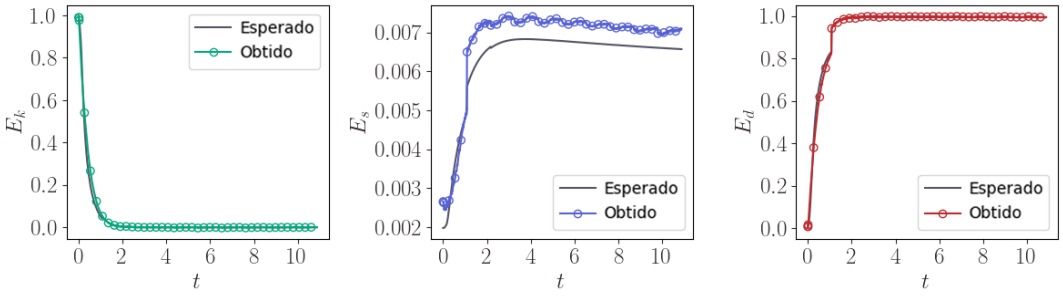
Uma vez que os datasets de treinamento são utilizados no treinamento do modelo, já era esperado que suas métricas também fossem ótimas. No entanto, o real interesse de comparar as métricas para treinamento e para teste é descobrir quão bem o modelo adquiriu adaptabilidade a novos datasets que não foram utilizados durante o treinamento. De acordo com o observado, as predições para os datasets de teste mantiveram métricas aceitáveis, mesmo apresentando instâncias com características diferentes das usadas nos datasets de treinamento.

Em relação à comparação entre os dois modelos, a métrica mais importante a ser analisada

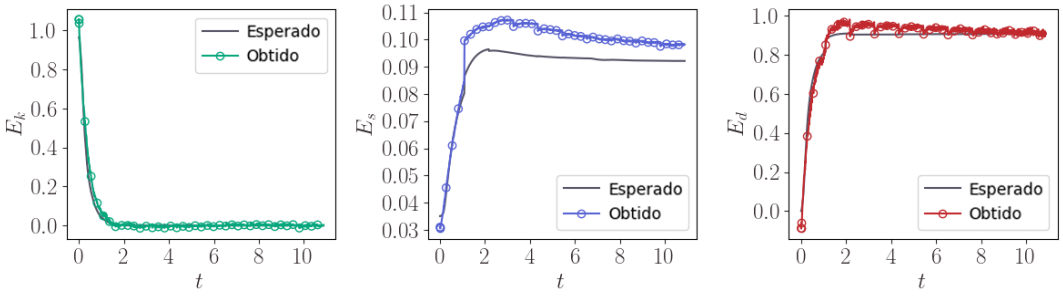
é a $NRMSE$, por oferecer uma medida normalizada do erro que desconsidera a escala dos dados. Comparando-os, podemos inferir que a arquitetura escolhida não apenas foi razoavelmente precisa, mas também altamente versátil para diferentes regimes. Além disso, verifica-se que apesar do modelo de colisão ter menos instâncias de treinamento do que o modelo de espalhamento, o erro obtido foi menor. Isso talvez possa indicar que a generalização do modelo para diferentes formatos iniciais de gotículas pode ser uma tarefa mais complexa do que generalizar o comportamento para modificações nos parâmetros do escoamento e do fluido.

8.2 Análise da predição das energias

Para melhor visualizar a eficiência de cada modelo, foram analisados os comportamentos das predições para diferentes instâncias dos datasets de teste de cada problema. Para o caso do espalhamento, foram estudadas duas instâncias do dataset de testes mostradas na Figura 48. O gráfico de cada energia é normalizado pela energia total no primeiro passo de tempo ($E_t(0)$).



(a) 1ª instância - Re = 30.73, We = 1659.38.



(b) 4ª instância - Re = 24.81, We = 90.62.

Figura 48 – Comparação entre as energias esperadas e as obtidas em duas instâncias do dataset de testes do espalhamento.

Nas Figuras 48a e 48b, podemos observar flutuações nas predições causadas pela técnica de fatiamento, apresentada no Capítulo 7.3, que foi aplicada nos datasets. Como cada instância é fatiada em múltiplas sub-instâncias, as predições podem oscilar entre o fim de uma fatia e o

começo de outra, como pode ser observado na energia superficial E_s mostrada na Figura 48a e na dissipação viscosa E_d da Figura 48b. Embora a técnica pareça introduzir erros devido a essas perturbações, testes mostraram que, na verdade, o uso da técnica aumenta bastante a acurácia do modelo. Isso ocorre devido ao aumento do número de instâncias e à redução no número de passos de tempo de cada sub-instância, o que acaba reduzindo a complexidade do treinamento.

Seguindo as investigações, também foi observado que a maioria das instâncias dos datasets de treinamento e de teste apresentam energias superficiais com valores muito menores em comparação com as demais energias, como pode ser visto na Figura 48a, por exemplo, fazendo com que seus erros de predição acabem sendo os mais visualmente acentuados.

A predominância de instâncias dentro do intervalo de valores baixos da energia superficial também influencia o processo de treinamento. Instâncias com números de Weber menores, que exibem maiores efeitos da tensão superficial, apresentam energias superficiais maiores do que a maioria das instâncias. Essas poucas instâncias com maior energia superficial acabam tendo a acurácia parcialmente comprometida, como mostrado na Figura 48b, visto que o modelo é ajustado principalmente para valores menores. Por motivos semelhantes, um número de Reynolds maior pode impactar negativamente a predição da energia cinética E_k, reduzindo a precisão da curva nos passos que antecedem o momento em que a gota para de se mover.

Por fim, a principal diferença no comportamento das energias do dataset de espalhamento se encontra na inclinação de suas curvas. Como o formato inicial da gota é um dos principais fatores que influenciam esse comportamento, a análise sugere que, mesmo ao testar um formato diferente dos usados no treinamento (vide Tabela 31), o modelo ainda foi capaz de gerar predições precisas para as instâncias de teste.

Em relação aos casos de colisão entre gotas, é adicionada uma outra camada de complexidade ao problema, visto que as energias agora exibem oscilações ao longo do tempo. Para o dataset de teste da colisão, são analisados os comportamentos da predição para quatro instâncias, mostradas na Figura 49. Novamente, os gráficos são normalizados pela energia total no primeiro passo de tempo.

As Figuras 49a e 49b demonstram a diferença ao modificar apenas o parâmetro de impacto (B), enquanto as Figuras 49c e 49d evidenciam os efeitos causados pela mudança dos números de Reynolds (Re) e de Weber (We). De acordo com as figuras, podemos confirmar uma excelente concordância entre os resultados esperados e os obtidos.

Na Figura 49a, podemos observar que o modelo pode gerar predições acuradas mesmo quando há "inconsistências" nas curvas (como visto no tempo $t \approx 6$). Além disso, embora as flutuações entre sub-instâncias ainda sejam detectáveis, como na Figura 49b, elas ocorrem muito menos frequentemente.

Ao aumentar Re e We nós, respectivamente, aumentamos E_k e diminuímos E_s, como ilustrado nas Figuras 49c e 49d. Essa alteração também influencia a frequência da oscilação, já que uma E_ks maior induz mais movimento e uma E_s menor implica que cada expansão da gota coalescida levará mais tempo para recuar ou estabilizar.

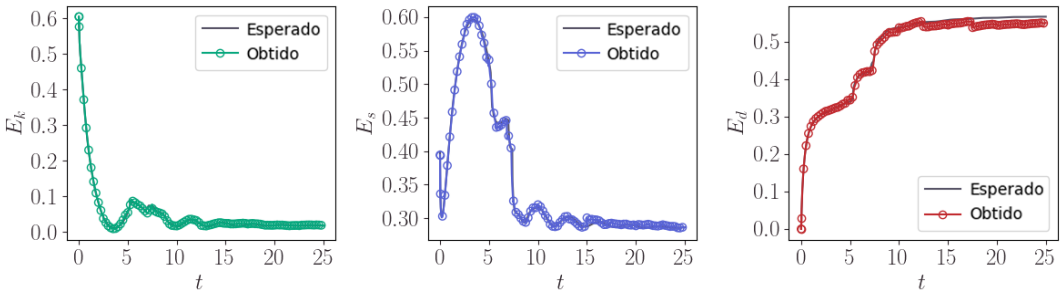
Em seguida, para examinar a significância dos números adimensionais como parâmetros de entrada da rede e explorar a possibilidade de construir um modelo sem eles, um estudo foi conduzido utilizando o dataset de colisão por meio de uma abordagem de predição em duas etapas.

8.3 Predição em duas etapas com redes neurais sequenciais

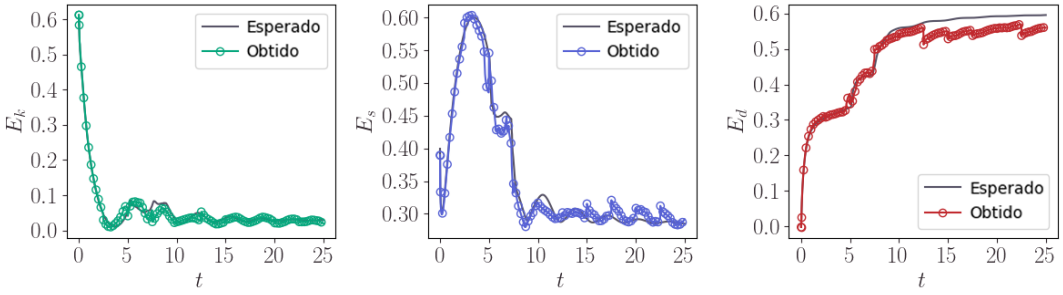
Inicialmente, consideramos apenas o diâmetro $D(t)$ e a altura $H(t)$ da gotícula como parâmetros de entrada para o modelo. Durante a primeira etapa, o objetivo é estimar, com uma certa precisão, as energias associadas ao processo de colisão utilizando apenas os dados geométricos. Posteriormente, aproveitando as energias preditas, um segundo modelo é treinado para predizer valores estáticos, como os parâmetros adimensionais Re e We.

Esta abordagem de predição em duas etapas tem uma importância especial para situações experimentais onde obter parâmetros precisos do fluido é uma tarefa complexa. Muitas vezes, mesmo tendo acesso a dados de vídeos de experimentos, a falta de conhecimento sobre as propriedades do fluido impede a predição precisa do balanço de energia. Em tais situações, o método proposto oferece uma solução robusta ao utilizar dados geométricos para predizer não apenas as energias, mas também os parâmetros adimensionais, que são essenciais para a compreensão da física relacionada à dinâmica.

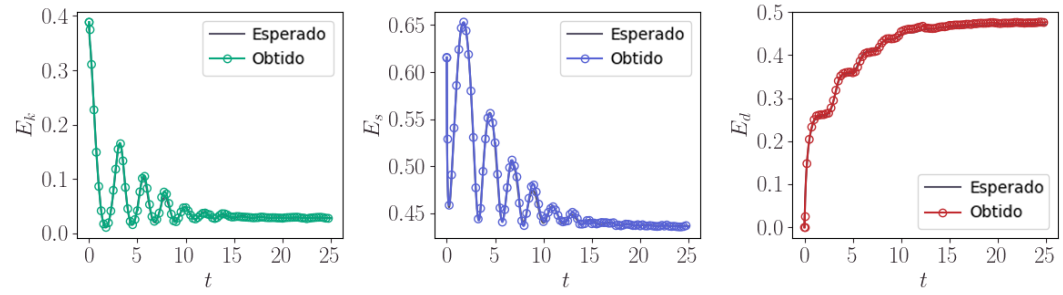
Em resumo, o processo envolve duas redes neurais sequenciais separadas. A saída da primeira rede neural, a qual prediz as energias do sistema, serve de entrada para a segunda rede, a qual prediz os números adimensionais, como ilustrado na Figura 50.



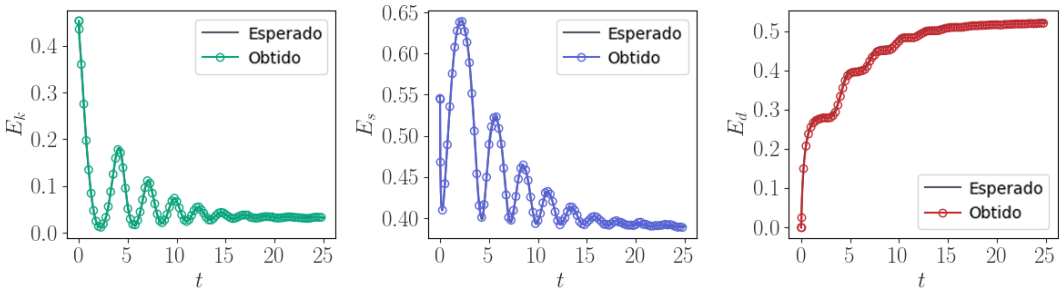
(a) 1ª instância - Re = 115.56, We = 53.0, B = 0.17.



(b) 10ª instância - Re = 115.56, We = 53.0, B = 0.21.



(c) 21ª instância - Re = 74.45, We = 22.0, B = 0.27.



(d) 22ª instância - Re = 85.48, We = 29.0, B = 0.27.

Figura 49 – Comparação entre as energias esperadas e as obtidas em quatro instâncias do dataset de testes da colisão.

8.3.1 Predição do balanço de energia

Para o primeiro passo, é aplicada uma rede neural LSTM para predizer todas as três energias utilizando apenas o diâmetro $D(t)$ e a altura $H(t)$ das gotículas. A Figura 51 ilustra um exemplo de como o comportamento da predição das energias é modificado quando os números

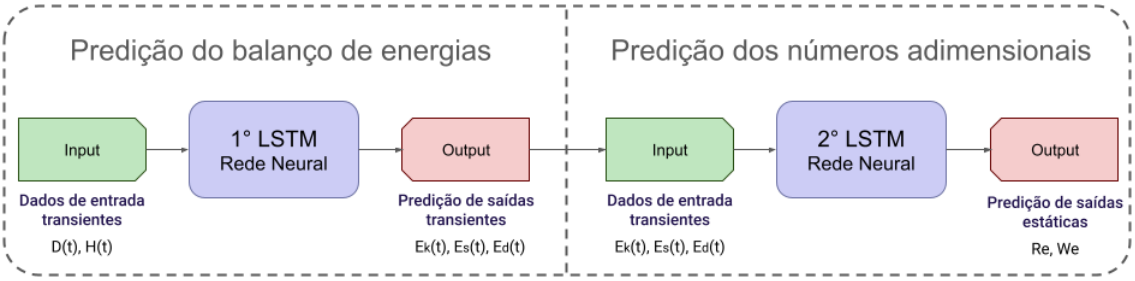
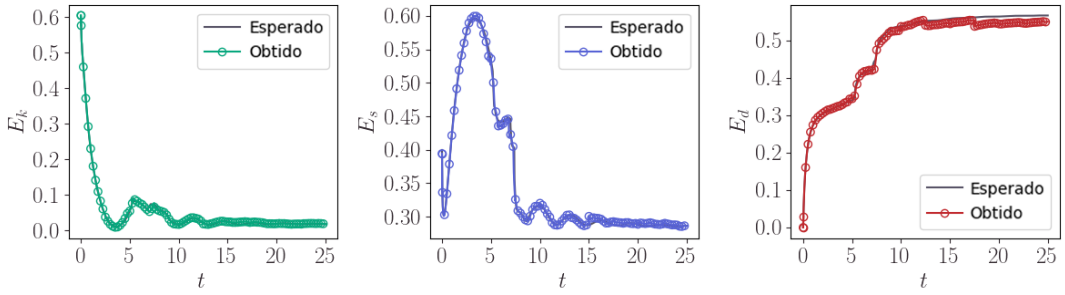
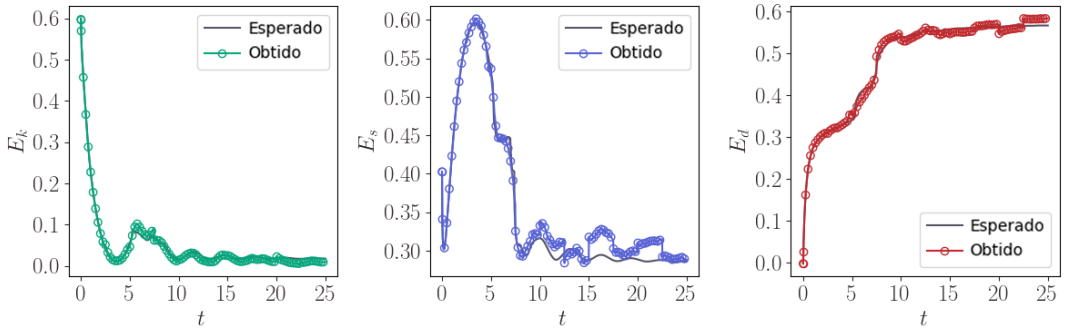


Figura 50 – Esquema de duas fases para a predição das energias e dos números adimensionais.

adimensionais são omitidos como parâmetros de entrada.



(a) Com os números adimensionais.



(b) Sem os números adimensionais.

Figura 51 – Comparação das predições de energias com e sem os números adimensionais como parâmetros de entrada - Re = 115.56, We = 53.0, B = 0.17.

Apesar do modelo apresentar uma diminuição na acurácia quando os números adimensionais são removidos, o comportamento geral das energias ainda consegue ser devidamente capturado. Além disso, dado o contexto da disponibilidade extremamente limitada de dados coletáveis a partir de apenas um vídeo, esse aumento no erro permanece tolerável. Para avaliar as diferenças nas métricas entre as instâncias de treinamento e teste para ambas as configurações, vamos examinar a Tabela 6.

No geral, se os números adimensionais são conhecidos, é recomendável optar pelo modelo que foi treinado com eles inclusos. Porém, ao comparar as métricas, é evidente que utilizar apenas

Métrica	Instâncias de treinamento		Instâncias de teste	
	Com	Sem	Com	Sem
R^2	0.99996	0.99963	0.99604	0.9776
$RMSE$	0.00124	0.00401	0.02732	0.03087
$NRMSE$	0.0019	0.00615	0.02736	0.04725

Tabela 6 – Comparação das métricas de avaliação para os modelos com e sem os números adimensionais Re, We e B como parâmetros de entrada.

dados geométricos continua sendo uma opção viável e com uma acurácia razoável.

8.3.2 Predição dos números adimensionais

Em um teste paralelo, foi realizada uma tentativa de predizer os números adimensionais diretamente utilizando apenas os dados geométricos. Porém, esta abordagem resultou em uma baixa precisão para o dataset de colisão, como mostrado no Apêndice A.

Logo, em vez de usar diretamente $D(t)$ e $H(t)$ para predizer os adimensionais Re e We, a segunda etapa envolve utilizar a rede neural treinada anteriormente para predizer as energias e, em seguida, passá-las como entrada para uma segunda rede neural responsável por predizer os adimensionais.

Dado que cada instância é dividida em sub-instâncias de 100 passos de tempo, o valor de predição selecionado é determinado pela mediana dos valores preditos para cada uma das 10 sub-instâncias. Isso é feito para mitigar potenciais ruídos causados por erros altos resultantes da utilização de energias aproximadas como entradas. Os resultados das predições para o dataset de teste da colisão são mostrados na Figura 52.

Apesar da introdução de ruídos decorrentes do uso das energias aproximadas, os valores preditos ainda estão, no geral, se alinhando com os esperados. Para analisar os resultados quantitativamente, as métricas anteriores são calculadas na Tabela 7, levando em conta tanto o dataset de teste quanto o de treinamento.

Com isso, demonstramos que, em certos regimes, pode ser viável predizer energias e números adimensionais usando apenas dados geométricos extraídos de vídeos de experimentos. Como uma continuação deste estudo, novas arquiteturas poderiam ser desenvolvidas e testadas com datasets adicionais para aprimorar a generalização da rede neural e, com isso, expandir sua

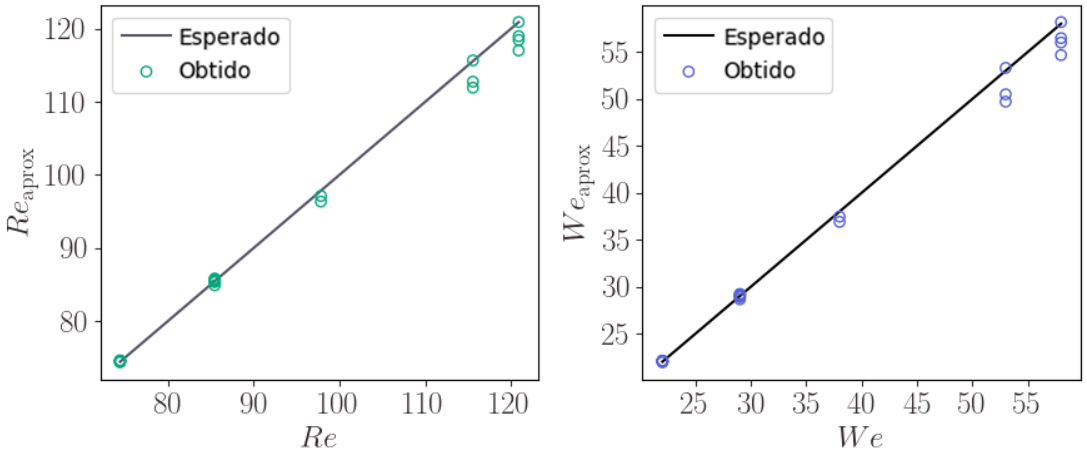


Figura 52 – Predição dos números adimensionais para cada uma das 22 instâncias do dataset de teste da colisão.

Métrica	Instâncias de treinamento	Instâncias de teste
R^2	0.99366	0.99823
$RMSE$	2.50602	1.39455
$NRMSE$	0.02534	0.0141

Tabela 7 – Comparação das métricas de avaliação para a predição dos números adimensionais utilizando as energias preditas na etapa anterior.

aplicabilidade para uma variedade maior de regimes.

9 Conclusões

Inicialmente, nós propusemos dois modelos LSTM para predizer o balanço de energia em dois problemas de superfície livre em movimento: o impacto de uma gotícula com diferentes formatos iniciais em uma superfície sólida e a colisão de duas gotículas de mesmo tamanho e formato. Como o primeiro dataset exigia a geração de dados a partir de formatos diferentes e incomuns, também foi desenvolvido um método capaz de extrair partículas de interfaces presentes em imagens digitais (como mostrado na Seção 4). Após o desenvolvimento dos dois primeiros modelos de teste, uma abordagem de predição sequencial em duas etapas foi aplicada na tentativa de predizer propriedades importantes utilizando exclusivamente dados geométricos. Este método mostra-se especialmente útil para observações experimentais, uma vez que usa apenas dados geométricos, que podem ser facilmente extraídos de imagens ou vídeos de experimentos. Ele fornece uma ferramenta de fácil uso para a desafiadora tarefa de predizer parâmetros de fluidos, como as energias e os números adimensionais.

No contexto dos dois primeiros modelos, com a abordagem de etapa única, ambos compartilharam a mesma arquitetura geral, sendo a única variação os dados utilizados para o treinamento. O primeiro modelo foi treinado usando dados de 95 simulações numéricas envolvendo sete formatos diferentes, extraídos de imagens de experimentos reais, impactando uma superfície sólida e espalhando ao longo do tempo. O segundo modelo utilizou dados de 50 simulações numéricas de duas gotículas idênticas que colidem horizontalmente, coalescem e sofrem oscilações de espalhamento e retração, causando também oscilações no balanço de energia.

Para avaliar a performance de cada modelo, nós conduzimos uma análise abrangente de métricas comumente empregadas na validação de modelos de aprendizado de máquina, como o Coeficiente de Determinação (R^2), a Raiz do Erro Quadrático Médio ($RMSE$) e a Raiz do Erro Quadrático Médio Normalizada($NRMSE$). As métricas R^2, $RMSE$ e $NRMSE$ para ambos os modelos são, respectivamente, 0.99429 e 0.99559, 0.02957 e 0.0137, e 0.02961 e 0.02096. Após analisar as métricas, pudemos observar que ambos os modelos alcançaram performances suficientemente próximas aos valores ideais de cada métrica, com R^2s próximos de um e $RMSE$s se aproximando de zero. Os modelos analisados são precisos de maneira similar mesmo quando aplicados a problemas consideravelmente diferentes, o que pôde ser avaliado pela similaridade da

magnitude de seus $NRMSE$s. O modelo para o espalhamento mostrou resultados promissores ao lidar adequadamente com um formato de gotícula completamente novo. Da mesma forma, o modelo para a colisão teve um bom desempenho para instâncias com conjuntos de parâmetros não utilizados no treinamento. A acurácia dos modelos também foi avaliada através de gráficos que mostram as energias cinética, superficial e de dissipação viscosa em algumas instâncias, confirmando excelente concordância entre os dados da simulação e os valores preditos. Isto sugere que as arquiteturas de aprendizado de máquina definidas têm o potencial de ser altamente adaptáveis e precisas para uma variedade de regimes de fluidos diferentes.

Para as previsões em duas etapas, as arquiteturas permaneceram similares, com a diferença de que agora utilizamos dois modelos conectados para realizar predições sequenciais. Primeiramente, uma rede LSTM prevê o balanço de energia usando apenas dados geométricos. Em seguida, as energias preditas servem como entrada em uma segunda rede LSTM responsável por predizer os números adimensionais estáticos Re e We. As métricas para a primeira etapa (energias) e a segunda (números adimensionais) são, respectivamente, 0.9776 e 0.99823, 0.03087 e 1.39455, e 0.04725 e 0.0141. Apesar de um leve aumento no erro quando comparado à estratégia de predição de uma etapa, as métricas para as predições de duas etapas ainda demonstram alta precisão. Além de uma análise das métricas, foi feita também uma análise visual do comportamento dos números adimensionais preditos. Os resultados mostram boa concordância para valores baixos de Re e We, mas o modelo parece ter maiores dificuldades de estabelecer predições acuradas para valores maiores. Esta dificuldade pode ser atribuída ao fato de que, para os dados de treinamento, há menos instâncias com valores mais altos de Re e We, como visto na Tabela 3. Por fim, justifica-se a aplicação dessa abordagem dado que, ao tentar predizer diretamente os números adimensionais através dos dados geométricos, os $RMSE$s resultantes foram consideravelmente mais altos, como demonstrado no Apêndice A.

Embora as métricas indiquem uma alta versatilidade, ainda há algumas questões a serem exploradas. Uma importante tarefa é o desafio de prever regimes além da coalescência no modelo de colisão, dado que nosso modelo atual depende predominantemente de dados de casos em que há apenas coalescência. Por exemplo, capturar os comportamentos distintos associados ao regime de rebote representa um desafio significativo, já que detectar a transição entre os regimes é uma tarefa complexa para modelos de aprendizado de máquina. Pesquisas futuras podem se aprofundar na busca de novos parâmetros de fluido apropriados para o fenômeno de rebote (ou qualquer outro regime, para ser exato) e explorar a interpretação e a generalização dos modelos.

Esperamos que esta pesquisa venha a destacar as vantagens de aproveitar o desempenho oferecido pelas arquiteturas LSTM para realizar predições de diferentes tipos de dados estáticos ou transientes em escoamentos complexos de fluidos complexos, como, por exemplo, na dinâmica de gotículas viscoelásticas [61, 62].

Referências

1 LIU, Q.; LO, J. H. Y.; LI, Y.; LIU, Y.; ZHAO, J.; XU, L. The role of drop shape in impact and splash. *Nature communications*, Springer Nature, v. 12, p. 3068, 2021. Disponível em: <https://doi.org/10.1038/s41467-021-23138-4>.

2 LOHSE, D. Fundamental fluid dynamics challenges in inkjet printing. *Annual Review of Fluid Mechanics*, Annual Review, v. 54, p. 349–382, 2022. Disponível em: <https://doi.org/10.1146/annurev-fluid-022321-114001>.

3 CHANDRA, S.; AVEDISIAN, C. T. On the collision of a droplet with a solid surface. *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, The Royal Society, v. 432, p. 13–41, 1991. Disponível em: <https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1991.0002>.

4 LAAN, N.; BRUIN, K. G. de; BARTOLO, D.; JOSSERAND, C.; BONN, D. Maximum diameter of impacting liquid droplets. *Phys. Rev. Appl.*, American Physical Society, v. 2, p. 044018, 2014. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevApplied.2.044018>.

5 FUKAI, J.; SHIIBA, Y.; YAMAMOTO, T.; MIYATAKE, O.; POULIKAKOS, D.; MEGARIDIS, C. M.; ZHAO, Z. Wetting effects on the spreading of a liquid droplet colliding with a flat surface: Experiment and modeling. *Physics of Fluids*, AIP Publishing, v. 7, p. 236–247, 1995. Disponível em: <https://doi.org/10.1063/1.868622>.

6 LEE, J. B.; DEROME, D.; GUYER, R.; CARMELIET, J. Modeling the maximum spreading of liquid droplets impacting wetting and nonwetting surfaces. *Langmuir*, ACS Publications, v. 32, p. 1299–1308, 2016. Disponível em: <https://doi.org/10.1021/acs.langmuir.5b04557>.

7 LIN, S.; ZHAO, B.; ZOU, S.; GUO, J.; WEI, Z.; CHEN, L. Impact of viscous droplets on different wettable surfaces: Impact phenomena, the maximum spreading factor, spreading time and post-impact oscillation. *Journal of Colloid and Interface Science*, Elsevier, v. 516, p. 86–97, 2018. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021979717314765>.

8 UKIWE, C.; KWOK, D. Y. On the maximum spreading diameter of impacting droplets on well-prepared solid surfaces. *Langmuir*, ACS Publications, v. 21, p. 666–673, 2005. Disponível em: <https://doi.org/10.1021/la0481288>.

9 PAN, K.-L.; LAW, C. K.; ZHOU, B. Experimental and mechanistic description of merging and bouncing in head-on binary droplet collision. *Journal of Applied Physics*, AIP Publishing, v. 103, p. 064901, 2008. Disponível em: <https://doi.org/10.1063/1.2841055>.

10 ESTRADE, J.-P.; CARENTZ, H.; LAVERGNE, G.; BISCOS, Y. Experimental investigation of dynamic binary collision of ethanol droplets – a model for droplet coalescence and bouncing. *International Journal of Heat and Fluid Flow*, Elsevier, v. 20, p. 486–491, 1999. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0142727X99000363>.

11 QIAN, J.; LAW, C. K. Regimes of coalescence and separation in droplet collision. *Journal of Fluid Mechanics*, Cambridge University Press, v. 331, p. 59–80, 1997. Disponível em: <https://doi.org/10.1017/S0022112096003722>.

12 QIAN, J.; TRYGGVASON, G.; LAW, C.; QIAN, J.; TRYGGVASON, G.; LAW, C. *An experimental and computational study of bouncing and deformation in droplet collision*. American Institute of Aeronautics and Astronautics, 1997. Disponível em: <https://arc.aiaa.org/doi/abs/10.2514/6.1997-129>.

13 WILLIS, K. D.; ORME, M. E. Experiments on the dynamics of droplet collisions in a vacuum. *Experiments in Fluids*, Springer Nature, v. 29, p. 347–358, 2000. Disponível em: <https://doi.org/10.1007/s003489900092>.

14 YARIN, A. L. Drop impact dynamics: Splashing, spreading, receding, bouncing. . . . *Annual Review of Fluid Mechanics*, Annual Reviews, v. 38, p. 159–192, 2006. Disponível em: <https://doi.org/10.1146/annurev.fluid.38.050304.092144>.

15 STONE, H. A. Dynamics of drop deformation and breakup in viscous fluids. *Annual Review of Fluid Mechanics*, Annual Reviews, v. 26, p. 65–102, 1994. Disponível em: <https://doi.org/10.1146/annurev.fl.26.010194.000433>.

16 JOSSERAND, C.; THORODDSEN, S. T. Drop impact on a solid surface. *Annual Review of Fluid Mechanics*, Annual Reviews, v. 48, p. 365–391, 2016. Disponível em: <https://doi.org/10.1146/annurev-fluid-122414-034401>.

17 SCHROLL, R. D.; JOSSERAND, C.; ZALESKI, S.; ZHANG, W. W. Impact of a viscous liquid drop. *Physical Review Letters*, American Physical Society, v. 104, p. 034504, 2010. Disponível em: <https://link.aps.org/doi/10.1103/PhysRevLett.104.034504>.

18 EGGERS, J.; FONTELOS, M. A.; JOSSERAND, C.; ZALESKI, S. Drop dynamics after impact on a solid wall: theory and simulations. *Physics of Fluids*, AIP Publishing, v. 22, p. 062101, 2010. Disponível em: <https://doi.org/10.1063/1.3432498>.

19 LEE, T.; LIU, L. Lattice boltzmann simulations of micron-scale drop impact on dry surfaces. *Journal of Computational Physics*, Elsevier, v. 229, p. 8045–8063, 2010. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999110003761>.

20 LUNKAD, S. F.; BUWA, V. V.; NIGAM, K. Numerical simulations of drop impact and spreading on horizontal and inclined surfaces. *Chemical Engineering Science*, Elsevier, v. 62, p. 7214–7224, 2007. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0009250907005386>.

21 TRYGGVASON, G.; BUNNER, B.; ESMAEELI, A.; JURIC, D.; AL-RAWAHI, N.; TAUBER, W.; HAN, J.; NAS, S.; JAN, Y. A front-tracking method for the computations of multiphase flow. *Journal of Computational Physics*, Elsevier, v. 169, p. 708–759, 2001. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999101967269>.

22 PAN, Y.; SUGA, K. Numerical simulation of binary liquid droplet collision. *Physics of Fluids*, AIP Publishing, v. 17, p. 082105, 2005. Disponível em: <https://doi.org/10.1063/1.2009527>.

23 QI, Y.; LU, J.; SCARDOVELLI, R.; ZALESKI, S.; TRYGGVASON, G. Computing curvature for volume of fluid methods using machine learning. *Journal of Computational Physics*, Elsevier, v. 377, p. 155–161, 1981. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0021999118307046>.

24 ALECSANDER, D. Aproximação numérica de curvaturas de interface: frações de volume e aplicação de redes neurais. *TCC (Graduação) – Ciência da Computação, FCT-Unesp*, 2021.

25 YANG, S.; HOU, Y.; SHANG, Y.; ZHONG, X. Bpnn and cnn-based ai modeling of spreading and icing pattern of a water droplet impact on a supercooled surface. *AIP Advances*, AIP Publishing, v. 12, p. 045209, 2022. Disponível em: <https://doi.org/10.1063/5.0082568>.

26 TEMBELY, M.; VADILLO, D. C.; DOLATABADI, A.; SOUCEMARIANADIN, A. A machine learning approach for predicting the maximum spreading factor of droplets upon impact on surfaces with various wettabilities. *Processes*, MDPI, v. 10, p. 1141, 2022. Disponível em: <https://www.mdpi.com/2227-9717/10/6/1141>.

27 TANG, J.; YU, S.; HOU, X.; WU, T.; LIU, H. Universal model for predicting maximum spreading of drop impact on a smooth surface developed using boosting machine learning models. *Industrial & Engineering Chemistry Research*, ACS Publications, v. 62, p. 15268–15277, 2023. Disponível em: <https://doi.org/10.1021/acs.iecr.3c01560>.

28 AU-YEUNG, L.; TSAI, P. A. Predicting impact outcomes and maximum spreading of drop impact on heated nanostructures using machine learning. *Langmuir*, ACS Publications, v. 39, p. 18327–18341, 2023. Disponível em: <https://doi.org/10.1021/acs.langmuir.3c02405>.

29 HEIDARI, E.; DAEICHIAN, A.; SOBATI, M. A.; MOVAHEDIRAD, S. Prediction of the droplet spreading dynamics on a solid substrate at irregular sampling intervals: Nonlinear auto-regressive exogenous artificial neural network approach (narx-ann). *Chemical Engineering Research and Design*, Elsevier, v. 156, p. 263–272, 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0263876220300460>.

30 CHOUDHURY, R.; NASIR, D. M. T.; KAISER, D. R.; GHIMIRE, R.; ALIYU, D. A.; SOHANI, D. B.; ATANBORI, D. J.; RATHAUR, D. R.; MISHRA, D. R. Uncovering the effect of physical conditions and surface roughness on the maximum spreading factor of impinging droplets using a supervised artificial neural network model. *Industrial & Engineering Chemistry Research*, ACS Publications, v. 62, p. 21208–21221, 2023. Disponível em: <https://doi.org/10.1021/acs.iecr.3c02650>.

31 YEE, J.; IGARASHI, D.; MIYATAKE, S.; TAGAWA, Y. Prediction of the morphological evolution of a splashing drop using an encoder–decoder. *Machine Learning: Science and Technology*, IOP Publishing, v. 4, p. 025002, 2023. Disponível em: <https://dx.doi.org/10.1088/2632-2153/acc727>.

32 YEE, J.; YAMANAKA, A.; TAGAWA, Y. Image features of a splashing drop on a solid surface extracted using a feedforward neural network. *Physics of Fluids*, AIP Publishing, v. 34, p. 013317, 2022. Disponível em: <https://doi.org/10.1063/5.0077050>.

33 YEE, J.; IGARASHI, D.; PRADIPTO; YAMANAKA, A.; TAGAWA, Y. *Correlation between morphological evolution of splashing drop and exerted impact force revealed by interpretation of explainable artificial intelligence*. 2023. Disponível em: <https://doi.org/10.48550/arXiv.2309.10266>.

34 FRANÇA, H. L. Numerical simulation of complex fluid flows with free surfaces. *Dissertação de doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)*, 2022.

35 GAIKWAD, A.; CHANG, T.; GIERA, B.; WATKINS, N.; MUKHERJEE, S.; PASCALL, A.; STOBBE, D.; RAO, P. In-process monitoring and prediction of droplet quality in droplet-on-demand liquid metal jetting additive manufacturing using machine learning. *Journal*

of Intelligent Manufacturing, Springer Nature, v. 33, p. 2093–2117, 2022. Disponível em: <https://doi.org/10.1007/s10845-022-01977-2>.

36 HASSANZADEH, H.; JOSHI, S.; TAGHAVI, S. M. Predicting buoyant jet characteristics: a machine learning approach. *Chemical Product and Process Modeling*, De Gruyter, 2023. Disponível em: <https://doi.org/10.1515/cppm-2023-0026>.

37 ZHANG, X.; JI, T.; XIE, F.; ZHENG, H.; ZHENG, Y. Unsteady flow prediction from sparse measurements by compressed sensing reduced order modeling. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 393, p. 114800, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S004578252200130X>.

38 PASANDIDEH-FARD, M.; QIAO, Y. M.; CHANDRA, S.; MOSTAGHIMI, J. Capillary effects during droplet impact on a solid surface. *Physics of Fluids*, AIP Publishing, v. 8, p. 650–659, 1996. Disponível em: <https://doi.org/10.1063/1.868850>.

39 HADJICONSTANTINOU, N. G. Estimating the maximum splat diameter of a solidifying droplet. *ASME International Mechanical Engineering Congress and Exposition*, American Society of Mechanical Engineers, Heat Transfer: Volume 1, p. 311–316, 1999. Disponível em: <https://doi.org/10.1115/IMECE1999-0998>.

40 LEE, J. B.; DEROME, D.; DOLATABADI, A.; CARMELIET, J. Energy budget of liquid drop impact at maximum spreading: Numerical simulations and experiments. *Langmuir*, ACS Publications, v. 32, p. 1279–1288, 2016. Disponível em: <https://doi.org/10.1021/acs.langmuir.5b03848>.

41 ANTONOV, D.; SHLEGEL, N.; STRIZHAK, P.; TARLET, D.; BELLETTRE, J. Energy analysis of secondary droplet atomization schemes. *International Communications in Heat and Mass Transfer*, Elsevier, v. 117, p. 104666, 2020. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0735193320301949>.

42 ZHANG, Z.; ZHANG, P. Kinetic energy recovery and interface hysteresis of bouncing droplets after inelastic head-on collision. *Physics of Fluids*, AIP Publishing, v. 29, p. 103306, 2017. Disponível em: <https://doi.org/10.1063/1.5000547>.

43 AL-DIRAWI, K. H.; BAYLY, A. E. A new model for the bouncing regime boundary in binary droplet collisions. *Physics of Fluids*, AIP Publishing, v. 31, p. 027105, 2019. Disponível em: <https://doi.org/10.1063/1.5085762>.

44 RAMíREZ-SOTO, O.; SANJAY, V.; LOHSE, D.; PHAM, J. T.; VOLLMER, D. Lifting a sessile oil drop from a superamphiphobic surface with an impacting one. *Science Advances*, Science Advances, v. 34, p. eaba4330, 2020. Disponível em: <https://www.science.org/doi/abs/10.1126/sciadv.aba4330>.

45 HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, MIT Press, v. 9, p. 1735–1780, 1997. Disponível em: <https://doi.org/10.1162/neco.1997.9.8.1735>.

46 CHORIN, A. J.; MARSDEN, J. E. *A mathematical introduction to fluid mechanics*. [S.l.]: Springer-Verlag, 1979. ISBN 9783540904069.

47 GUERMOND, J.; MINEV, P.; SHEN, J. An overview of projection methods for incompressible flows. *Computer Methods in Applied Mechanics and Engineering*, Elsevier, v. 195, p. 6011–6045, 2006. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0045782505004640>.

48 REIS, G. A.; TASSO, I. V. M.; SOUZA, L. F.; CUMINATO, J. A. A compact finite differences exact projection method for the navier–stokes equations on a staggered grid with fourth-order spatial precision. *Computers & Fluids*, Elsevier, v. 118, p. 19–31, 2015. Disponível em: <https://www.sciencedirect.com/science/article/pii/S004579301500198X>.

49 OISHI, C. M.; THOMPSON, R. L.; MARTINS, F. P. Transient motions of elasto-viscoplastic thixotropic materials subjected to an imposed stress field and to stress-based free-surface boundary conditions. *International Journal of Engineering Science*, Elsevier, v. 109, p. 165–201, 2016. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0020722516308175>.

50 FRANÇA, H. L. Um método numérico para o tratamento de mudanças topológicas em escoamentos viscoelásticos com superfície livre. *Dissertação de mestrado em Matemática Aplicada e Computacional FCT-Unesp*, 2018.

51 GLIMM, J.; GROVE, J.; LINDQUIST, B.; MCBRYAN, O. A.; TRYGGVASON, G. The bifurcation of tracked scalar waves. *SIAM Journal on Scientific and Statistical Computing*, SIAM, v. 9, p. 61–79, 1988. Disponível em: <https://doi.org/10.1137/0909006>.

52 GLIMM, J.; GROVE, J. W.; ZHANG, Y. Interface tracking for axissymmetric flows. *SIAM Journal on Scientific Computing*, SIAM, v. 24, p. 208–236, 2002. Disponível em: <https://doi.org/10.1137/S1064827500366690>.

53 TAMIM, S. I.; BOSTWICK, J. B. Oscillations of a soft viscoelastic drop. *npj Microgravity*, Nature Portfolio, v. 7, p. 42, 2021. Disponível em: <https://doi.org/10.1038/s41526-021-00169-1>.

54 SASHITTAL, P.; CHIODI, R.; MORGAN, T. B.; DESJARDINS, O.; HEINDEL, T. J.; BODONY, D. J. Modal analysis and interface tracking of multiphase flows using dynamic mode decomposition. *International Journal of Multiphase Flow*, Elsevier, v. 157, p. 104198, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0301932222001781>.

55 NONATO, L.; MANGIAVACCHI, N.; SOUSA, F. S.; CASTELO, A.; CUMINATO, J. A mass-conserving smooth method. *Mecánica Computacional*, Asociación Argentina de Mecánica Computacional, v. 23, p. 1897–1910, 2004.

56 STAUDEMEYER, R. C.; MORRIS, E. R. Understanding lstm – a tutorial into long short-term memory recurrent neural network. 2019. Disponível em: <https://doi.org/10.48550/arXiv.1909.09586>.

57 JAIN, L. C.; MEDSKER, L. R. *Recurrent Neural Networks: Design and Applications*. 1st. ed. [S.l.]: CRC Press, Inc., 1999. ISBN 0849371813.

58 NOH, S.-H. Analysis of gradient vanishing of rnns and performance comparison. *Information*, MDPI, v. 12, p. 11, 2021. Disponível em: <https://www.mdpi.com/2078-2489/12/11/442>.

59 GROSSE, R. Lecture 15: Exploding and vanishing gradients. *University of Toronto Computer Science*, 2017.

60 OPTUNA - A hyperparameter optimization framework. <https://optuna.org/>. Accessed: 2024-03-07.

61 OISHI, C. M.; THOMPSON, R. L.; MARTINS, F. P. Impact of capillary drops of complex fluids on a solid surface. *Physics of Fluids*, AIP Publishing, v. 31, p. 123109, 2019. Disponível em: <https://doi.org/10.1063/1.5129640>.

62 FRANÇA, H. L.; OISHI, C. M. Numerical investigation of shear-thinning and viscoelastic binary droplet collision. *Journal of Non-Newtonian Fluid Mechanics*, Elsevier, v. 302, p. 104750, 2022. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0377025722000076>.

Apêndices

APÊNDICE A - Predições estáticas dos números adimensionais usando dados geométricos

Para avaliar a viabilidade de predizer os números adimensionais diretamente utilizando apenas dados geométricos, treinamos uma rede LSTM com o dataset de treinamento da colisão. A Figura 53 ilustra os valores preditos para cada instância do conjunto de teste da colisão, enquanto uma comparação das métricas entre o caso de duas etapas e o caso de etapa única (direto) é apresentada na Tabela 8.

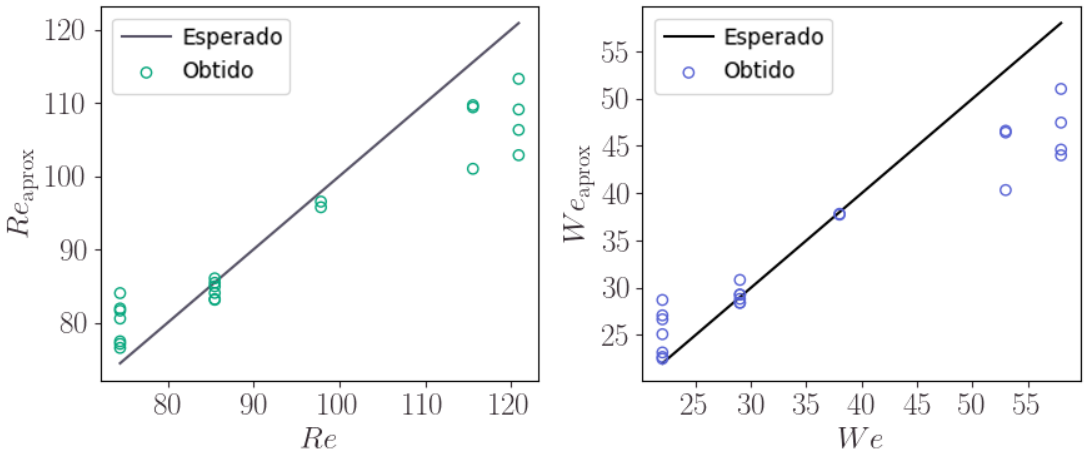


Figura 53 – Predição direta dos números adimensionais para cada uma das 22 instâncias do conjunto de teste da colisão.

Comparando as Figuras 52 e 53 e analisando a diferença nas métricas da Tabela 8, fica evidente que a performance da rede neural para a abordagem de uma etapa, que opera utilizando diretamente os dados geométricos como entrada, é inferior à da estratégia de predição em duas etapas apresentada na Seção 8.3.2. Esta comparação destaca o papel crucial da abordagem de rede neural sequencial de duas etapas na melhoria da precisão e robustez das previsões dos números adimensionais.

Métrica	Duas etapas	Direta
R^2	0.99823	0.95478
$RMSE$	1.39455	7.04277
$NRMSE$	0.0141	0.07122

Tabela 8 – Comparação das métricas de avaliação para as abordagens direta e de duas etapas.