UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

MATHEUS OSVALDO CAMARGO

FERRAMENTA COMPUTACIONAL PARA A CARACTERIZAÇÃO DE SENSORES LMR APLICADOS AO MONITORAMENTO DE ÓLEO DE TRANSFORMADORES ELÉTRICOS

CURITIBA

2025

versão 11.0 (abr.25)

MATHEUS OSVALDO CAMARGO

FERRAMENTA COMPUTACIONAL PARA A CARACTERIZAÇÃO DE SENSORES LMR APLICADOS AO MONITORAMENTO DE ÓLEO DE TRANSFORMADORES ELÉTRICOS

COMPUTATIONAL TOOL FOR THE CHARACTERIZATION OF LMR SENSORS APPLIED TO THE MONITORING OF OIL IN ELECTRICAL TRANSFORMERS

Trabalho de conclusão de curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Prof. Dr. Uilian José Dreyer.
Coorientadora: Eng. Mara Regina Alves de Assis.

CURITIBA

2025

MATHEUS OSVALDO CAMARGO

FERRAMENTA COMPUTACIONAL PARA A CARACTERIZAÇÃO DE SENSORES LMR APLICADOS AO MONITORAMENTO DE ÓLEO DE TRANSFORMADORES ELÉTRICOS

Trabalho de conclusão de curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia Elétrica do curso de Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 01/julho/2025

_____

André Biffe Di Renzo
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Mara Regina Alves de Assis
Mestrado
Universidade Tecnológica Federal do Paraná

_____

Nathalia de Campos Prediger
Doutorado
Universidade Tecnológica Federal do Paraná

_____

Uilian José Dreyer
Doutorado
Universidade Tecnológica Federal do Paraná

CURITIBA

2025

AGRADECIMENTOS

Agradeço aos meus pais, Sidnei e Hilderli, ao meu irmão Hildney, e todos meus amigos e familiares por sempre me apoiarem e oferecerem todo o suporte necessário para enfrentar os desafios que me trouxeram até aqui.

Ao meu orientador, professor Dr. Uilian José Dreyer, pela paciência e por compartilhar comigo seu conhecimento, contribuindo de forma essencial para a concretização deste trabalho.

Também agradeço à minha coorientadora, Eng. Mara Regina Alves de Assis, por dedicar tantas horas de apoio que foram fundamentais durante a realização deste trabalho.

RESUMO

Transformadores de potência isolados a óleo são fundamentais para a distribuição de energia, e a integridade do óleo isolante é crucial para garantir a segurança e a eficiência do sistema. Este trabalho apresenta o desenvolvimento de uma ferramenta computacional para aquisição de dados e caracterização de sensores ópticos baseados em ressonância de modos de perdas (LMR), visando seu uso no monitoramento do óleo isolante de forma mais simples e ágil que os métodos tradicionais. Os sensores LMR são fabricados com a deposição de um filme fino de um óxido metálico sobre fibras ópticas multimodo, após a remoção parcial da cobertura, o que o torna sensível a detecção de variações no índice de refração em volta da região recoberta. Para avaliação desse desempenho de detecção, foram realizados ensaios, através da elaboração de uma bancada experimental, expondo os sensores a diferentes soluções de água e glicerina, com índices variando entre os valores 1,3333 a 1,4755. O processamento dos dados obtidos experimentalmente foi automatizado por um script em Python, que inclui a importação, suavização do sinal, detecção de picos por diferentes métodos (sem ajuste, Gaussiano, Lorentziano e Pseudo-Voigt) e a geração das curvas de calibração. A metodologia foi aplicada a dois sensores LMR que apresentaram comportamentos distintos: enquanto o Sensor A exigiu o uso do método Pseudo-Voigt, obtendo coeficiente de determinação de 0,9915 e sensibilidade de 306,02 nm/RIU, o Sensor B apresentou maior estabilidade e permitiu a calibração direta, alcançando o coeficiente de 0,9991 e sensibilidade de 355,11 nm/RIU. Esses resultados evidenciam a eficácia da metodologia de calibração desenvolvida e reforçam a importância da escolha adequada do sensor, considerando suas características de sensibilidade e estabilidade. Além disso, apontam para a viabilidade de aplicar sensores LMR como refratômetros, desde que submetidos a calibração dentro da faixa de interesse, com potencial uso em sistemas de monitoramento preditivo de transformadores elétricos.

Palavras-chave: Óleo isolante; Transformadores de Potência; Ressonância de modos de perdas; LMR; Calibração; Monitoramento Preditivo.

ABSTRACT

Oil-insulated power transformers are fundamental for energy distribution, and the integrity of the insulating oil is crucial to ensure the safety and efficiency of the system. This work presents the development of a computational tool for data acquisition and characterization of optical sensors based on Lossy Mode Resonance (LMR), aiming at their use in monitoring insulating oil in a simpler and faster way compared to traditional methods. LMR sensors are fabricated by depositing a thin film of a metal oxide onto multimode optical fibers, after partial removal of the cladding, making them sensitive to variations in the refractive index around the coated region. To evaluate their detection performance, experiments were conducted using a custom-built test bench, exposing the sensors to different water and glycerin solutions, with refractive indices ranging from 1.3333 to 1.4755. The processing of the experimental data was automated through a Python script, which includes data import, signal smoothing, peak detection using different methods (no adjustment, Gaussian, Lorentzian, and Pseudo-Voigt), and generation of calibration curves. The methodology was applied to two LMR sensors that exhibited distinct behaviors: while Sensor A required the use of the Pseudo-Voigt method, achieving a determination coefficient of 0.9915 and a sensitivity of 306.02 nm/RIU, Sensor B showed greater stability and allowed direct calibration, reaching a coefficient of 0.9991 and a sensitivity of 355.11 nm/RIU. These results highlight the effectiveness of the proposed calibration methodology and reinforce the importance of proper sensor selection, considering their sensitivity and stability characteristics. Furthermore, they demonstrate the feasibility of applying LMR sensors as refractometers, provided they are calibrated within the target range, with potential use in predictive monitoring systems for power transformers.

Keywords: Insulating Oil; Power Transformers; Lossy Mode Resonance; LMR; Calibration; Predictive Monitoring.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Tema

Durante a segunda revolução industrial, ocorreu o desenvolvimento e aprimoramento de várias tecnologias, sendo a eletricidade uma das mais notáveis. Seu uso generalizado em residências, comércios e indústrias tornou-se indispensável na vida das pessoas e impulsionou a criação de diversas tecnologias dependentes da eletricidade (BECHARA, 2010).

Como resultado, a produção de energia precisou acompanhar o constante aumento de demanda, o que tornou o sistema elétrico altamente dependente de uma infraestrutura confiável. Nesse contexto, os transformadores de potência desempenham um papel fundamental, viabilizando a transmissão e distribuição da eletricidade (CAVACO, 2008). No entanto, falhas nesses equipamentos podem causar prejuízos significativos e comprometer a segurança dos envolvidos (BENETTI, 2016).

Um dos fatores críticos para o funcionamento dos transformadores é a qualidade do óleo isolante, que garante a rigidez dielétrica e a dissipação térmica. Seu desgaste pode indicar falhas incipientes no sistema, exigindo monitoramento periódico para evitar interrupções indesejadas (GODINHO, 2014; ABNT, 2007). Tradicionalmente, essas análises são realizadas por meio de ensaios laboratoriais, como ensaios físico-químicos e a cromatografia gasosa, que, embora eficazes, são demorados e custosos (CARDOSO, 2005).

Como alternativa, sensores ópticos baseados em fibra óptica vêm ganhando destaque em estudos para o monitoramento do óleo isolante. Tal fato, ocorre devido as características inerentes a esses sensores, como a elevada sensibilidade a variações do índice de refração, imunidade a interferências eletromagnéticas e viabilidade de operação em ambientes adversos (MEITEI *et al.*, 2023; MAHANTA; LASKAR, 2018).

No entanto, para que os sensores ópticos possam fornecer dados confiáveis sobre o estado do óleo isolante, é indispensável que passem por um processo de calibração. A calibração tem como objetivo estabelecer uma relação precisa entre a resposta espectral do sensor e as propriedades físicas do óleo, como o índice de refração. Esse processo, porém, ainda representa um desafio, pois depende de

parâmetros que podem variar significativamente de um sensor para outro (VILLAR *et al.*, 2010; IMAS *et al.*, 2023).

Além disso, a falta de protocolos padronizados e a necessidade de realizar diversos testes com soluções de referência tornam a calibração manual um procedimento trabalhoso, demorado e suscetível a erros humanos (MEITEI *et al.*, 2023). Essas dificuldades indicam a necessidade de soluções que tornem o processo de calibração mais confiável, eficiente e aplicável em contextos reais de operação desses sensores, como por exemplo, em sistemas de monitoramento de transformadores elétricos.

1.1.1 Delimitação do tema

Sensores ópticos baseados em Ressonância de Modos de Perdas (Lossy Mode Resonance – LMR) têm sido investigados como alternativa viável para monitoramento de parâmetros físicos em sistemas elétricos, destacando-se por sua sensibilidade, imunidade eletromagnética e compacidade (MEITEI *et al.*, 2023; DREYER *et al.*, 2018). Contudo, para que esses sensores possam ser aplicados de forma confiável, é indispensável submetê-los a um processo de calibração que estabeleça a relação entre sua resposta espectral e o índice de refração do meio (DREYER *et al.*, 2018). Essa etapa, embora fundamental, ainda representa um desafio devido à variabilidade entre sensores e à ausência de protocolos padronizados, o que justifica a proposta deste trabalho: desenvolver uma ferramenta computacional para auxiliar na calibração de sensores LMR por meio de uma bancada experimental e um script em Python para processar os dados espectrais.

1.2 Problemas e Premissas

A avaliação da condição dos transformadores em operação no Brasil é essencial, especialmente porque muitos já ultrapassaram sua vida útil, aumentando os riscos de falhas e os custos com manutenção (GUIMARÃES, 2022). Esse desafio também se repete em outros países, como na Noruega, onde 15% dos transformadores em operação têm mais de 50 anos, exigindo um planejamento de substituição estimado em 100 milhões de dólares (FOROS; ISTAD, 2020). Identificar e priorizar equipamentos em condições críticas é fundamental para evitar falhas e

otimizar os investimentos diante da complexidade e do custo de renovação do sistema elétrico.

Paralelamente, o processo de calibração dos sensores ópticos LMR, propostos para o monitoramento preditivo desses equipamentos, apresenta desafios relevantes, especialmente quando realizado de forma manual. Por envolver grandes volumes de dados espectrais, a manipulação dessas informações torna-se trabalhosa e propensa a erros (MAHANTA; LASKAR, 2018; IMAS, 2023; MEITEI *et al*., 2023). Antes da automação proposta neste trabalho, os cálculos e análises dependiam de softwares matemáticos que necessitavam de uma preparação prévia dos resultados obtidos experimentalmente, o que aumentava a probabilidade de falhas humanas e comprometia a confiabilidade dos resultados.

Além disso, fatores externos podem influenciar diretamente o processo de calibração, exigindo ajustes constantes para minimizar desvios e garantir a precisão das medições (LIN; CHEN, 2021). A implementação de uma ferramenta de calibração contribui para reduzir essas falhas, aumentando a confiabilidade e a reprodutibilidade dos resultados para os futuros estudos para a aplicação desses sensores no sistema elétrico.

1.3 Objetivos

O objetivo principal deste trabalho é desenvolver uma ferramenta computacional de calibração para os sensores ópticos LMR. Para isso, torna-se necessário elaborar uma bancada experimental para expor os sensores a diferentes índices de refração, possibilitando a aquisição de dados espectrais. A ferramenta desenvolvida deve ser capaz de importar esses dados, identificar os picos de ressonância LMR e, gerar como resultado, curvas de calibração e funções matemáticas que descrevem o comportamento do sensor e permitem a determinação de sua sensibilidade.

1.3.1 Objetivo geral

Desenvolver um método computacional para a aquisição de dados do processo de caracterização dos sensores ópticos LMR, com a finalidade em contribuir para sua aplicação em sistemas de monitoramento de parâmetros físicos em transformadores elétricos.

1.3.2 Objetivos Específicos

Neste trabalho, tem-se por objetivo específico:

- Realizar revisão bibliográfica sobre transformadores de potência e sensores ópticos LMR, abordando seus princípios, aplicações e métodos de manutenção;
- Desenvolver bancada experimental para expor os sensores LMR a diferentes índices de refração, utilizando soluções de água e glicerina em concentrações conhecidas;
- Desenvolver script em Python para automatizar o processamento dos dados espectrais, incluindo a importação, tratamento, detecção de picos de ressonância LMR e geração das curvas de calibração;
- Aplicar diferentes métodos de detecção de picos para avaliar a precisão e adequação na caracterização espectral dos sensores LMR;
- Avaliar o desempenho e o comportamento de diferentes sensores LMR quanto à curva de calibração gerada e a sensibilidade determinada;
- Apresentar os resultados e as possíveis contribuições do processo de calibração desenvolvido para futuros estudos com os sensores LMR;

1.4 Justificativa

Sensores ópticos, como os do tipo LMR, precisam passar por processos de calibração que validem seu funcionamento e garantam medições confiáveis. Com o aumento do interesse por esses sensores em aplicações industriais e no setor elétrico, este trabalho traz como justificativa a resposta às dificuldades encontradas na calibração manual, que exige grande volume de dados, tempo e é suscetível a erros (MAHANTA; LASKAR, 2018; MEITEI *et al.*, 2023).

Neste contexto, a ferramenta computacional proposta neste trabalho busca reduzir a necessidade de manipulação prévia dos dados obtidos experimentalmente, tornando o processo de calibração mais reprodutível e ágil, mesmo em condições menos ideais.

Dessa forma, o trabalho contribui para o avanço de metodologias experimentais no campo da instrumentação óptica, com potencial aplicação no monitoramento das condições do óleo isolante em transformadores elétricos.

1.5 Metodologia da pesquisa

A pesquisa foi conduzida em etapas sequenciais, iniciando com uma revisão bibliográfica sobre transformadores de potência e métodos de manutenção, seguida do estudo dos princípios de funcionamento dos sensores ópticos LMR, com foco em sua teoria e aplicações.

Em seguida, foi elaborada uma bancada experimental para a calibração dos sensores, tomando como referência o método de calibração proposto por Dreyer *et al*. (2018) no artigo "*Gas Detection Using LMR-Based Optical Fiber Sensors"*. A bancada foi composta por uma fonte de luz halógena (AQ4305), fibras ópticas multimodo recobertas com filme fino de SnO_2 e um espectrômetro QE65000 da *Ocean Optics*®, permitindo a coleta dos espectros transmitidos pelos sensores imersos em soluções de água e glicerina com diferentes índices de refração.

Com os dados espectrais obtidos, foi desenvolvido um script em Python para automatizar o processamento. O script realiza a importação dos espectros, aplica filtros de suavização, identifica a região espectral de interesse e detecta os picos de ressonância por diferentes métodos: sem filtro, ajuste Gaussiano, Lorentziano e Pseudo-Voigt. Essa abordagem permitiu analisar os sensores mesmo em espectros mais ruidosos, possibilitando a obtenção de curvas de calibração, ainda que com perda de precisão. A partir dos deslocamentos espectrais, foram geradas curvas de calibração e funções matemáticas que descrevem o comportamento e a sensibilidade dos sensores frente às variações do índice de refração.

Por fim, dois sensores LMR foram comparados quanto ao desempenho da metodologia desenvolvida, considerando critérios como sensibilidade e o coeficiente de determinação das curvas geradas. Os resultados obtidos fornecem subsídios para futuras aplicações desses sensores no monitoramento preditivo do óleo isolante de transformadores de potência.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Transformador a óleo isolante

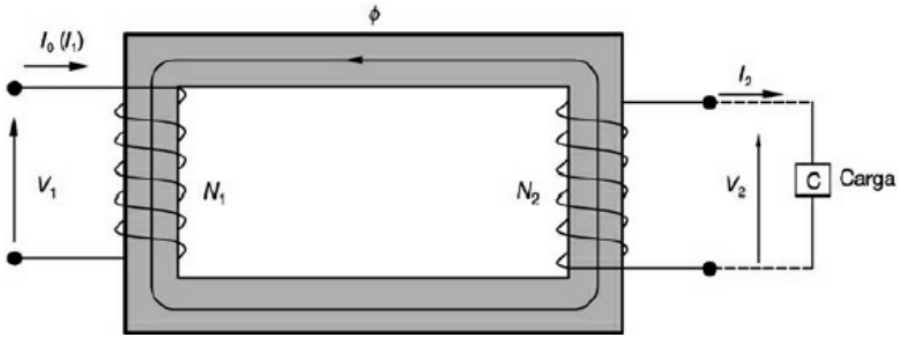
De acordo com a NBR 5356 (2007), um transformador de potência é um equipamento estático, de construção simples e alto rendimento, composto por dois ou mais enrolamentos, conhecidos como primário e secundário. Este equipamento, por meio da indução eletromagnética, é capaz de transformar um sistema de tensão e corrente alternadas em outro sistema de tensão e corrente, geralmente com valores diferentes, mas à mesma frequência. Seu principal objetivo é transmitir potência elétrica (MAMEDE, 2011).

O princípio básico de funcionamento de um transformador de potência é baseado na lei de indução de Faraday:

$$e_{ind} = \frac{d\lambda}{dt} \qquad \textbf{(2.1)}$$

onde λ é o fluxo eletromagnético concatenado pelo enrolamento que está conectado a uma fonte que induz uma força eletromotriz (e_{ind}) para outro enrolamento conectado a uma carga (CHAPMAN, 2013). A energia que é transferida nesse processo não depende de partes mecânicas, mas sim de um caminho magnético potencializado (Figura 1), chamado de núcleo do transformador (MARTIGNONI, 1983).

Figura 1 - Circuito magnético simplificado de um transformador



Fonte: Mamede (2011, p. 666).

No entanto, a corrente elétrica que passa pelos enrolamentos primário e secundário durante a operação de um transformador, quando em carga, gera uma grande quantidade de calor devido as condições de resistência elétrica dos seus condutores. Outros fatores como perdas por histereses e correntes de Foucault geradas no processo de indução eletromagnética, acabam também, que mesmo a vazio, gerando perdas por calor. Este calor, se não for adequadamente gerenciado,

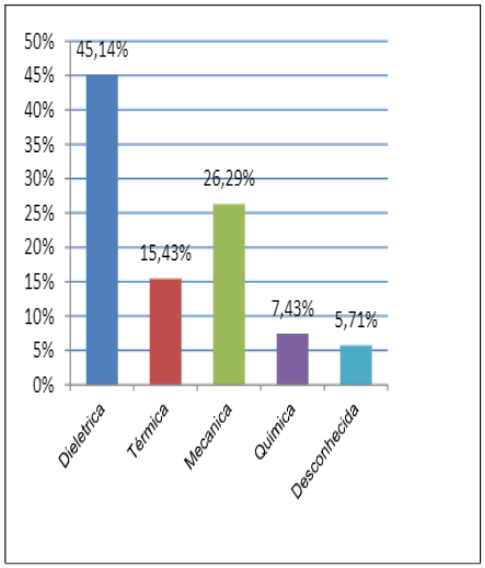
pode levar a uma série de problemas, incluindo a degradação da isolação, o aumento da resistência dos enrolamentos e, em casos extremos, a falha do transformador (MAMEDE, 2011).

Portanto, é essencial ter um sistema eficaz de refrigeração e isolamento em um transformador. O sistema de refrigeração é projetado para dissipar o calor gerado durante a operação do transformador. Isso pode ser realizado através de uma variedade de métodos, incluindo a circulação de ar ou líquido de refrigeração, como o óleo, ao redor dos enrolamentos (MARTIGNONI, 1983).

Nos transformadores isolados a óleo mineral, o óleo atua simultaneamente como meio refrigerante e dielétrico, graças à sua alta capacidade térmica e elevada rigidez dielétrica (MARTIGNONI, 1983). No entanto, sob altas temperaturas, ocorre a degradação do óleo com formação de gases dissolvidos, reduzindo sua eficácia isolante e podendo levar a falhas prematuras (NACARATO, 2018).

Dentro deste contexto, o CIGRE Brasil (Comitê Nacional Brasileiro de Produção e Transmissão de Energia Elétrica) participou de uma pesquisa sobre falhas em transformadores ocorridas no período de 2004 a 2009. Os resultados dessa análise, referentes à natureza das falhas no cenário em questão, estão representados no gráfico da Figura 2, onde se destaca, a predominância das falhas de origem dielétrica, ou seja, problemas de origem correlacionados ao sistema de isolamento (BASTOS *et. al*, 2013).

Figura 2 - Percentual de falha por origem



Fonte: Bastos *et al.* (2013 p. 24)

2.1.1 Análise do estado do óleo isolante

O óleo isolante é fundamental para o bom funcionamento dos transformadores, atuando como meio de isolamento elétrico e refrigeração. No entanto, ao longo do tempo e sob condições adversas, ele sofre degradações que podem comprometer o desempenho do sistema de isolamento. Por isso, é essencial monitorar constantemente seu estado, adotando métodos de diagnóstico capazes de detectar falhas em estágios iniciais, como as análises físico-químicas e a cromatografia gasosa, conforme preconizado por normas técnicas (BARBOSA, 2008).

As análises físico-químicas avaliam diretamente propriedades do óleo, como rigidez dielétrica, teor de água, acidez e viscosidade, permitindo identificar sinais de envelhecimento e contaminação. Esses parâmetros, regulamentados por normas como a ABNT NBR 5356, auxiliam na definição da qualidade do óleo e orientam ações corretivas como secagem, filtragem ou substituição, contribuindo para prolongar a vida útil dos transformadores (PALITÓ, 2019).

Já a Análise de Gases Dissolvidos (AGD) é amplamente utilizada para diagnosticar falhas internas, detectando a presença de gases gerados pela decomposição do óleo, como H_2, CH_4 e C_2H_2. A AGD é realizada via cromatografia gasosa, sendo uma técnica precisa, porém trabalhosa e dependente de mão de obra especializada. Apesar de eficaz, seu custo e complexidade justificam o crescente interesse por alternativas mais práticas e contínuas, como sensores ópticos, foco central deste trabalho (ABNT NBR 7274, 2012; PALITÓ, 2019; SILVA, 2012).

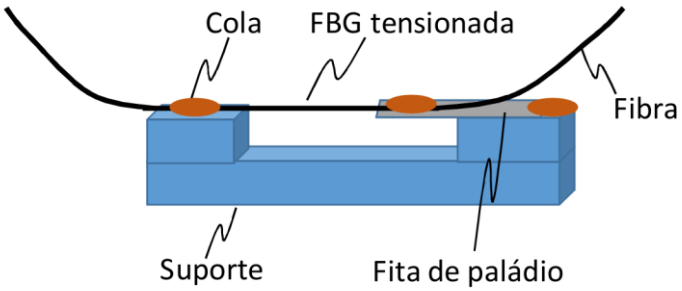
2.2 Trabalhos relacionados

2.2.1 Sensores ópticos e o óleo isolante

Os sensores ópticos são instrumentos que empregam a luz como meio para avaliar características físicas, tais como temperatura, pressão, deformação ou níveis de gases. Eles possuem benefícios quando comparados aos sensores tradicionais, incluindo resistência a interferências eletromagnéticas, alta precisão, leveza e tamanho compacto. Além disso, podem ser integrados às fibras ópticas, o que permite a transmissão de sinais luminosos a grandes distâncias e a realização de medições distribuídas (FONTANA; MARTINS FILHO, 2011; BASSAN *et al*. 2019).

Um exemplo de aplicação desses sensores está no trabalho de Bassan *et al*. (2019), que utiliza a tecnologia de redes de Bragg em fibra óptica (FBG) junto a uma fita de paládio para detectar hidrogênio dissolvido no óleo isolante, como demonstrado na Figura 3. Nesse sistema, a absorção do gás pela fita de paládio causa sua expansão, alterando o índice de refração da fibra e, consequentemente, o comprimento de onda refletido pela FBG. Essa variação é interpretada como um indicativo da presença de hidrogênio, um dos principais gases associados a falhas internas no transformador (BASSAN *et al.,* 2019).

Figura 3 – Esquemático do sensor de hidrogênio de Bassan *et al.*



Fonte: Bassan *et al*. (2019, p. 5)

Além de aplicações específicas, como a de Bassan, estudos mais recentes demonstram que sensores ópticos têm sido amplamente investigados para o monitoramento de diversas propriedades do óleo isolante, incluindo umidade, temperatura, envelhecimento e nível de óleo. A revisão conduzida por Meitei *et al*. (2023) destaca os avanços recentes na utilização de fibras ópticas em sensores distribuídos e localizados, evidenciando a tendência crescente de substituição de

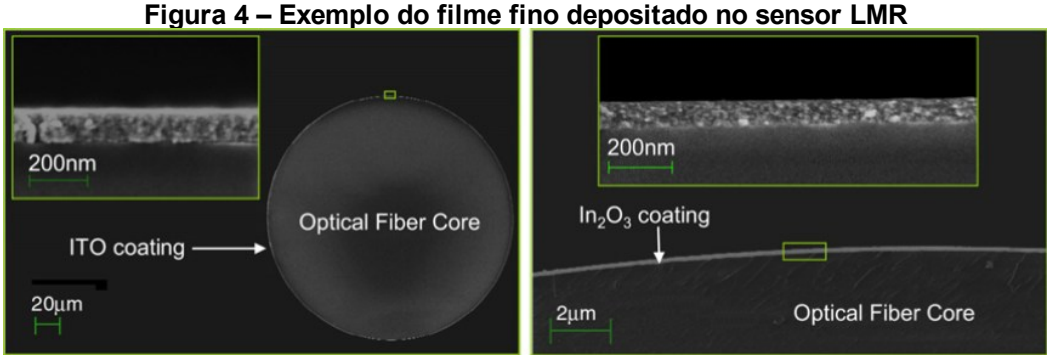
métodos tradicionais, como a análise físico-química e a cromatografia gasosa, por soluções baseadas em tecnologias ópticas, mais práticas e compatíveis com sistemas de monitoramento contínuo (MEITEI *et al.*, 2023).

2.2.2 Sensores ópticos para medição de índice de refração

No contexto atual da tecnologia de sensores, os sensores em fibra ótica baseados em LMR, estão emergindo como uma solução promissora para uma variedade de aplicações de sensoriamento. Tendo que, assim como os sensores ópticos, trazem os mesmos benefícios, como possuir dimensões compactas e baixo impacto a interferências eletromagnéticas (DREYER, 2018).

Os sensores LMR são fabricados através de um processo em que o revestimento é removido de uma pequena porção da fibra ótica, e um filme fino de óxidos metálicos, como SnO_2, In_2O_3 ou ITO, é depositado sobre essa seção exposta, como é mostrado na Figura 4 (DREYER, 2018).

Figura 4 – Exemplo do filme fino depositado no sensor LMR



Fonte: Villar et al. (2010, p. 2)

A resposta espectral desses sensores ocorre em função da interação óptica entre o revestimento e o meio circundante. Alterações no índice de refração ao redor do filme afetam sua permissividade elétrica, provocando um deslocamento na posição do pico de ressonância no espectro de transmissão. Essa propriedade é explorada para detectar variações no índice de refração do meio ao redor da fibra (DREYER, 2018).

Dentre os estudos relevantes que utilizam sensores LMR, destaca-se o trabalho de Dreyer *et al*. (2018), que em seu artigo "*Gas Detection Using LMR-Based Optical Fiber Sensors"* explorou o uso de sensores LMR com filmes de SnO_2 para a detecção de gases como NH_3, NO, CO_2 e O_2. Os resultados mostraram que através

da mudança do índice de refração do meio, o sensor foi capaz de identificar os diferentes gases, sendo o NO o gás com maior sensibilidade e o NH_3 o mais estável (DREYER *et al.*, 2018).

Complementando essa perspectiva, estudos recentes como os de Fuentes *et al*. (2020) e Saini *et al*. (2023) apresentam soluções inovadoras para o desenvolvimento de sensores ópticos voltados à medição do índice de refração. Fuentes *et al*. (2020) apresentaram um sensor planar baseado em LMR, revestido com um filme fino de óxido de cobre (CuO) sobre substrato de vidro. Metade dessa camada foi coberta com PDMS (polidimetilsiloxano) para permitir a medição simultânea de índice de refração e temperatura, separando claramente as contribuições térmicas e ópticas no espectro de ressonância (FUENTES *et al*., 2020). Por outro lado, Saini *et al*. (2023), no trabalho "*Analysis of graphene coated optical fiber for visible range refractive index sensing*", investigaram teoricamente um sensor em fibra óptica multimodo revestido com grafeno e uma camada polimérica intermediária de alto índice refrativo, específico para medição de índice de refração na região visível do espectro, alcançando sensibilidade de 300 nm/RIU (SAINI *et al*., 2023). Esses estudos evidenciam o potencial dos sensores ópticos em aplicações que exigem alta sensibilidade e medições específicas ou simultâneas, dependendo das necessidades técnicas e operacionais.

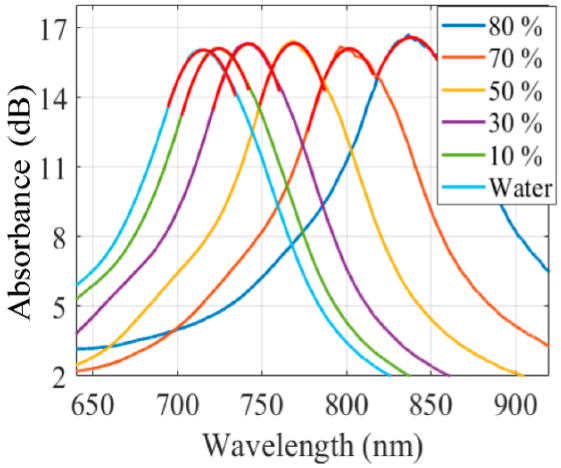
2.2.3 Estratégias de calibração e caracterização de sensores ópticos

Os trabalhos citados anteriormente reforçam consistentemente a necessidade crucial da calibração dos sensores ópticos para assegurar medições confiáveis e precisas, especialmente em dispositivos baseados em ressonâncias espectrais, como o LMR. Um procedimento comum para essa finalidade, empregado por diversos pesquisadores, é a exposição do sensor a uma série de meios líquidos cujos índices de refração são previamente determinados e conhecidos. Essa técnica permite obter pontos de dados que relacionam a resposta medida do sensor aos valores de índice de refração conhecidos, servindo de base tanto para calibração quanto para caracterização (MUKHERJEE *et al*., 2010).

Nesse contexto, Dreyer *et al*. (2018), apesar do foco principal do estudo ser a detecção de gases com sensores LMR, como citado anteriormente, apresentam um método de calibração relevante. A abordagem consistiu em submergir o sensor em soluções aquosas de glicerina com concentrações conhecidas (0% a 80%) e registrar

o deslocamento espectral da ressonância LMR de cada amostra, conforme ilustrado na Figura 5. Esse procedimento permitiu estabelecer a relação entre as variações do índice de refração e o deslocamento do pico da ressonância, validando a sensibilidade óptica do sensor e assegurando sua aplicabilidade para as medições realizadas por Dreyer *et al*. (2018). Além disso, esse procedimento específico foi particularmente relevante como inspiração inicial para o método experimental adotado neste trabalho.

Figura 5 – Espectro de calibração do sensor LMR em diferentes soluções



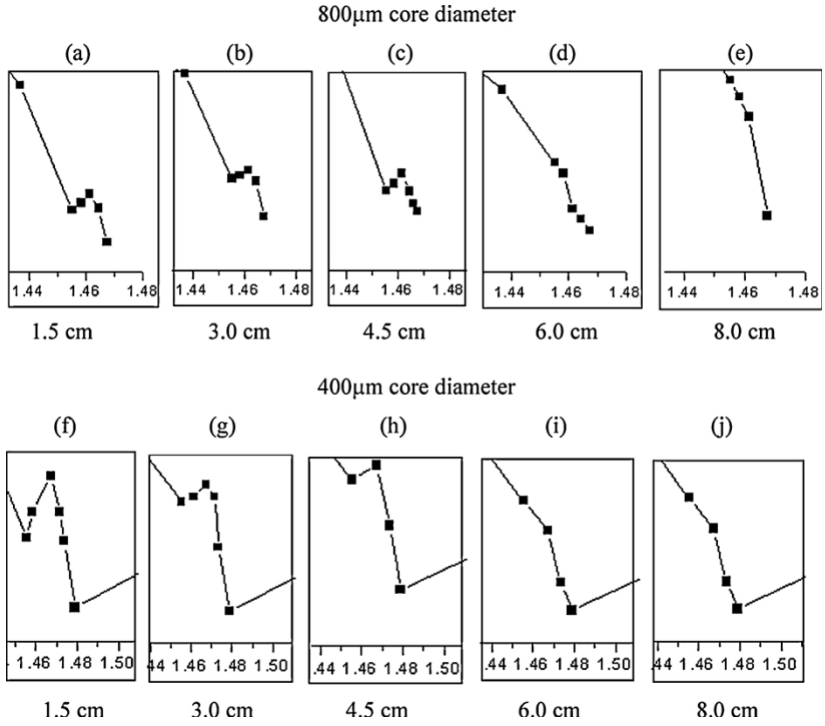
Fonte: Dreyer *et al*. (2018, p. 2)

Uma abordagem tecnicamente semelhante é apresentada no trabalho de Mukherjee *et al*. (2010), intitulado "*Characterization of a fiber optic liquid refractive index sensor*", que propõe uma metodologia experimental para a caracterização de sensores ópticos de índice de refração com líquidos de propriedades óticas previamente conhecidas. Neste estudo, o sensor foi construído a partir de uma fibra óptica parcialmente decapada, permitindo que a luz transmitida interagisse diretamente com o meio externo, sem a existência do revestimento metálico como os sensores LMR. Essa fibra foi então imersa em diferentes soluções, incluindo soluções de sacarose, de glicerol e nitrobenzeno puro, cobrindo uma faixa ampla de índices de refração, desde valores abaixo do índice de refração do revestimento até acima do núcleo da fibra que compõem o sensor.

Em vez de monitorar o deslocamento espectral, Mukherjee *et al*. (2010) adotaram como parâmetro de sensibilidade a razão entre intensidade da luz transmitida pela fibra óptica e uma intensidade de referência. Essa razão foi registrada para quinze líquidos com índices de refração conhecidos, utilizando dois grupos de

cinco sensores, que se diferenciavam pelo diâmetro do núcleo da fibra (400 µm e 800 µm) e por diferentes comprimentos da região sem revestimento (1,5 cm a 1,8 cm). A abordagem permitiu avaliar como a intensidade transmitida varia em função do índice de refração e como o comprimento da região decapada influencia a sensibilidade do sensor, como é observado na Figura 6 (MUKHERJEE *et al.*, 2010).

Figura 6 – Relação da razão "R" das intensidades com os diferentes IR



Fonte: Mukherjee *et al*. (2010, p. 268)

Portanto, apesar de distintas nas estratégias e nos tipos de sensores analisados, as metodologias de Dreyer *et al*. (2018) e Mukherjee *et al*. (2010) compartilham a mesma lógica experimental de imersão dos sensores em líquidos com índice de refração conhecido, reforçando a eficácia dessa abordagem como base para calibração e caracterização de sensores ópticos em diferentes configurações.

2.2.4 Normas para calibração de sensores ópticos

A calibração e caracterização de sensores ópticos devem seguir diretrizes que assegurem rastreabilidade, repetibilidade e confiabilidade dos dados obtidos. No entanto, a ausência de normas específicas para determinados tipos de sensores ainda representa uma barreira significativa à sua aplicação prática, sobretudo em contextos industriais e científicos que demandam precisão e reprodutibilidade. A falta de

padronização dificulta a definição de critérios objetivos para especificar, validar e projetar sensores ópticos, além de comprometer a avaliação de sua estabilidade e desempenho ao longo do tempo (IEEE STANDARDS ASSOCIATION, 2017).

A norma internacional IEC 61757, elaborada pela Comissão Eletrotécnica Internacional (IEC), surgiu com o propósito de estabelecer uma estrutura técnica comum para sensores ópticos em fibra. Sua parte genérica, a IEC 61757-1:2018, define requisitos metrológicos essenciais, como repetibilidade, linearidade, faixa de operação, tempo de resposta e estabilidade em longo prazo. Além disso, apresenta princípios fundamentais, como a identificação da grandeza medida, o detalhamento do princípio de operação, a descrição do sistema de interrogação óptica e o uso de materiais de referência com propriedades conhecidas. A norma também estipula que os ensaios devem ser realizados sob condições ambientais controladas, com repetição sistemática, assegurando a comparabilidade e confiabilidade dos resultados entre diferentes sensores e fabricantes (IEC, 2018).

Portanto, o principal objetivo da IEC 61757 é padronizar a caracterização das propriedades metrológicas (incluindo a sensibilidade) e o desempenho dos sensores ópticos, estabelecendo seus limites operacionais por meio de testes controlados. Embora não mencione explicitamente o termo "calibração", a norma adota princípios alinhados a esse processo, como a definição de curvas de resposta precisas e o uso de padrões confiáveis para comparação.

Além disso, iniciativas paralelas como o documento SEAFOM-MSP-01, promovido pelo consórcio SEAFOM (“*Subsea Fiber Optic Monitoring Group*”), estabelecem critérios específicos para sensores distribuídos, como repetibilidade de temperatura e estabilidade de sinal, complementando as diretrizes da IEC (IEEE STANDARDS ASSOCIATION, 2017).

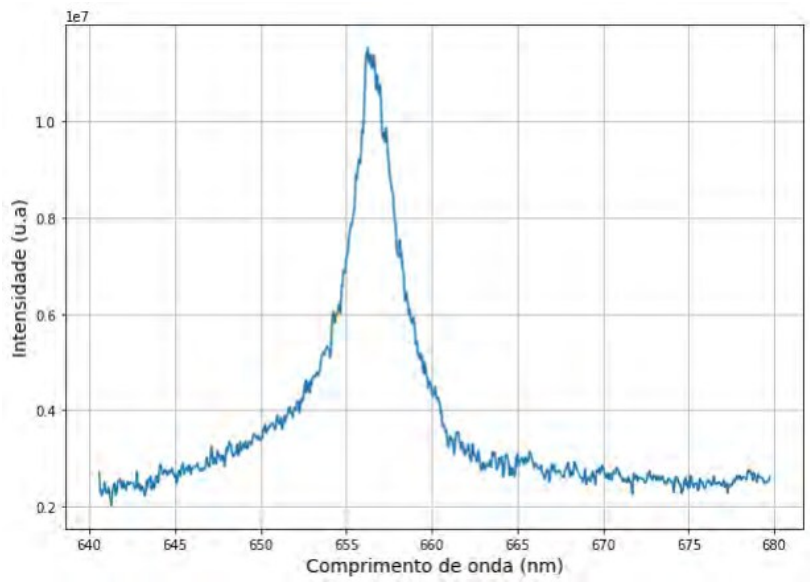
A norma IEC 61757 também é estruturada em subseções específicas conforme o tipo de sensor. Um exemplo é a IEC 61757-1-1, que trata de sensores de deformação baseados em FBG, detalhando metodologias de medição e critérios técnicos associados (IEEE STANDARDS ASSOCIATION, 2017).

Assim, embora ainda não existam normas específicas para sensores baseados em LMR, como os utilizados neste trabalho, a estrutura conceitual da IEC 61757 fornece uma base técnica robusta e alinhada com as boas práticas internacionais, sendo aplicável como referência para o desenvolvimento e a calibração de novos dispositivos ópticos.

2.2.5 Técnicas de detecção de picos em sinais espectrais

A identificação de picos em sinais espectrais é uma etapa fundamental na caracterização do comportamento óptico de sensores LMR. A posição do pico espectral está diretamente relacionada ao índice de refração do meio externo, e a precisão na sua determinação impacta diretamente a sensibilidade e a confiabilidade do sensor. Entretanto, os espectros obtidos experimentalmente (Figura 7) podem apresentar ruídos, flutuações de base e, muitas vezes, picos assimétricos o que exige a adoção de métodos computacionais eficazes para o ajuste da posição desses picos (MORAES *et al.*, 2022).

Figura 7 – Exemplo de espectro com ruídos gerado experimentalmente



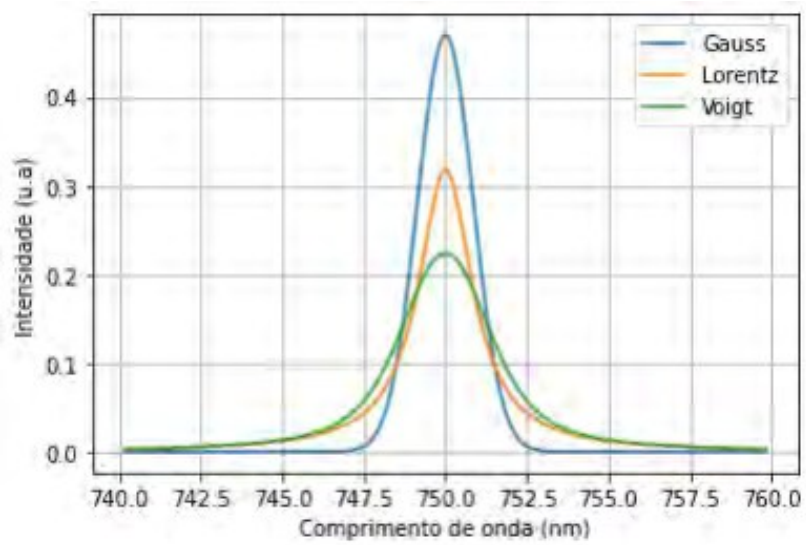
Fonte: Moraes *et al*. (2022, p. 6)

Em trabalhos recentes, têm sido utilizadas funções analíticas como a Gaussiana, a Lorentziana e a função de Voigt para o ajuste de picos. Sendo a Voigt, a função definida como a convolução entre uma Gaussiana e uma Lorentziana, amplamente reconhecida por sua capacidade de representar perfis espectrais reais mais complexos, especialmente quando o pico apresenta uma combinação de fatores que afetam sua forma e largura, como variações no ambiente do sensor ou sobreposição de múltiplos efeitos físicos (JAIN *et al.*, 2018; MORAES *et al.*, 2022).

As funções Gaussianas e Lorentzianas são frequentemente empregadas por sua simplicidade e capacidade de representar, respectivamente, picos com caudas suaves ou mais acentuadas. A escolha entre uma ou outra depende da forma do pico

observado experimentalmente, sendo a Gaussiana mais indicada para sinais com distribuição simétrica e a Lorentziana para perfis mais estreitos com maior concentração no centro. No entanto, em medições experimentais, é comum que os picos apresentem deformações ou características que não se ajustam perfeitamente a esses modelos puros. Para lidar com essa limitação, alguns estudos optam por utilizar a função de Voigt, que combina as duas distribuições para representar formas mais complexas (JAIN *et al.*, 2018; MORAES *et al.*, 2022). A Figura 8 ilustra a diferença no comportamento das funções, evidenciando como cada modelo se ajusta de forma distinta ao perfil espectral em torno de um mesmo pico.

Figura 8 – Comparação entre os perfis Gaussiano, Lorentziano e Voigt

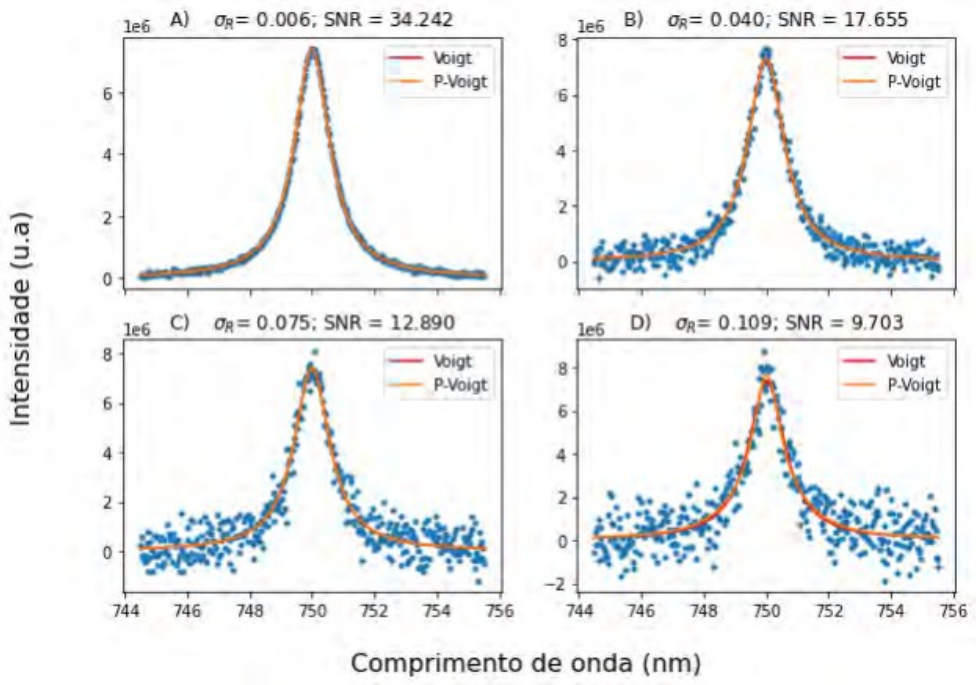


Fonte: Moraes *et al*. (2022, p. 2)

Contudo, devido ao custo computacional associado à complexidade da implementação da função Voigt, uma alternativa adotada é utilizar a função pseudo-Voigt, uma aproximação da função original, baseada em uma combinação ponderada das formas Gaussiana e Lorentziana. Esse modelo se mostra vantajoso por mais simples de calcular e suficientemente preciso para aplicações práticas. Moraes *et al*. (2022) demonstram que esse modelo oferece resultados robustos mesmo em espectros com ruído, possibilitando uma estimativa mais confiável da posição e largura dos picos. A Figura 9 ilustra esse comportamento ao comparar o ajuste das funções Voigt e pseudo-Voigt sob diferentes níveis de ruído. Observa-se que, mesmo com degradação progressiva do sinal, a pseudo-Voigt mantém uma boa concordância com a função Voigt, evidenciando estabilidade e adequação de ambas

as funções mesmo em ambientes experimentais mais ruidosos (MORAES *et al.*, 2022).

Figura 9 – Comparação entre os perfis Voigt e Pseudo-Voigt



Fonte: Moraes *et al*. (2022, p. 2)

Assim, considerando as necessidades identificadas nos trabalhos relacionados, também, destaca-se a importância da implementação de métodos para a detecção da posição dos picos de ressonância LMR, utilizando os mesmos ajustes de curvas baseados em funções como a Gaussiana, Lorentziana e a pseudo-Voigt. Entre elas, a pseudo-Voigt se sobressai pela simplicidade computacional em relação a Voigt original e pela eficiência em lidar com dados experimentais ruidosos. Como a calibração dos sensores depende diretamente da identificação desses picos em espectros reais, é fundamental empregar modelos de ajuste que possibilitem resultados confiáveis mesmo em condições de medição sujeitas a instabilidades.

2.3 Teoria LMR para sensores de fibra óptica

A teoria por trás dos sensores LMR envolve a interação entre ondas evanescentes e modos guiados em filmes finos depositados sobre fibras óticas, permitindo a detecção precisa de mudanças no índice de refração do meio envolvente (PALIWAL; JOHN, 2015).

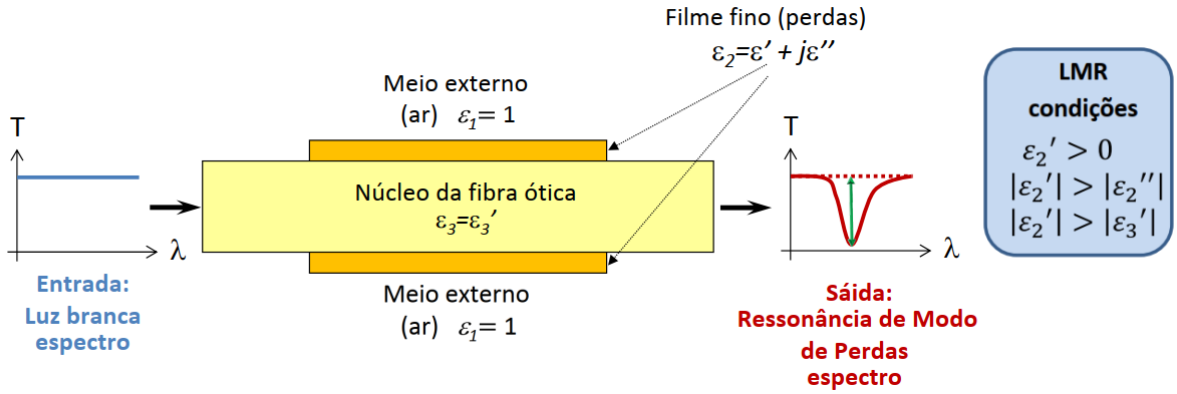
2.3.1 Aplicação da teoria LMR

A Ressonância de Modo de Perda (LMR) é um fenômeno que ocorre quando a luz que é conduzida dentro de uma fibra óptica interage com um revestimento especial na superfície da fibra. À medida que a luz passa pela fibra e encontra esse revestimento, parte dela “escapa” na forma de uma onda evanescente, que pode ser considerada como uma pequena porção de luz que se projeta para fora da fibra (PALIWAL; JOHN, 2015).

Essa onda evanescente pode interagir com o filme, se acoplando com as ondas de luz que podem se propagar no filme fino, um processo denominado como acoplamento de modos. O acoplamento eficiente ocorre quando o índice efetivo da onda evanescente coincide com o índice efetivo dos modos de perdas em comprimentos de onda específicos. Este acoplamento depende da espessura do filme fino, e à medida que a espessura aumenta, modos adicionais são guiados no filme, resultando na redistribuição modal (PALIWAL; JOHN, 2015).

Quando esse acoplamento ocorre, uma parte da luz é absorvida pelo filme fino, perdendo intensidade luminosa em comprimentos de onda específicos. Isso cria um pico ou uma queda no espectro da luz que podemos medir, o qual é ilustrado na Figura 10, fornecendo informações valiosas sobre a interação entre a luz e o filme fino (PALIWAL; JOHN, 2015).

Figura 10 - Processo de absorção da luz por um sensor LMR



Fonte: Adaptado Arregui *et al.* (2014, p. 2)

Os guias de onda, formados por filmes finos de materiais semicondutores ou metálicos, apresentam três possibilidades fundamentais de interação com a luz

propagante: ressonância plasmônica de superfície (SPR), éxciton-polariton de longo alcance e modos com perda (LMR) (PALIWAL; JOHN, 2015).

Essas possibilidades estão diretamente ligadas à permissividade elétrica complexa do filme (ε_2).

$$\varepsilon_2 = \varepsilon' + j\varepsilon'' \quad \textbf{(4.1)}$$

A permissividade elétrica complexa existe devido à capacidade dos dipolos elétricos moleculares de se moverem de acordo com a intensidade do campo elétrico variante no tempo. Sempre existe um atraso ou adiantamento desses dipolos em seu movimento relativo ao campo elétrico aplicado (PALIWAL; JOHN, 2015).

A SPR ocorre quando a parte real da permissividade elétrica do filme fino é negativa e maior em amplitude do que sua parte imaginária e do guia de onda ótico. Este fenômeno é geralmente produzido com metais nobres como ouro e prata (VILLAR *et al.*, 2010).

Por outro lado, o éxciton-polariton de longo alcance ocorre quando a parte real da permissividade do filme fino é próxima de zero, e sua parte imaginária possui grande amplitude (PALIWAL; JOHN, 2015).

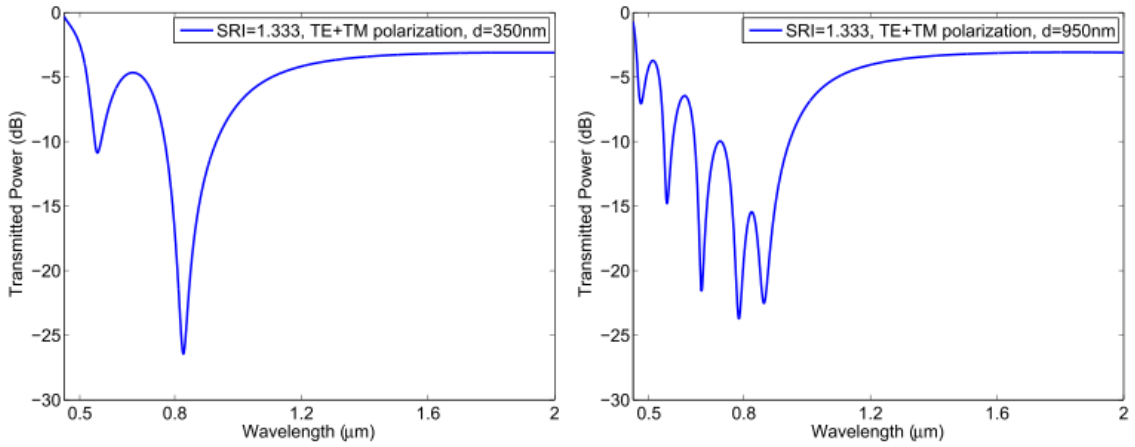
Entretanto, os modos com perda (LMR) são gerados quando a parte real da permissividade é positiva e maior do que sua parte imaginária, e maior em magnitude do que a permissividade da parte real dos materiais que circundam o filme fino, como o guia de ondas e o meio externo. A LMR é gerada pelo acoplamento da luz propagante na fibra ao material semicondutor depositado. A eficiência do acoplamento depende da espessura do revestimento e das condições de propagação das ondas, como sobreposição de modos e casamento de fase (DREYER, 2018). Essas condições para LMR são resumidas na Figura 10, conforme indicado por Arregui *et al.*, (2014).

As LMRs podem ser observadas tanto para polarizações TM (transversal magnética) quanto TE (transversal elétrica). Ao escolher o material, é possível que o LMR para o modo TM caia na mesma faixa espectral que o LMR para o modo TE. Isso elimina a necessidade de um polarizador óptico, simplificando significativamente a fabricação do sensor ou até mesmo o setup experimental dos dispositivos ópticos (ARREGUI *et al.*, 2014).

Além disso, as LMRs possuem a capacidade de ajustar finamente sua posição espectral apenas alterando a espessura do revestimento com perda. Em vez de ter

uma única ressonância óptica, várias ressonâncias podem surgir à medida que a espessura do revestimento com perda aumenta (Figura 11). Todos esses picos podem ser utilizados para sensoriamento ou outras aplicações (ARREGUI *et al.*, 2014).

Figura 11 – Múltiplas ressonâncias LMR com filmes finos de diferentes espessuras (d) para mesmo SRI

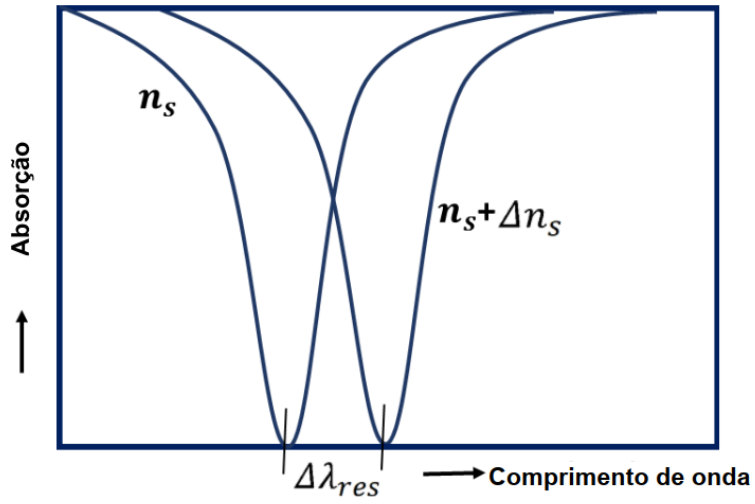


Fonte: Paliwal e John (2015, p. 5)

2.3.2 Sensibilidade do sensor LMR

Um dos métodos empregados para avaliar a sensibilidade dos sensores LMR é o interrogatório de comprimento de onda. Neste método, se mantem constante o ângulo de incidência sendo medido a refletância em função do comprimento de onda. Para isso, se utiliza uma fonte de luz de banda larga para lançar luz e se mede os comprimentos de onda de ressonância do modo com perda (λ_{res}) como o comprimento de onda em que a absorção é mínima (Figura 12) (PALIWAL; JOHN, 2015).

Figura 12 – Calibração do sensor LMR



Fonte: Adaptado Paliwal e John (2015, p. 3)

A sensibilidade do sensor LMR é determinada pela mudança no comprimento de onda de ressonância com a mudança do índice de refração da solução (SRI). Quanto mais significativa a mudança no deslocamento, melhor a sensibilidade do sensor LMR. Matematicamente, a sensibilidade do sensor ao SRI baseado em LMR com interrogatório espectral é dada por:

$$S_{n\lambda} = \frac{\Delta\lambda_{res}}{\Delta n_s} \qquad [\frac{nm}{RIU}] \tag{4.2}$$

Sendo $\frac{nm}{RIU}$ (nanômetros por Unidade de Índice de Refração) a unidade de quantização da sensibilidade do sensor (PALIWAL; JOHN, 2015).

Logo, a precisão de detecção, que é a capacidade do sensor LMR de detectar o comprimento de onda de ressonância com precisão e exatidão, é outro parâmetro de desempenho importante. Uma melhor precisão de detecção leva a uma detecção muito precisa do comprimento de onda de ressonância e, portanto, do SRI. A precisão de detecção depende da largura da curva LMR, quanto mais estreita a largura da curva LMR, melhor a precisão de detecção (PALIWAL; JOHN, 2015).

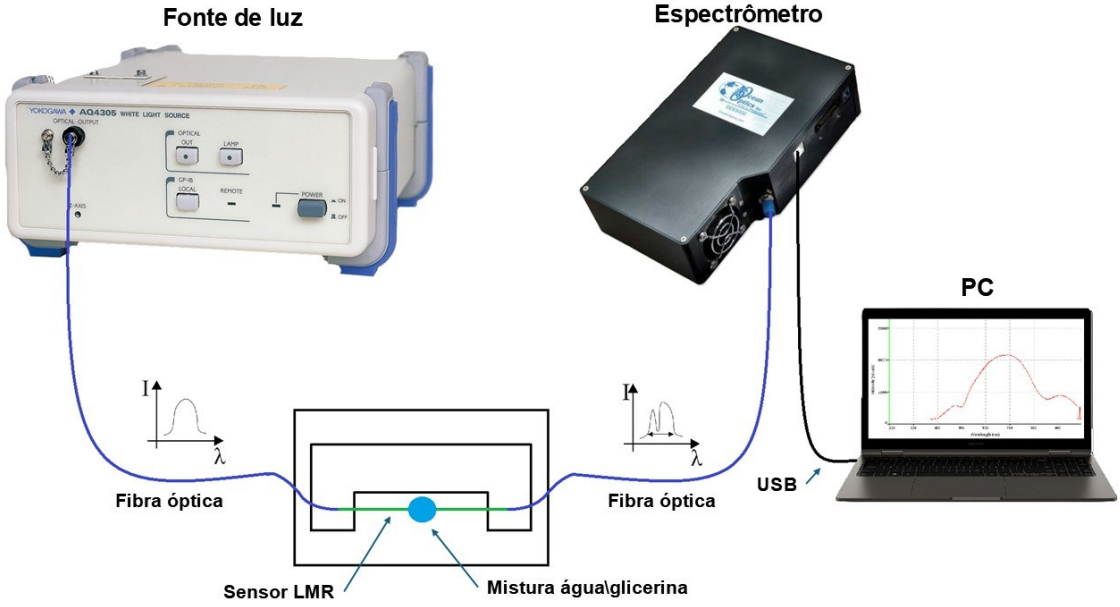
3 METODOLOGIA

Este capítulo apresenta as metodologias utilizadas no processo de caracterização de dois sensores do tipo LMR (Sensor A e Sensor B), com o objetivo de avaliá-los quanto à sua aplicabilidade para o monitoramento das condições de operação do óleo mineral isolante em transformadores. O foco desta etapa está na medição da resposta espectral diante a diferentes valores de SRI, visando à obtenção das curvas de ressonância e de seus respectivos parâmetros, como o comprimento de onda, a largura e a profundidade. Tais parâmetros permitem validar o desempenho dos sensores em termos de sensibilidade e precisão.

3.1 Bancada experimental

Para realizar a caracterização dos sensores, foi necessário realizar um sistema experimental que permitia variar o SRI que entra em contato com a superfície do sensor, e, também, um equipamento que permitia medir o espectro de transmissão ou reflexão dos sensores (DREYER, 2018). O sistema experimental, o qual foi utilizado, para essa finalidade é mostrado na Figura 13.

Figura 13 - Sistema experimental para caracterização do sensor LMR

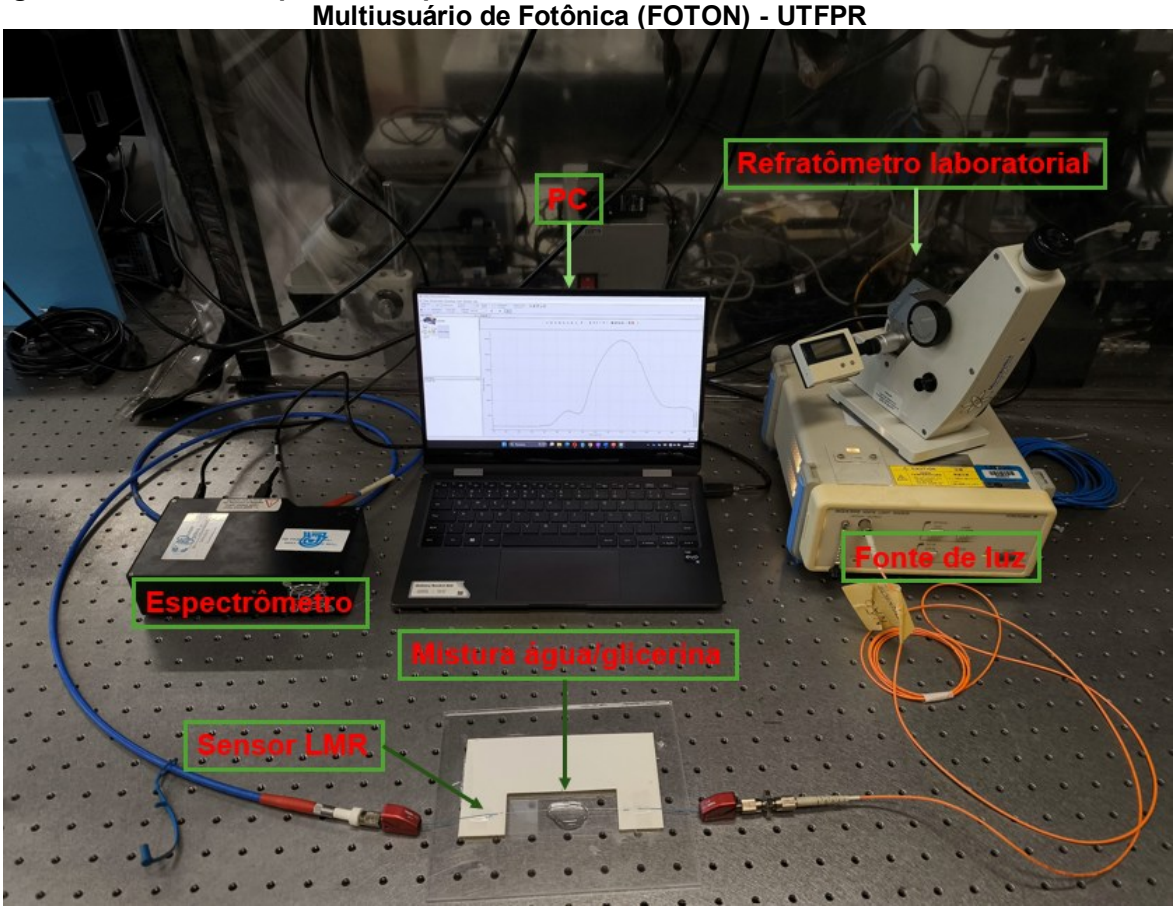


Fonte: Autoria própria (2025).

O sistema experimental, foi composto por uma fonte de luz branca *AQ4305 White Light Source* da *Yokogawa*®, que através de uma lâmpada halógena, emite um

feixe de luz branca pela fibra ótica até o sensor LMR de dióxido de estanho, que está imerso em uma mistura de água e glicerina. Esta configuração permite a variação do SRI, essencial para a caracterização dos sensores. A luz que interage com o sensor é então conduzida pela fibra ótica até o espectrômetro QE65000 da *Ocean Optics®,* que captura o espectro de transmissão. Os dados coletados pelo espectrômetro são então enviados para um computador via conexão *USB*, onde serão processados e exibidos graficamente. Este arranjo experimental não só facilitou a identificação das mudanças no SRI, mas também possibilitou um monitoramento das propriedades ópticas do sensor LMR sob diferentes condições experimentais. Na Figura 14 é representado a disposição real da preparação da bancada experimental no laboratório.

Figura 14 - Bancada experimental para caracterização do sensor LMR montada no Laboratório Multiusuário de Fotônica (FOTON) - UTFPR



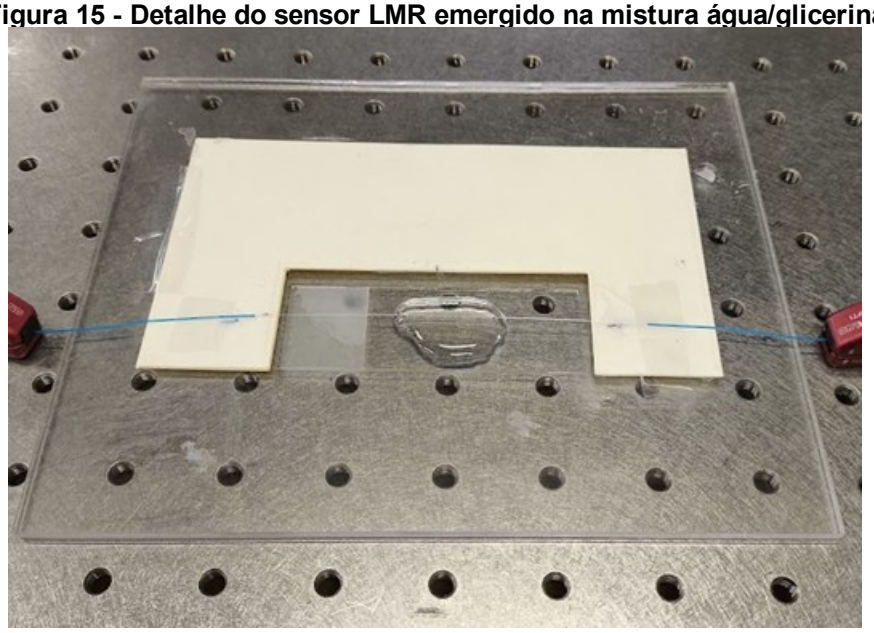
Fonte: Autoria própria (2024)

3.2 Características dos sensores LMR utilizados

Os sensores LMR utilizados neste experimento (Figura 15) são similares aos fabricados por Dreyer (2018) em sua dissertação de doutorado. Os sensores são

baseados em fibras ópticas multimodo com núcleo de 200 µm de diâmetro, com a casca removida em um comprimento de 2 cm, recobertas com um filme fino de SnO_2, depositado utilizando a técnica de pulverização de corrente contínua (CC). O processo de deposição, dura 2 minutos e 15 segundos sob uma pressão de $8x10^{-2}$ mbar e corrente de 90 mA, o que resulta em um filme com espessura de aproximadamente 200 nm. A técnica de pulverização CC envolve a aplicação de uma tensão contínua ao material fonte, liberando íons que se deslocam para o potencial terra, causando a deposição do material na fibra ótica. A câmara de deposição é evacuada para remover gases e gás argônio é injetado para estabilizar a pressão. A deposição pode ser monitorada em tempo real com uma fonte de luz halógena e um espectrômetro, permitindo interromper o processo quando a ressonância desejada é atingida (DREYER, 2018).

Figura 15 - Detalhe do sensor LMR emergido na mistura água/glicerina



Fonte: Autoria própria (2025)

3.3 Preparo das soluções água/glicerina

A Tabela 1 apresenta as diferentes soluções de água e glicerina que foram utilizadas neste experimento, bem como os seus respectivos SRI, medidos à mesma temperatura ambiente com um refratômetro Abbe laboratorial (Figura 14), que possui uma precisão de ±0,0002 na medição do índice de refração (BIOSYSTEMS, 2025). O valor do SRI de cada solução variou de acordo com a quantidade de glicerina, quanto maior era a presença de glicerina na solução maior era o SRI observado. As soluções

foram preparadas misturando-se volumes conhecidos de água e glicerina pura, usando um Becker de 600 ml.

Tabela 1 - Concentrações de água/glicerina usadas no processo de calibração e seus respectivos SRI

Solução	Volume de H_2O (ml)	Volume de glicerina (ml)	Índice de refração da Solução (SRI)
Água/Glicerina 00%	300	0	1,3333
Água/Glicerina 30%	210	90	1,3750
Água/Glicerina 50%	150	150	1,4005
Água/Glicerina 70%	90	210	1,4290
Água/Glicerina 100%	0	300	1,4755

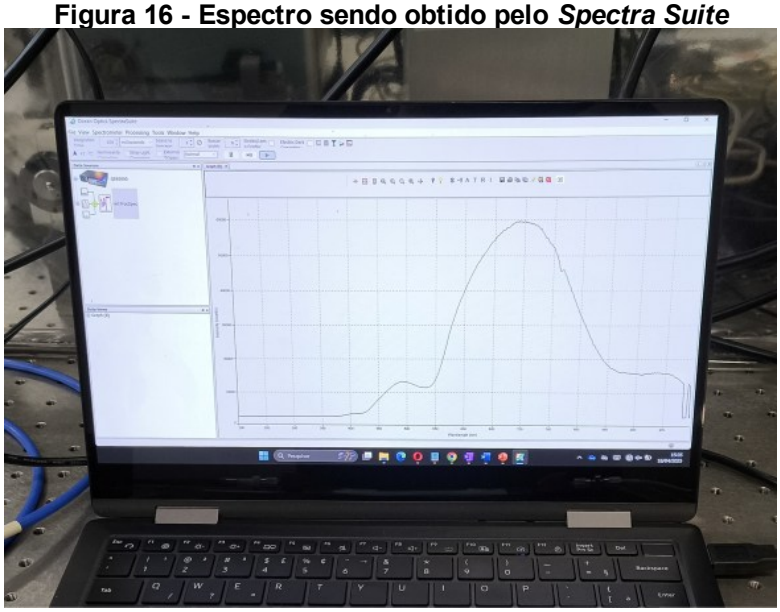
Fonte: Autoria própria (2025)

3.4 Coleta de dados

Para cada solução testada, foi necessário esperar um tempo de estabilização para garantir que os sensores atinjam o equilíbrio com a solução. Após a estabilização, os espectros foram medidos pelo espectrômetro e salvos no computador utilizando o programa *Spectra Suite* (Figura 16), que é o *software* próprio do espectrômetro para obtenção de dados. As medições foram realizadas seguindo a ordem crescente das amostras, começando pela solução com 0% de glicerina até a de 100%. Para cada concentração, foram feitas três medições consecutivas, sempre higienizando o sensor para retornar ao seu estado inicial (sensor limpo e seco) entre uma medição e outra. Esse procedimento totalizou quinze medições para cada sensor, cobrindo toda a faixa de interesse para a aplicação dos sensores.

Os dados coletados foram então salvos no formato de arquivos Excel (*.xlsx*), fornecidos pelo *Spectra Suite*, para as futuras análises dos dados. Sendo que também se mostrou necessário obter e salvar os dados do espectro da luz branca da fonte de luz usada (*AQ4305)*, a fim de possibilitar e facilitar a visualização posterior dos picos de ressonância LMR.

Figura 16 - Espectro sendo obtido pelo *Spectra Suite*

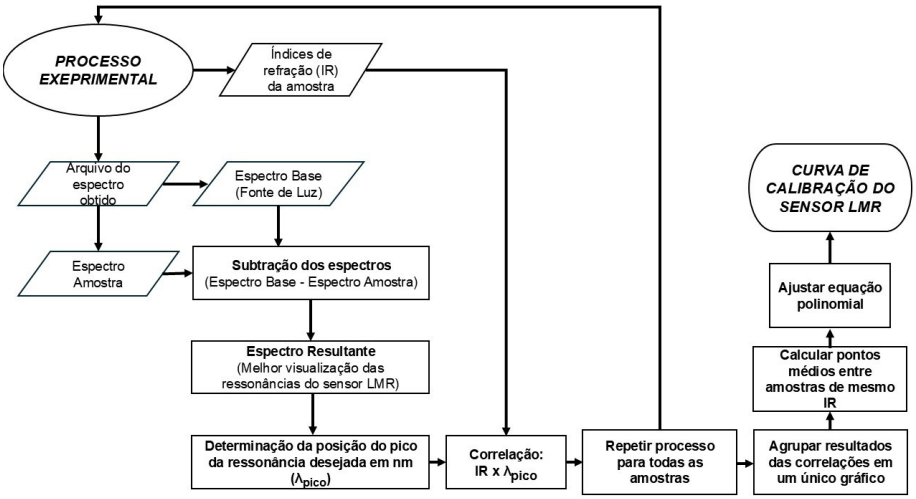


Fonte: Autoria própria (2025)

3.5 Ferramenta computacional

A automatização do pós-tratamento de dados e da caracterização do sensor LMR tornou-se essencial devido às dificuldades do processo manual após a coleta dos dados, que se demonstraram massivos. O método convencional, ilustrado no fluxograma da Figura 17, exige várias etapas, desde a obtenção dos espectros até a criação da curva de calibração. Esses processos são propensos a erros e podem levar muito tempo quando há muitas amostras, prejudicando a eficiência da análise. Para resolver esses problemas, foi criado um script em Python, capaz de otimizar essas etapas e reduzir significativamente o tempo de execução.

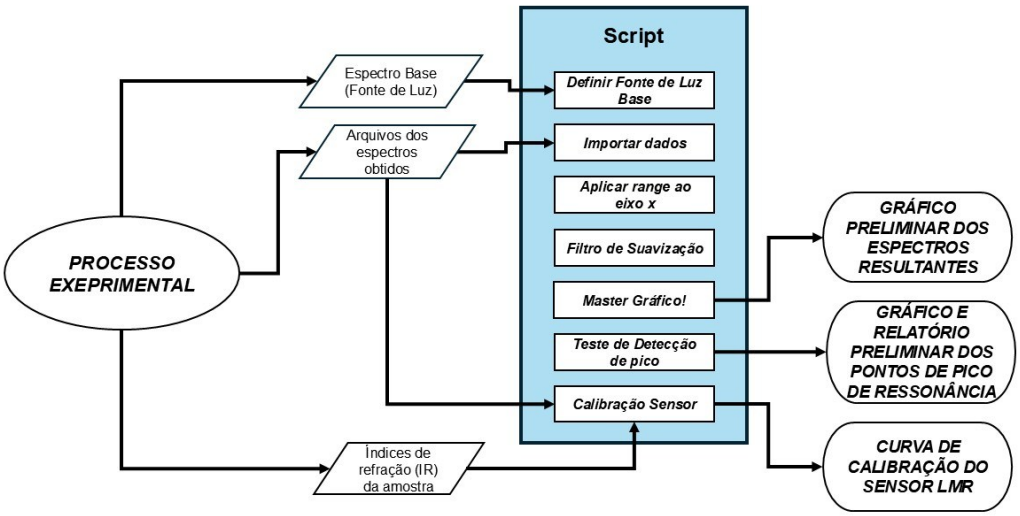
Figura 17 – Fluxograma do processo manual para o pós-processamento dos dados obtidos experimentalmente



Fonte: Autoria própria (2025)

Para mitigar essas dificuldades, foi desenvolvido um código em *Python* capaz de gerenciar e processar os dados obtidos pelo software *Spectra Suite*, como representado no fluxograma da Figura 18. O código foi estruturado de maneira modular, permitindo atualizações e adaptações futuras. Além disso, a interface do usuário (Figura 19) concentra diversas funções essenciais para a manipulação e análise dos espectros. Cada função apresenta instruções claras para orientar o usuário, e o script completo pode ser consultado no Apêndice A -.

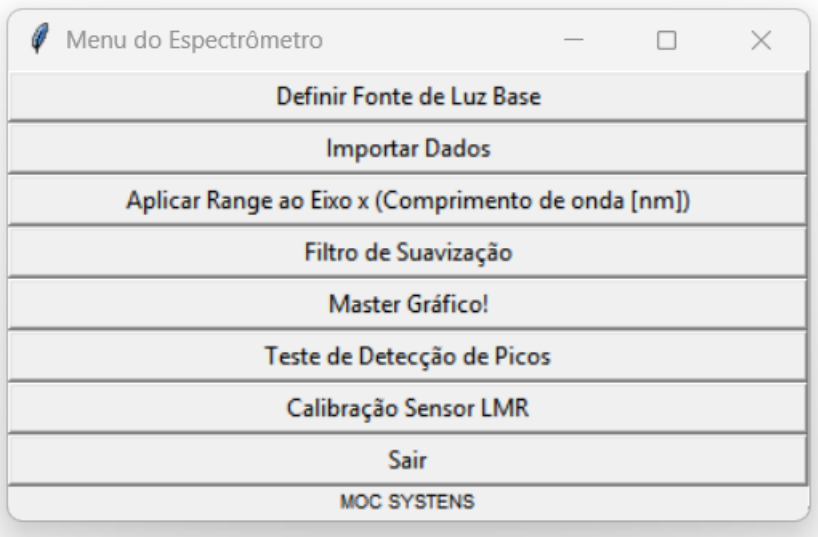
Figura 18 – Fluxograma geral do script



Fonte: Autoria própria (2025)

3.6 Detalhamento das funções do script desenvolvido

Figura 19 - Menu do usuário do script em *Python*



Fonte: Autoria própria (2025)

A seguir, cada função implementada no script é descrita, seguindo a ordem necessária para o fluxo de dados e processamento. Essas funções compõem o menu da interface do usuário e são fundamentais para o pós-processamento dos dados espectrais e suas análises, contribuindo para a caracterização do sensor LMR.

3.6.1 Definir Fonte de Luz Base

A função "Definir Fonte de Luz Base" é essencial para estabelecer um ponto de referência nas análises seguintes. Ela permite que o usuário selecione e carregue para o script um arquivo *.xlsx*, contendo os valores de comprimento de onda (coluna A) e as respectivas intensidades (coluna B), conforme mostrado na Figura 20. Para essa etapa, o arquivo deve conter exclusivamente os dados do espectro da fonte de luz utilizada, sem a presença do sensor LMR.

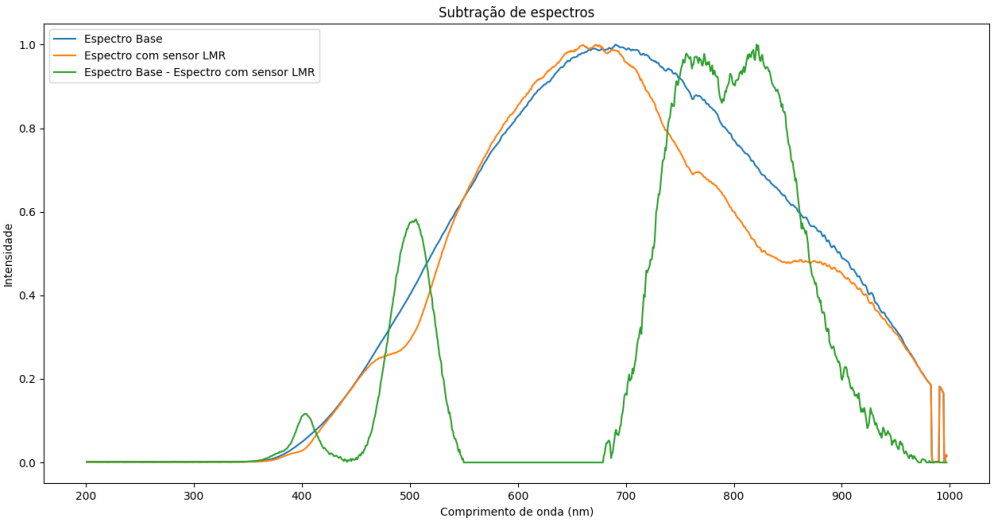
Figura 20 - Exemplo de padrão de dados a serem importados

	A	B
1	199,95	2570,28
2	200,74	2569,95
3	201,54	2570,61
4	202,33	2570,18
5	203,13	2570,7
6	203,92	2569,46
7	204,71	2569,59
8	205,51	2569,9
9	206,3	2569,22
10	207,09	2570,08
11	207,89	2570,2
12	208,68	2570,2
13	209,48	2570,62
14	210,27	2570,76
15	211,06	2571,1
16	211,86	2570,42
17	212,65	2571,39
18	213,44	2572,04
19	214,24	2573,3
20	215,03	2573,9
21	215,82	2574,81
22	216,62	2574,2
23	217,41	2573,88
24	218,21	2572,71
25	219	2572,22

Fonte: Autoria própria (2025)

Esses dados são usados para realizar a diferença entre o espectro base e os dados dos espectros da fibra ótica com o sensor LMR, como é exemplificado na Figura 21 e descrito como parte do processo no fluxograma da Figura 17. Assim, as funções que exibem como resultado os gráficos dos espectros de ressonância LMR mostrarão apenas picos nos comprimentos de onda na região que houver ressonância, ao em vez de vales de absorção no espectro da fonte de luz. Isso facilita a visualização e a análise das próximas funções.

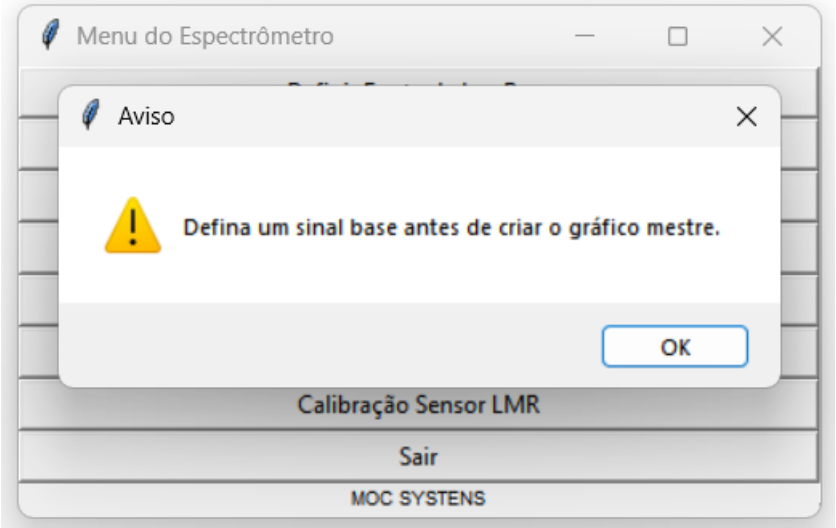
Figura 21 - Exemplo de subtração de espectros para obtenção de picos de ressonância LMR



Fonte: Autoria própria (2025)

Caso o arquivo base não estiver no formato correto ou não houver um sinal predefinido, as funções dependentes desse processo de subtração de espectros exibirão avisos (Figura 22) indicando que não será possível prosseguir com a função desejada e que um sinal base deve ser definido.

Figura 22 - Exemplo de aviso pela falta da definição do sinal base



Fonte: Autoria própria (2025)

3.6.2 Importar Dados

Esta função permite ao usuário importar dados de arquivos *.xlsx* gerados pelo *Spectra Suite* (de acordo com os padrões da Figura 20) diretamente para o banco de dados do script. Assim, importando os dados experimentais das amostras para o ambiente de análise.

Ao executar esta função, o usuário seleciona manualmente os arquivos desejados para importação. Em seguida, o script processa esses arquivos, verifica sua formatação e os salva em uma pasta específica definida pelo próprio código, garantindo que estejam prontos para serem acessados pelas demais funções do script quando necessárias.

Durante o processamento dos arquivos, os valores de intensidades medidos são normalizados, ajustados para uma escala comum (0 a 1), o que permite que haja uma melhor visualização das intensidades obtidas. Este ajuste é feito automaticamente pelo script, que calcula os valores mínimos e máximos dos espectros e aplica a normalização. Todos os dados devem ser normalizados para evitar erros de parametrização e garantir o funcionamento correto do script.

3.6.3 Aplicar Range ao Eixo X (Comprimento de onda [nm])

A função "*Aplicar Range ao Eixo X*" permite que o usuário possa ajustar o valor máximo e mínimo para a exibição do eixo x, que representa os comprimentos de onda em nanômetros. Essa funcionalidade é especialmente útil para focar em uma faixa específica do espectro, como a região onde estão os picos de ressonância do sensor LMR. Além de oferecer maior controle visual sobre os dados, facilita a análise ao eliminar partes irrelevantes do espectro ou áreas com ruídos desnecessários.

Essa configuração está diretamente relacionada à geração do "Master Gráfico!", que será mais bem comentada nos tópicos seguintes, já que o *range* definido pelo usuário determina qual porção do espectro será exibida. Isso é particularmente importante ao comparar espectros de diferentes amostras, garantindo que apenas a região de interesse seja analisada.

Se o usuário não definir manualmente o *range*, o script ajusta automaticamente o eixo x para exibir todo o espectro disponível nos dados importados. Essa funcionalidade padrão assegura a continuidade do fluxo de trabalho mesmo quando a personalização não for realizada.

3.6.4 Filtro de Suavização

O filtro de suavização do script usa o método Savitzky-Golay (S-G) para reduzir ruídos em sinais sem comprometer características importantes do espectro, como a forma e altura dos picos. O método Savitzky-Golay suaviza os dados dividindo

o sinal em pequenas janelas móveis. Em cada janela, ajusta-se um polinômio de baixa ordem que representa os dados e substitui o ponto central da janela pelo valor calculado a partir do polinômio, criando um valor suavizado. Esse processo é repetido ao longo de todo o sinal, garantindo a redução do ruído enquanto preserva picos e tendências essenciais (SCHAFER, 2011).

No script a função principal responsável pela suavização é a "suavizar_grafico". Nela, as intensidades de cada coluna de intensidade dos espectros importados são processadas usando a função "savgol_filter" (Figura 23), da biblioteca "*scipy*" do *Python*, ajustando os parâmetros do filtro conforme a necessidade. Inicialmente, são definidos o tamanho da janela de suavização (50 pontos) e a ordem do polinômio (4º), que controlam o nível de suavização (SCIPY, 2025).

Figura 23 – Função "savgol_filter"

```
savgol_filter(col_y, window_length=50, polyorder=4)
```

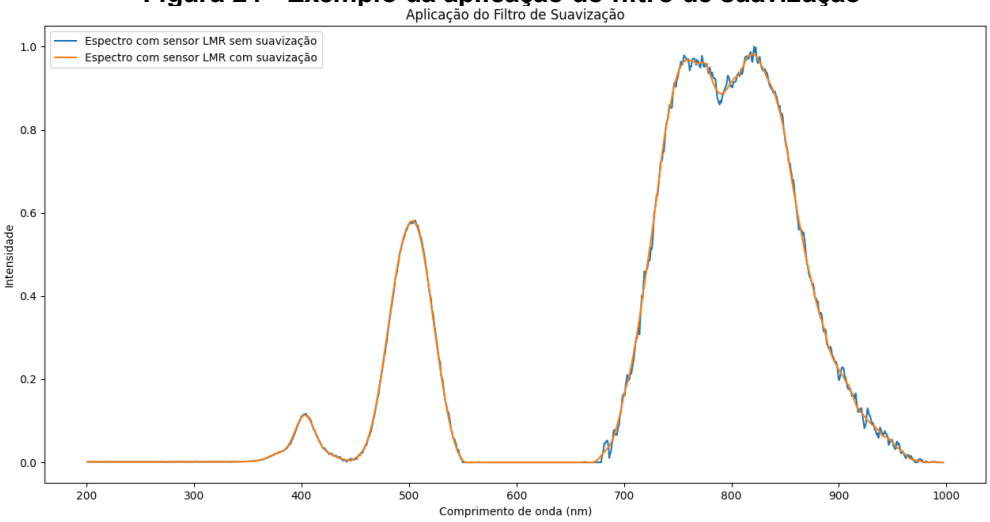
Fonte: Autoria própria (2025)

Se ocorrer um erro, como a incompatibilidade entre o tamanho da janela e a ordem do polinômio, esses valores são ajustados automaticamente. A ordem do polinômio pode ser reduzida até um, e o tamanho da janela pode ser diminuído até encontrar uma configuração viável, ou atingir o valor mínimo de zero.

O script também valida os dados suavizados através da função "erro_suavizacao", que verifica a diferença entre os espectros originais e suavizado, comparando a posição dos maiores picos de ressonância. Se o erro ultrapassar a tolerância (0,8 nm por padrão), o tamanho da janela do filtro é reduzido e o processo é repetido. Caso o ajuste automático não seja bem-sucedido, uma mensagem é exibida ao usuário, indicando que o processo não pôde ser concluído, e retorna ao menu principal.

Ao final do processo, caso a função esteja ativa pelo usuário, o espectro suavizado pode ser exibido através da função "Master Gráfico!", como mostra o exemplo da Figura 24.

Figura 24 - Exemplo da aplicação do filtro de suavização

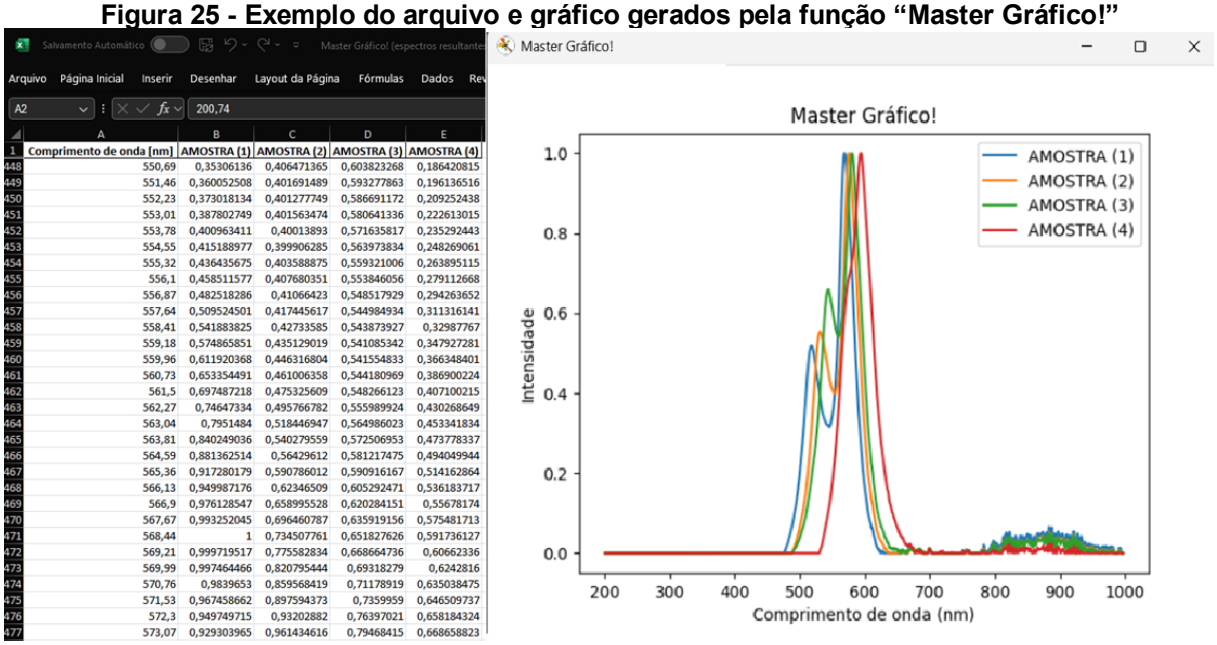


Fonte: Autoria própria (2024)

3.6.5 Master Gráfico!

A função em questão coleta todos os espectros salvos e os subtrai do espectro base para gerar um único gráfico com todos os resultados. Esta tem sua execução condicionada às funções anteriores, sendo elas o “Filtro de Suavização” e “Aplicar Range ao Eixo X (Comprimento de onda [nm])”. Essas funções determinam como o “Master Gráfico!” será plotado para o usuário. Após a criação do gráfico, o script salva os resultados em um único arquivo *.xlsx* denominado como “Master Gráfico! (espectros resultantes)” e exibe a pasta em que foi salvo. Caso a função de suavização esteja ativada, o usuário, também, é questionado sobre a necessidade de salvar os resultados suavizados ou não. Assim, a função emparelha todos os espectros de ressonância LMR em um único gráfico e registra os resultados para facilitar uma análise preliminar dos espectros, como pode ser visto no exemplo da Figura 25, com a exibição dos espectros de 4 amostras.

Figura 25 - Exemplo do arquivo e gráfico gerados pela função “Master Gráfico!”



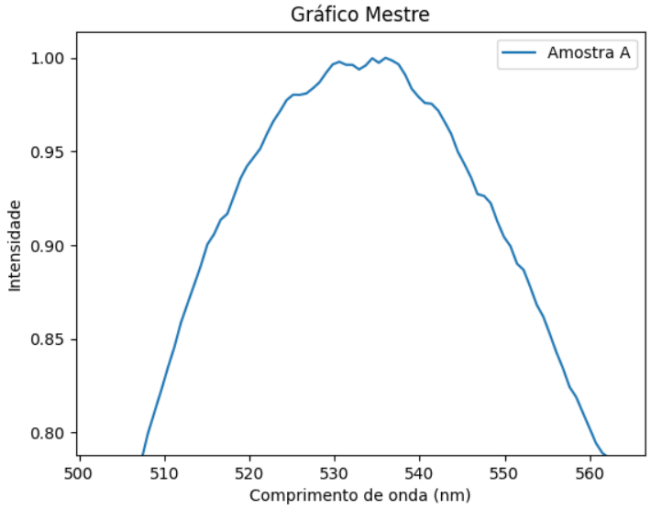
	A	B	C	D	E
1	Comprimento de onda [nm]	AMOSTRA (1)	AMOSTRA (2)	AMOSTRA (3)	AMOSTRA (4)
448	550,69	0,35306136	0,406471365	0,603823268	0,186420815
449	551,46	0,360052508	0,401691489	0,593277863	0,196136516
450	552,23	0,373018134	0,401277749	0,586691172	0,209252438
451	553,01	0,387802749	0,401563474	0,580641336	0,222613015
452	553,78	0,400963411	0,40013893	0,571635817	0,235292443
453	554,55	0,415188977	0,399906285	0,563973834	0,248269061
454	555,32	0,436435675	0,403588875	0,559321006	0,263895115
455	556,1	0,458511577	0,407680351	0,553846056	0,279112668
456	556,87	0,482518286	0,41066423	0,548517929	0,294263652
457	557,64	0,509524501	0,417445617	0,544984934	0,311316141
458	558,41	0,541883825	0,42733585	0,543873927	0,32987767
459	559,18	0,574865851	0,435129019	0,541085342	0,347927281
460	559,96	0,611920368	0,446316804	0,541554833	0,366348401
461	560,73	0,653354491	0,461006358	0,544180969	0,386900224
462	561,5	0,697487218	0,475325609	0,548266123	0,407100215
463	562,27	0,74647334	0,495766782	0,555989924	0,430268649
464	563,04	0,7951484	0,518446947	0,564986023	0,453341834
465	563,81	0,840249036	0,540279559	0,572506953	0,473778337
466	564,59	0,881362514	0,56429612	0,581217475	0,494049944
467	565,36	0,917280179	0,590786012	0,590916167	0,514162864
468	566,13	0,949987176	0,62346509	0,605292471	0,536183717
469	566,9	0,976128547	0,658995528	0,620284151	0,55678174
470	567,67	0,993252045	0,696460787	0,635919156	0,575481713
471	568,44	1	0,734507761	0,651827626	0,591736127
472	569,21	0,999719517	0,775582834	0,668664736	0,60662336
473	569,99	0,997464466	0,820795444	0,69318279	0,6242816
474	570,76	0,9839653	0,859568419	0,71178919	0,635038475
475	571,53	0,967458662	0,897594373	0,7359959	0,646509737
476	572,3	0,949749715	0,93202882	0,76397021	0,658184324
477	573,07	0,929303965	0,961434616	0,79468415	0,668658823

Fonte: Autoria própria (2025)

3.6.6 Teste de Detecção de Picos

A função “Teste de Detecção de Picos” é uma das etapas mais desafiadoras do script, exigindo maior necessidade de processamento computacional, pois envolve a identificação das regiões de picos nos espectros do sensor LMR. Esses picos são cruciais para determinar os comprimentos de onda associados às ressonâncias do sensor e para a subsequente construção de curvas de calibração. A complexidade dessa função está na necessidade de ajustar uma extrapolação matemática que melhor descreve o comportamento do espectro na região dos picos, considerando que os ruídos e as características experimentais podem deformar o espectro original, como se pode observar na Figura 26.

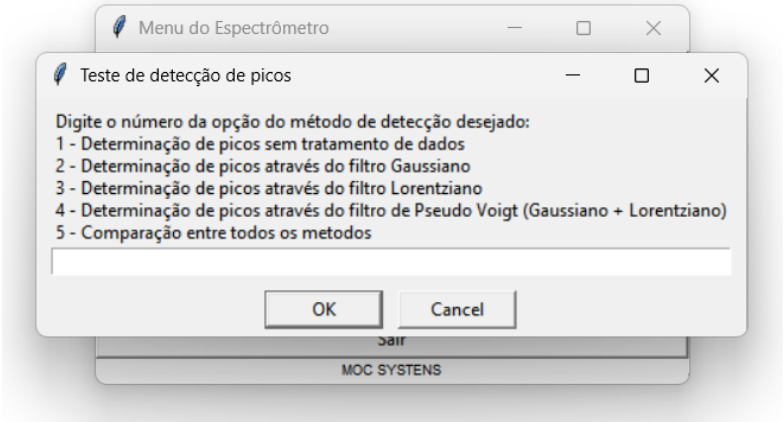
Figura 26 - Exemplo de distorção de pico na região da ressonância LMR



Fonte: Autoria própria (2025)

Com o objetivo de testar diferentes métodos e seus desempenhos para atender as necessidades experimentais e melhorar a confiabilidade dos resultados, a função inicia solicitando ao usuário que escolha dentre cinco métodos disponíveis para detecção de picos: "Determinação de picos sem tratamento de dados", "Determinação de picos através do filtro Gaussiano", "Determinação de picos através do filtro Lorentziano", "Determinação de picos através do filtro Pseudo Voigt (Gaussiano + Lorentziano)", e "Comparação entre todos os métodos" para uma análise de desempenho dentre eles. Cada método possui características únicas que podem se adequar a diferentes tipos de espectros e níveis de ruído e adicionar agilidade ao processo. Na Figura 27, está como o menu de seleção é exibido ao usuário.

Figura 27 – Menu de seleção dos métodos de detecção de pico



Fonte: Autoria própria (2025)

3.6.6.1 Determinação de picos sem tratamento de dados

Esse método identifica os picos diretamente nos dados originais, sem aplicar suavizações ou ajustes matemáticos, utilizando a função “find_peaks” (Figura 28), da biblioteca “*scipy*”, assim como os demais métodos utilizam, porém de forma direta para localizar as posições dos picos, com base em parâmetros configuráveis, como altura mínima, proeminência, distância entre os picos e largura. Tendo que o pico a ser obtido do espectro será o que possuir o valor máximo de intensidade e atender os parâmetros definidos na Figura 28 (SCIPY, 2025).

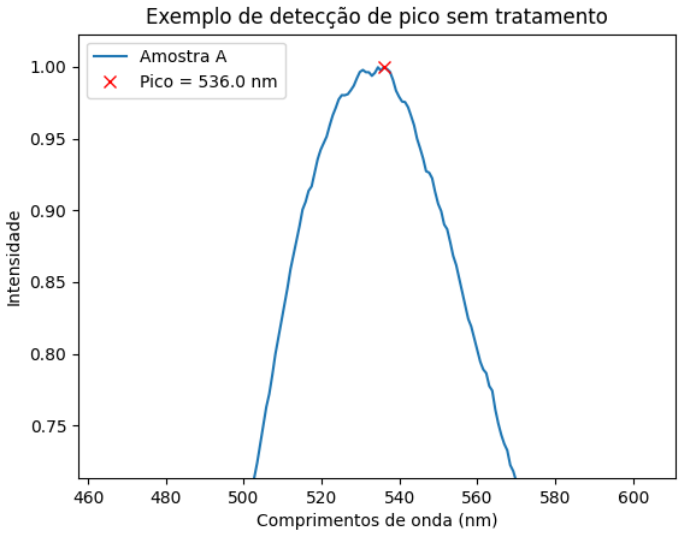
Figura 28 – Função “find_peaks”

```
find_peaks(intensidades, height=1, prominence=0.5, width=10, distance=10)
```

Fonte: Autoria própria (2025)

Apesar de ser rápido e eficiente, esse método pode ser limitado quando os espectros apresentam ruídos ou deformações significativas. Isso pode ser observado na Figura 29, onde o pico é deslocado devido à presença de um ruído predominante. Mas ainda podendo servir de referência para a conferência com os demais métodos.

Figura 29 – Exemplo de deslocamento de pico devido a ruídos no espectro



Fonte: Autoria própria (2025)

3.6.6.2 Determinação de picos através do filtro Gaussiano

O método do filtro Gaussiano é uma abordagem mais complexa para a detecção de picos em comparação ao método anterior, pois modela teoricamente cada região do pico do espectro como uma curva de perfil gaussiano, buscando o

melhor ajuste aos dados experimentais. Isso torna o método especialmente útil para suavizar o espectro e reduzir a influência de ruídos. Assim, como também é amplamente usado no processamento de imagens digitais para suavizar e reduzir ruídos na observação de astros (JESUS; COSTA, 2015).

O método foi desenvolvido seguindo o fluxograma da Figura 30. Após carregar os espectros das amostras a função inicia utilizando o mesmo método anterior para encontrar os picos originais dos espectros. Em seguida, calcula-se a prominência e a largura de cada pico. Esses cálculos ajudam a definir as regiões ao redor dos picos para a otimização, criando uma máscara de pontos que consideram apenas os valores próximo ao pico do espectro original. Uma quantidade limitada de máscaras é criada (75 partições), as quais abrange um intervalo ajustável de altura relativa entre 40% a 70% da altura do pico original. A máscara é construída para reter os pontos que estão entre a base esquerda e direita do pico e acima da altura definida otimizando a necessidade de processamento computacional. Após essa identificação inicial, o filtro realiza um ajuste fino ao redor de cada pico.

Em seguida, utilizando como base a equação gaussiana:

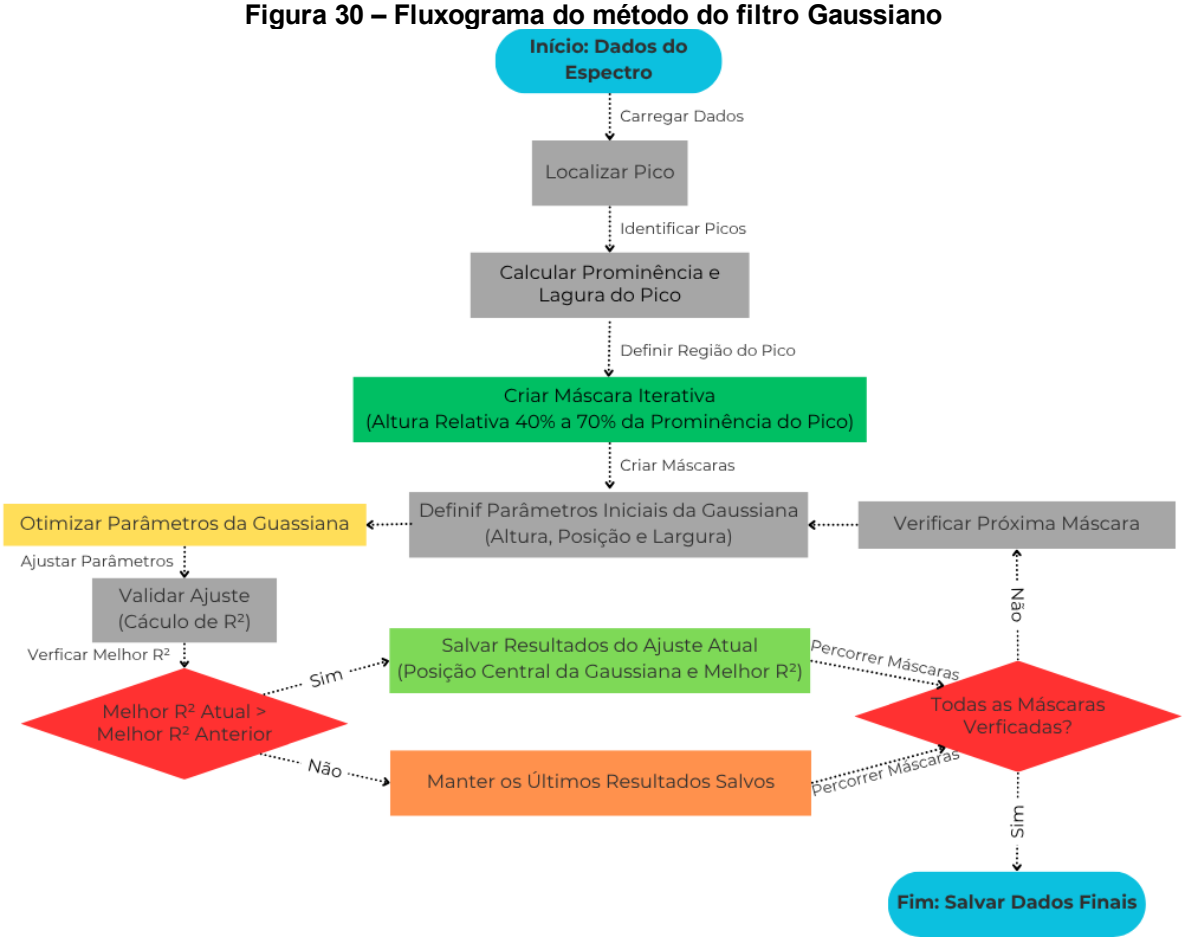
$$G(x) = a * e^{-\frac{(x-b)^2}{2*c^2}} \qquad \textbf{(6.1)}$$

Os parâmetros iniciais da curva gaussiana, como altura "a", posição central "b" e largura "c", são definidos com base nos dados obtidos (MORAES *et al.*, 2022).

Então, a função "minimize" da biblioteca "*scipy*", utilizando o algoritmo "L-BFGS-B", é empregada para ajustar os parâmetros da gaussiana de forma a minimizar o erro do coeficiente de determinação (R^2) entre a curva ajustada e os dados experimentais. Durante o processo de otimização, o código realiza ajustes automáticos para melhorar a precisão. Ele verifica continuamente se o ajuste atende aos critérios de qualidade, como manter a amplitude próxima à altura do pico original, não variando mais que 10% da intensidade, e a posição central com uma variação máxima menor que 5% da posição identificada inicialmente. Se o valor de R^2 não for o melhor em comparação a sua última iteração, os parâmetros não são salvos, e o processo segue percorrendo todas as máscaras criadas, salvando apenas a melhor curva gaussiana que descreva os dados com o maior valor do coeficiente de determinação dentre todas as iterações (SCIPY, 2025).

Ao fim do método, determinado os melhores parâmetros, a posição central (“b”) da função gaussiana é armazenado junto ao valor do R^2 armazenados para etapa de saída de dados da função principal.

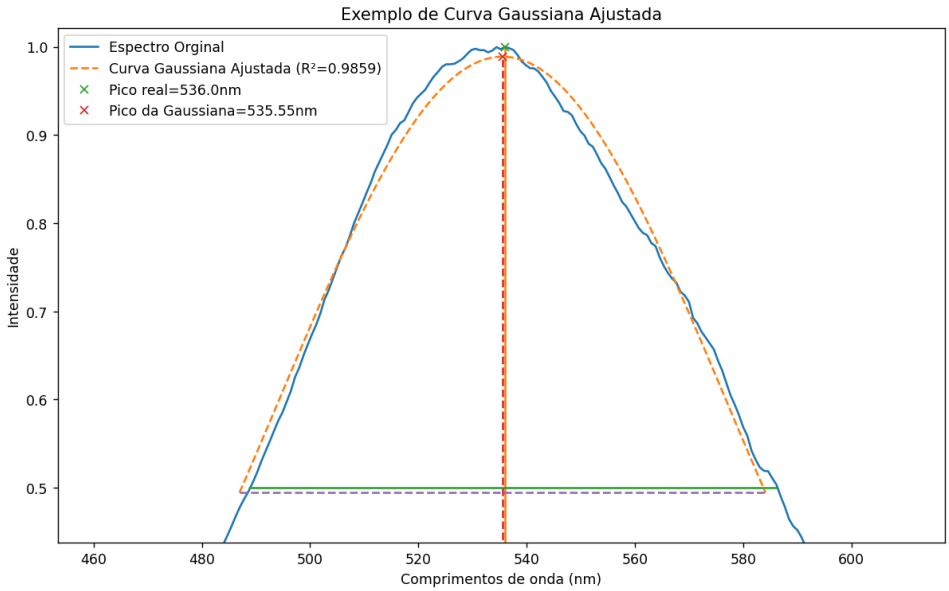
Figura 30 – Fluxograma do método do filtro Gaussiano



Fonte: Autoria própria (2025)

Essas etapas permitem uma análise mais detalhada do espectro, proporcionando uma identificação mais precisa e confiável dos picos, mesmo na presença de ruídos experimentais. Na Figura 31 é demonstrado um exemplo desse processo interno realizado pelo método, é possível observar que o pico é ligeiramente deslocado de posição devida à projeção da curva gaussiana sobre os dados originais.

Figura 31 – Exemplo do processo interno do método de detecção de pico com filtro Gaussiano



Fonte: Autoria própria (2025)

3.6.6.3Determinação de picos através do filtro Lorentziano

Este método é especialmente útil em espectros experimentais onde os picos possuem formatos mais próximos de uma distribuição Lorentziana, o que pode ocorrer devido a processos de ressonância ou características específicas das amostras analisadas.

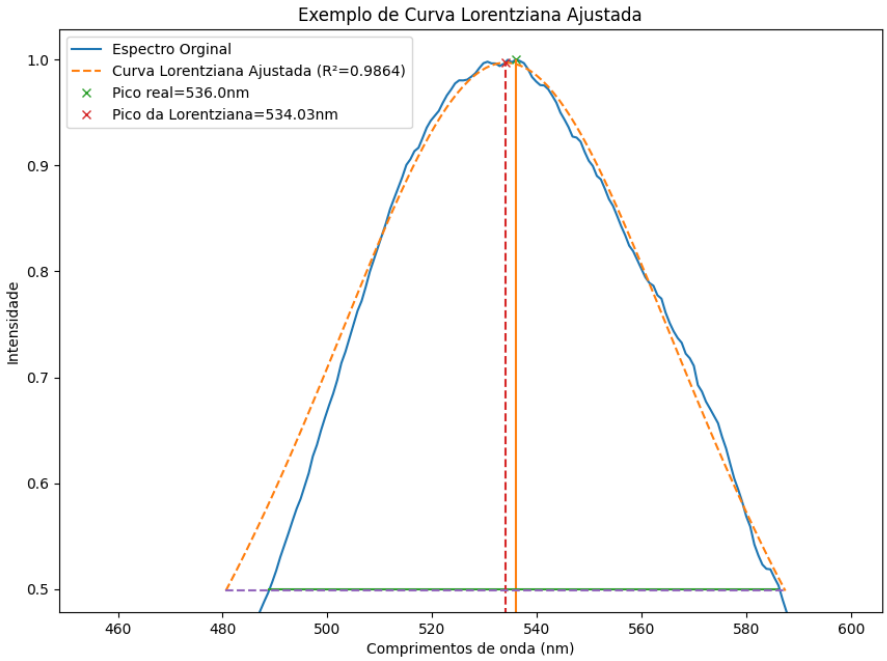
O método do filtro Lorentziano segue o mesmo fluxograma do método Gaussiano, conforme ilustrado na Figura 30, mas adapta o ajuste matemático para uma função Lorentziana, que pode ser descrita como:

$$L(x) = \frac{a * c^2}{(x-b)^2 + c^2} \qquad \textbf{(6.2)}$$

Nesse caso, "a" é a altura, "b" a posição central, e "c" a meia largura a meia altura do cume da curva (MORAES *et al.*, 2022). Após a identificação inicial dos picos com "find_peaks", o método define as máscaras de pontos baseadas em uma altura relativa (40% a 70% do pico original com 75 partições) e otimiza os parâmetros da função Lorentziana usando o algoritmo "minimize" para maximizar o coeficiente de determinação, seguindo os mesmos parâmetros de limitações do método anterior. Então, todas as máscaras criadas são percorridas, salvando apenas os melhores ajustes que maximizem R^2. Ao final, o valor da posição central e o valor de R^2 da melhor função Lorentziana ajustada são armazenados para análise posterior.

A Figura 45 apresenta um exemplo do ajuste Lorentziano aplicado ao mesmo espectro experimental utilizado para a demonstração do filtro Gaussiano (Figura 31). Sendo possível notar que para este caso, ocorreu uma pequena melhora no valor do coeficiente de determinação, com um maior deslocamento do pico da curva projetada ao valor original do espectro.

Figura 32 – Exemplo do processo interno do método de detecção de pico com filtro Lorentziano



Fonte: Autoria própria (2025)

3.6.6.4Determinação de picos através do filtro de Pseudo Voigt

O método do filtro Pseudo Voigt é o mais complexo entre todos os apresentados e de maior necessidade computacional, pois combina características das funções Gaussiana e Lorentziana em uma única função, permitindo uma modelagem mais flexível dos espectros experimentais. A função Pseudo Voigt utiliza um fator de mistura "m", variando entre 0 e 1, para ajustar a contribuição relativa de cada componente (Gaussiana e Lorentziana) à forma final da curva. Esse método é muito útil em situações em que os picos possuem características intermediárias entre os dois modelos ou quando o formato dos picos de ressonância não é bem definido (MORAES *et al.*, 2022).

Esse método segue o mesmo fluxograma descrito anteriormente para o filtro Gaussiano (Figura 30), iniciando da mesma maneira e seguindo os mesmos

processos e limitações no deslocamento dos picos em relação a posição no espectro original.

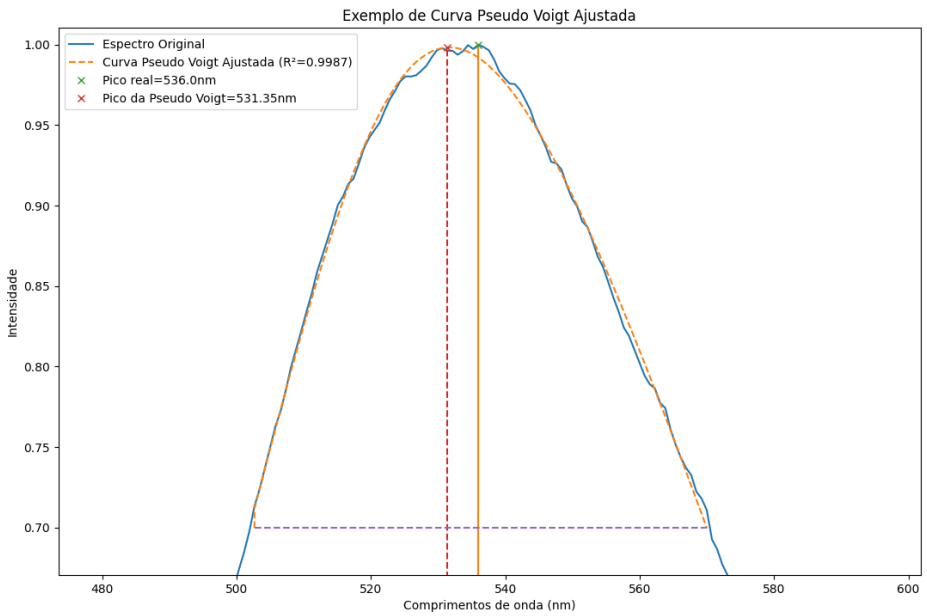
O diferencial do método, está na utilização da função Pseudo Voigt e na complexidade do ajuste, que exige a otimização simultânea de todos os parâmetros das funções Gaussiana e Lorentziana, além do fator de mistura "m". A equação da curva Pseudo Voigt é descrita como:

$$PV(x) = m * L(x) + (1 - m) * G(x) \quad \textbf{(6.3)}$$

Onde $L(x)$ é a função Lorentziana e $G(x)$ é a função Gaussiana (MORAES *et al.*, 2022). Durante o processo, são ajustados parâmetros como a amplitude, posição central e largura das curvas individualmente, bem como o fator "m", buscando minimizar o erro em relação aos dados originais e assim maximizando o coeficiente de determinação, e ao final salvando a posição do pico da curva ajustada e o melhor valor R^2 encontrado.

O exemplo do processo interno deste método é ilustrado na Figura 33, utilizando o mesmo espectro de exemplo dos métodos anteriores, onde se observa, que para este caso, a curva Pseudo Voigt proporciona um ajuste mais preciso ao espectro experimental, superando as limitações dos filtros Gaussiano ou Lorentziano.

Figura 33 – Exemplo do processo interno do método de detecção de pico com filtro Pseudo Voigt



Fonte: Autoria própria (2025)

Esse método é o mais confiável entre os apresentados, sendo uma boa escolha para situações em que a análise exige maior precisão na identificação dos picos em espectros mais ruidosos.

3.6.6.5Comparação entre todos os métodos

O método de "Comparação entre todos os métodos" não é exatamente um método independente de detecção de picos, mas uma abordagem que simplifica a escolha entre os quatro métodos apresentados anteriormente. Sendo projetado para o usuário verificar dentre qual das técnicas possuem os melhores resultados para os dados experimentais os quais serão analisados.

A função executa automaticamente cada um dos métodos, aplicando-os aos espectros fornecidos. Ao final da execução, os melhores valores de R^2 e a posição do pico da função ajustada de cada método é salvada para análise e exibição posterior.

Portanto, o objetivo deste método é permitir ao usuário avaliar qual das abordagens é mais adequada para os dados experimentais, considerando ruídos, deformações e outras características específicas do espectro. Assim, economizando tempo e esforço, já que elimina a necessidade de o usuário testar cada método manualmente, para situações em que os espectros possuem variabilidade significativa ou quando não há uma técnica previamente conhecida que funcione melhor.

3.6.6.6Saída de dados da função

Após aplicar o método selecionado, a função salva os dados processados em um arquivo *.xlsx* nomeado como "Valores testes de pico", localizado na pasta “Determinando picos”. Esses dados incluem os comprimentos de onda dos picos detectados para cada amostra e os valores de R^2, que são gerados nos métodos que realizam ajustes matemáticos, como os filtros Gaussiano, Lorentziano e Pseudo Voigt. Esses resultados são importantes para analisar a posição dos picos e avaliar a qualidade dos ajustes realizados, permitindo que o usuário tenha uma ideia preliminar da precisão de cada método.

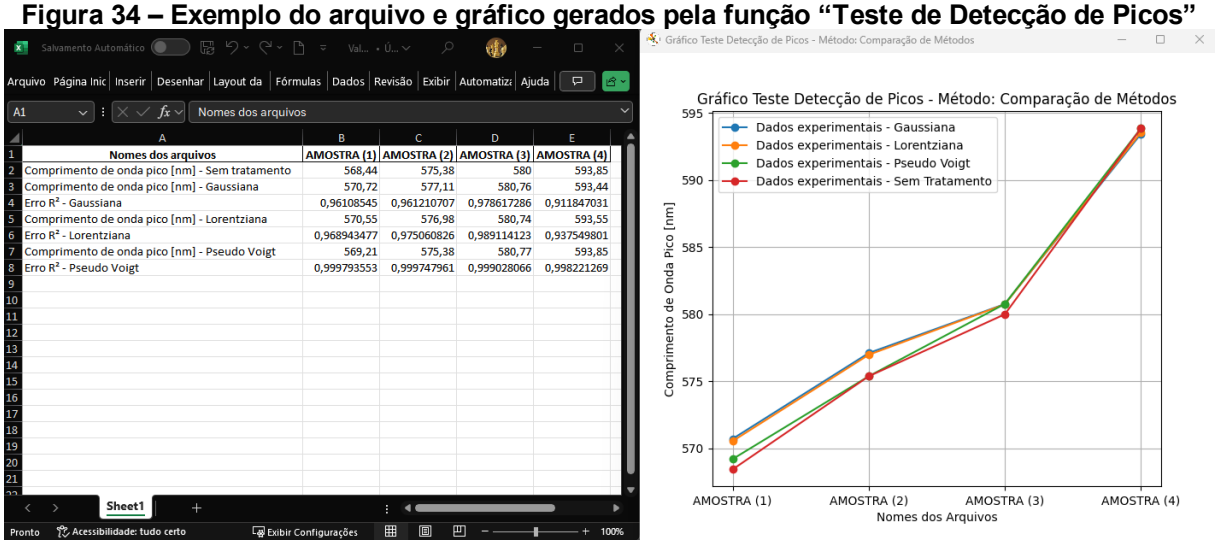
Quando a opção de "Comparação entre todos os métodos" é escolhida, os resultados são organizados lado a lado, mostrando as posições dos picos e os valores de R^2 de cada método. Isso facilita a comparação direta, ajudando o usuário a identificar o método mais adequado para os dados experimentais, considerando

ruídos ou características específicas do espectro. Essa abordagem agiliza o processo de análise ao consolidar os resultados de forma clara e objetiva.

Além disso, a função também gera gráficos que mostram as posições dos picos detectados por cada método. Esses gráficos utilizam marcadores e legendas para diferenciar os métodos e amostras, apresentando o comprimento de onda dos picos no eixo vertical e os nomes dos arquivos de cada amostra no eixo horizontal. Assim, o usuário pode visualizar as diferenças entre os métodos e a tendência geral entre as amostras, além de identificar variações causadas por ruídos ou ajustes.

A Figura 34 ilustra as saídas da função, incluindo a planilha de resultados e o gráfico gerado na opção de comparação entre métodos. Nota-se que, para a primeira amostra, o método Lorentziano apresenta um pico deslocado, sugerindo baixa confiabilidade para calibração do sensor LMR, possivelmente devido a ruído nos dados. Já o método Pseudo Voigt se destaca por manter maior proximidade com o pico original e melhores valores de R^2.

Figura 34 – Exemplo do arquivo e gráfico gerados pela função "Teste de Detecção de Picos"



Nomes dos arquivos	AMOSTRA (1)	AMOSTRA (2)	AMOSTRA (3)	AMOSTRA (4)
Comprimento de onda pico [nm] - Sem tratamento	568,44	575,38	580	593,85
Comprimento de onda pico [nm] - Gaussiana	570,72	577,11	580,76	593,44
Erro R^2 - Gaussiana	0,96108545	0,961210707	0,978617286	0,911847031
Comprimento de onda pico [nm] - Lorentziana	570,55	576,98	580,74	593,55
Erro R^2 - Lorentziana	0,968943477	0,975060826	0,989114123	0,937549801
Comprimento de onda pico [nm] - Pseudo Voigt	569,21	575,38	580,77	593,85
Erro R^2 - Pseudo Voigt	0,999793553	0,999747961	0,999028066	0,998221269

Fonte: Autoria própria (2025)

A saída organizada, com os dados estruturados por amostra e método, torna o processo mais eficiente e facilita o uso das informações para aplicações futuras, como definir qual método optar para a construção das curvas de calibração ou estudos mais detalhados sobre o comportamento dos sensores.

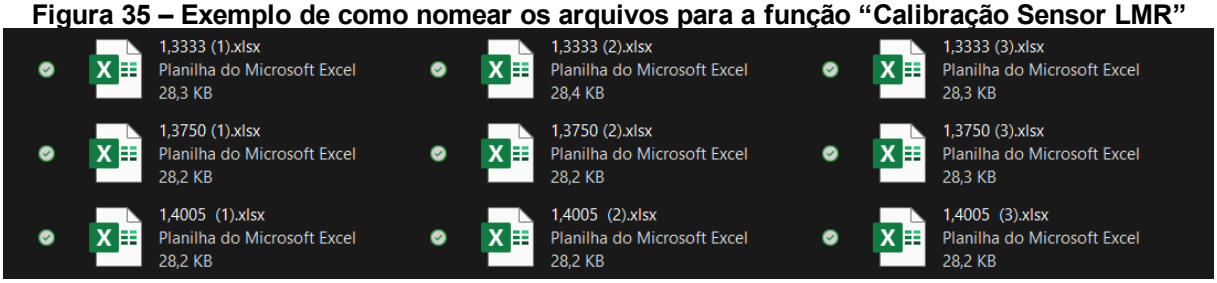
3.6.7 Calibração Sensor LMR

A função "Calibração Sensor LMR", a principal deste código, tem como objetivo estabelecer uma relação matemática entre o comprimento de onda dos picos de ressonância detectados no espectro do sensor LMR e os valores de índice de refração medidos por um instrumento laboratorial, como por exemplo, um refratômetro. A partir dessa calibração, é possível converter futuras medições do sensor em valores de índice de refração e aferir o comportamento geral do processo experimental.

3.6.7.1 Entrada de dados

Para realizar a calibração, o usuário deve fornecer os arquivos *.xlsx* contendo os espectros das amostras. Esses arquivos devem estar organizados na pasta **"**Dados para a Calibração**"**, criada automaticamente pelo script no mesmo diretório onde o código está alocado. Esses arquivos devem conter os espectros das amostras seguindo o padrão apresentado na Figura 20. Cada arquivo deve ser nomeado de acordo com o valor do índice de refração correspondente à amostra. Além disso, é importante que todas as amostras tenham a mesma quantidade de repetições medidas para os mesmos índices. Por exemplo, caso existam três medições para os índices de refração 1,3333, 1,3750 e 1,4005, os arquivos devem ser nomeados conforme o exemplo mostrado na Figura 35.

Figura 35 – Exemplo de como nomear os arquivos para a função "Calibração Sensor LMR"

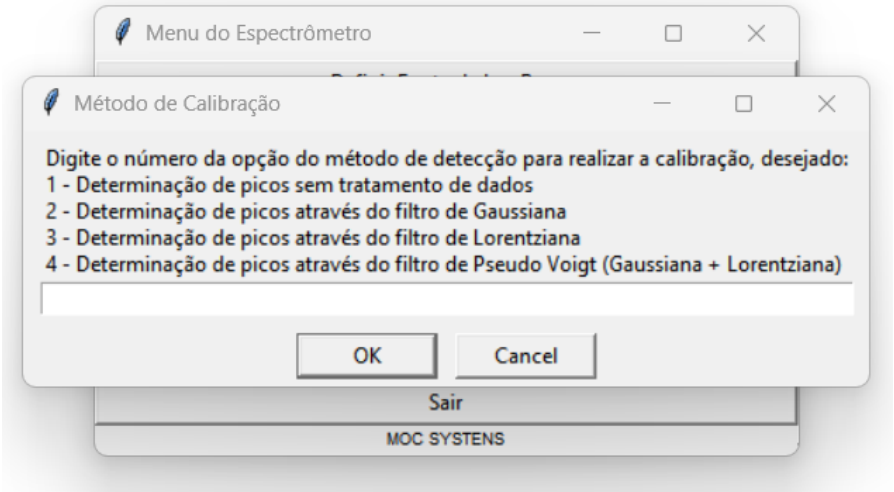


Fonte: Autoria própria (2025)

3.6.7.2 Processamento de dados

O processo de calibração inicia com a importação dos espectros e a identificação dos picos de ressonância em cada amostra. Para isso, a função exibe um menu como o da Figura 36, onde o usuário deve escolher um dos métodos para a detecção de picos disponíveis.

Figura 36 – Menu para a escolha do método para o processo de calibração do sensor LMR



Fonte: Autoria própria (2025)

Esses métodos são os mesmos utilizados na função "Teste de Detecção de Picos", seguindo os mesmos processos de análise e parâmetros. A partir da seleção do método, a função executa a detecção dos picos em cada espectro e calcula a média das posições para cada índice de refração de mesmo valor. Além disso, determina o desvio padrão entre as amostras, o que permite avaliar a confiabilidade dos valores médios obtidos.

Após essa etapa, o usuário é solicitado a informar o erro do refratômetro utilizado na medição dos índices de refração, o qual é considerado como padrão de ±0,0002 do valor medido. Esse valor é fundamental para que as barras de erro sejam corretamente representadas no gráfico final, refletindo a incerteza das medições experimentais.

Ao final, a função ajusta uma equação polinomial de segunda ordem que melhor se adapte a tendência da correlação entre os comprimentos de onda dos picos médios obtidos, e os valores de índice de refração medidos. Além disso, o valor de custo do ajuste, R^2, é calculado, e ambos os resultados são salvos para serem exibidos posteriormente.

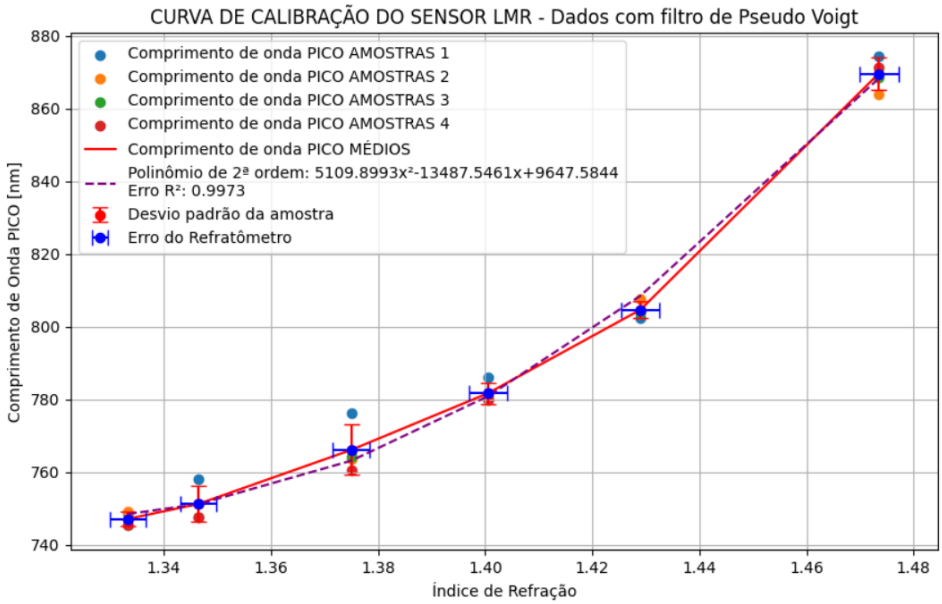
3.6.7.3 Saída de dados

A função gera três principais saídas:

- ***Gráfico da curva de calibração*** (Figura 37): apresenta a curva ajustada juntamente com os pontos experimentais, permitindo

visualizar a relação entre os picos detectados e os índices de refração. O gráfico também inclui barras de erro, calculadas a partir do desvio padrão das amostras e do erro do refratômetro informado pelo usuário. Além disso, o gráfico é salvo como uma imagem *.png*, nomeada como "grafico_calibracao".

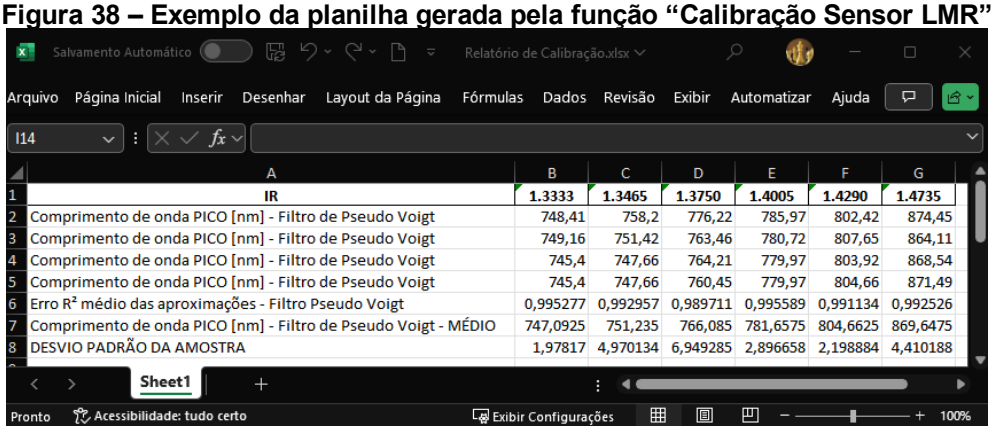
Figura 37 – Exemplo do gráfico gerado pela função "Calibração Sensor LMR"



Fonte: Autoria própria (2025)

- ***Arquivo de planilha*** (Figura 44): os dados processados são salvos em um arquivo *.xlsx* chamado **"**Relatório de Calibração**"**. Esse arquivo contém os valores médios das posições dos picos, os desvios padrões e os valores médios de R^2, que indicam a qualidade do ajuste.

Figura 38 – Exemplo da planilha gerada pela função “Calibração Sensor LMR”

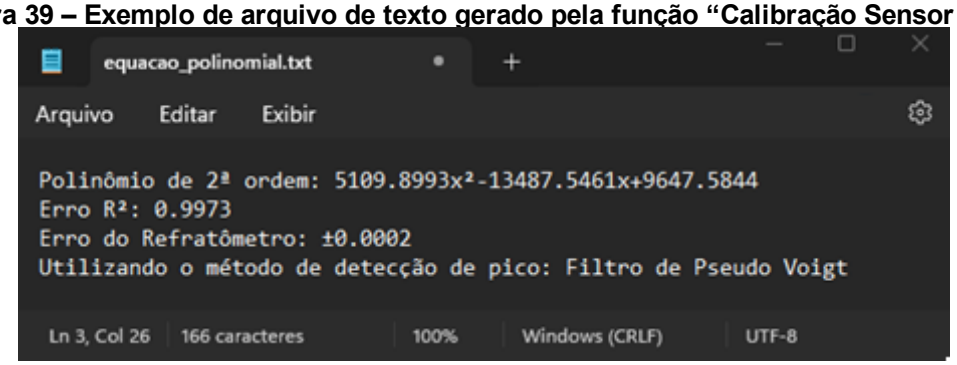


	A	B	C	D	E	F	G
1	IR	1.3333	1.3465	1.3750	1.4005	1.4290	1.4735
2	Comprimento de onda PICO [nm] - Filtro de Pseudo Voigt	748,41	758,2	776,22	785,97	802,42	874,45
3	Comprimento de onda PICO [nm] - Filtro de Pseudo Voigt	749,16	751,42	763,46	780,72	807,65	864,11
4	Comprimento de onda PICO [nm] - Filtro de Pseudo Voigt	745,4	747,66	764,21	779,97	803,92	868,54
5	Comprimento de onda PICO [nm] - Filtro de Pseudo Voigt	745,4	747,66	760,45	779,97	804,66	871,49
6	Erro R^2 médio das aproximações - Filtro Pseudo Voigt	0,995277	0,992957	0,989711	0,995589	0,991134	0,992526
7	Comprimento de onda PICO [nm] - Filtro de Pseudo Voigt - MÉDIO	747,0925	751,235	766,085	781,6575	804,6625	869,6475
8	DESVIO PADRÃO DA AMOSTRA	1,97817	4,970134	6,949285	2,896658	2,198884	4,410188

Fonte: Autoria própria (2025)

- ***Arquivo de texto*** (Figura 39): um relatório adicional em arquivo *.txt* denominado como " equacao_polinomial", também é criado, para registrar a equação da curva ajustada, o coeficiente de determinação, o erro do refratômetro utilizado e o método utilizado na detecção de picos.

Figura 39 – Exemplo de arquivo de texto gerado pela função “Calibração Sensor LMR”



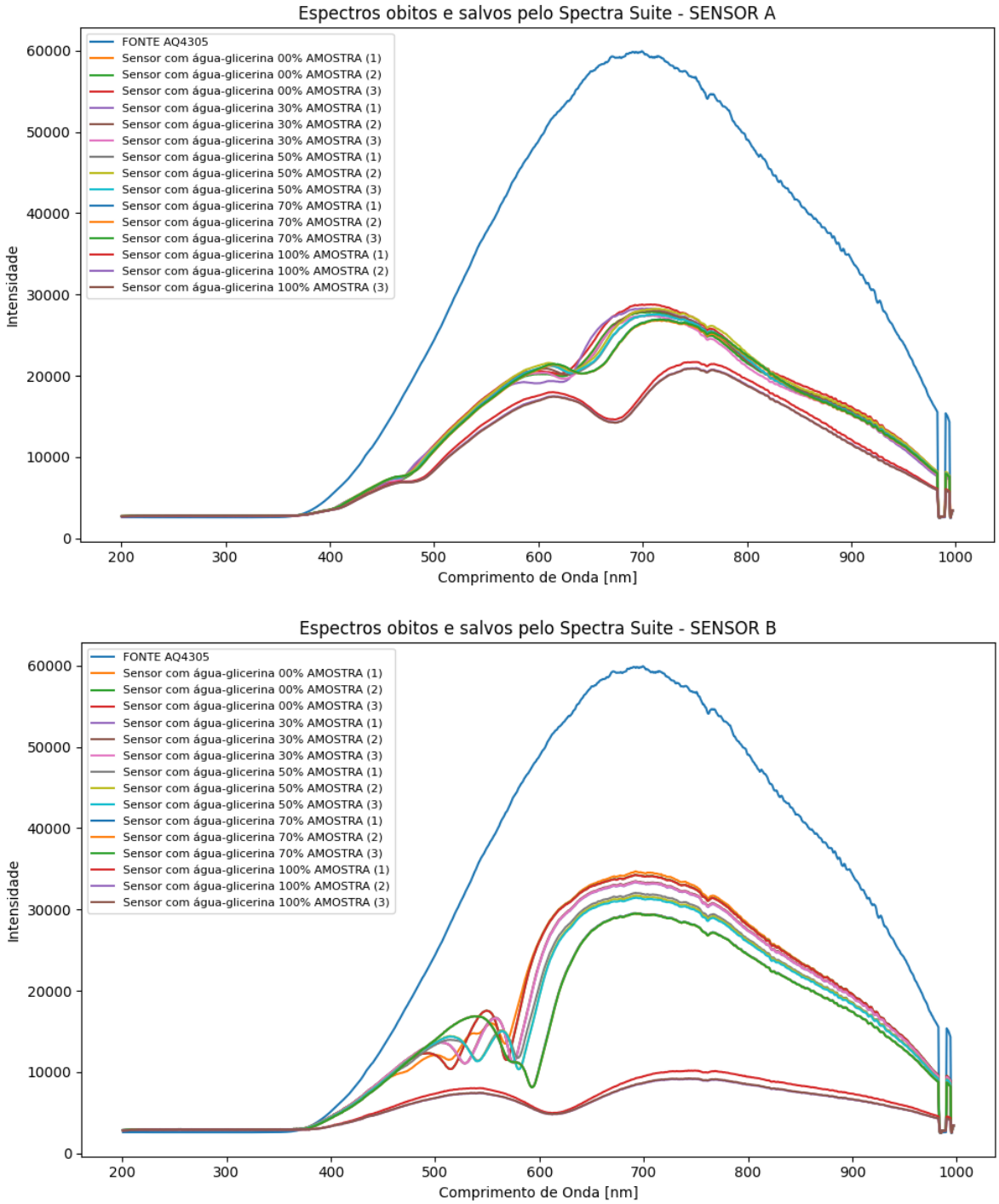
Fonte: Autoria própria (2025)

Todas as saídas salvas pela função de calibração, são direcionadas para pasta nomeada como “Relatório de Calibração”, que é criada automaticamente no mesmo local onde o código está salvo. Assim, essas informações garantem que o processo de calibração seja documentado e possa ser utilizado posteriormente para demais análises.

3.7 Inclusão dos dados coletados no script desenvolvido

Com todos os espectros das soluções água/glicerina medidos, através da bancada experimental, e salvos pelo Spectra Suite, os dados então foram todos importados para o script personalizado desenvolvido em Python (Figura 40).

Figura 40 - Gráficos com todos os espectros obtidos pelo *Spectra Suite*



Fonte: Autoria própria (2025)

O processo de importação foi realizado individualmente para cada sensor, garantindo que não houvesse interferências entre os dados dos sensores “A” e “B”. Dessa forma, os espectros de cada sensor foram carregados separadamente no script para o processamento.

Após a importação dos dados, o próximo passo foi a definição da fonte de luz base. Para ambos os sensores foi utilizada a mesma fonte de luz como referência. O

espectro escolhido como base foi a "**FONTE AQ4305**", definido por meio da função "***Definir Fonte de Luz Base***" no script.

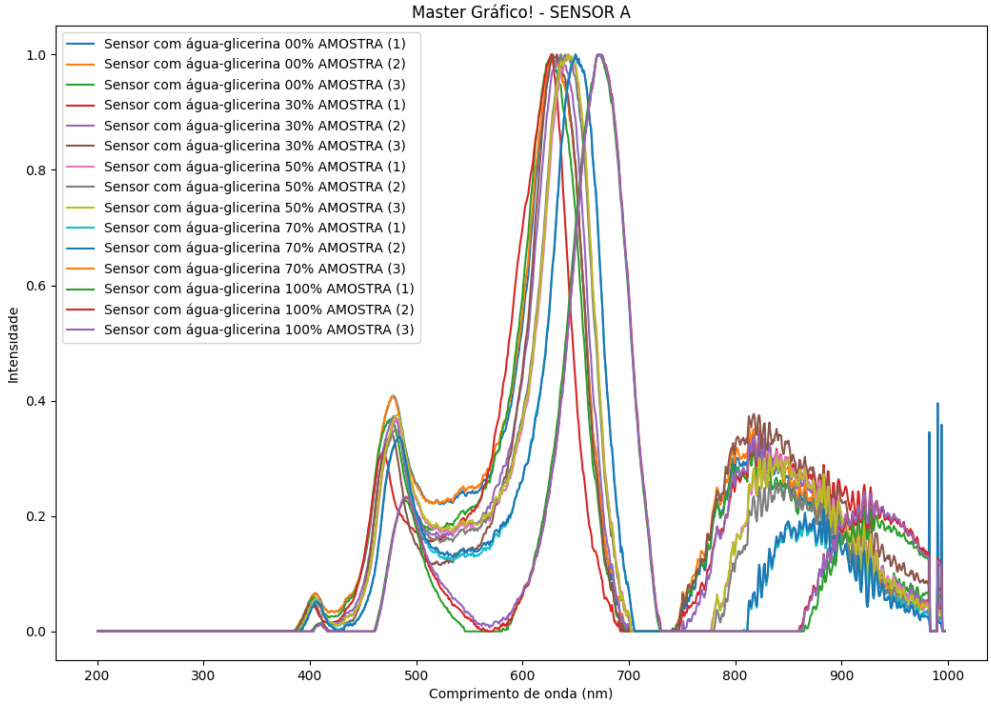
Com a definição da fonte de luz base, as demais funções foram habilitadas, permitindo a exibição dos espectros processados com as ressonâncias LMR devidamente isoladas, os comprimentos de onda de pico e a suas curvas de calibrações. Nas próximas seções são demonstrados os resultados para ambos os sensores e suas comparações.

4 RESULTADOS E DISCUSSÕES

4.1 Resultados Sensor A

Utilizando a função "***Master Gráfico!***" para o conjunto de dados do Sensor A, foram obtidos como saída o gráfico apresentado na Figura 41.

Figura 41 – "Master Gráfico!" para o processo de caracterização do Sensor A

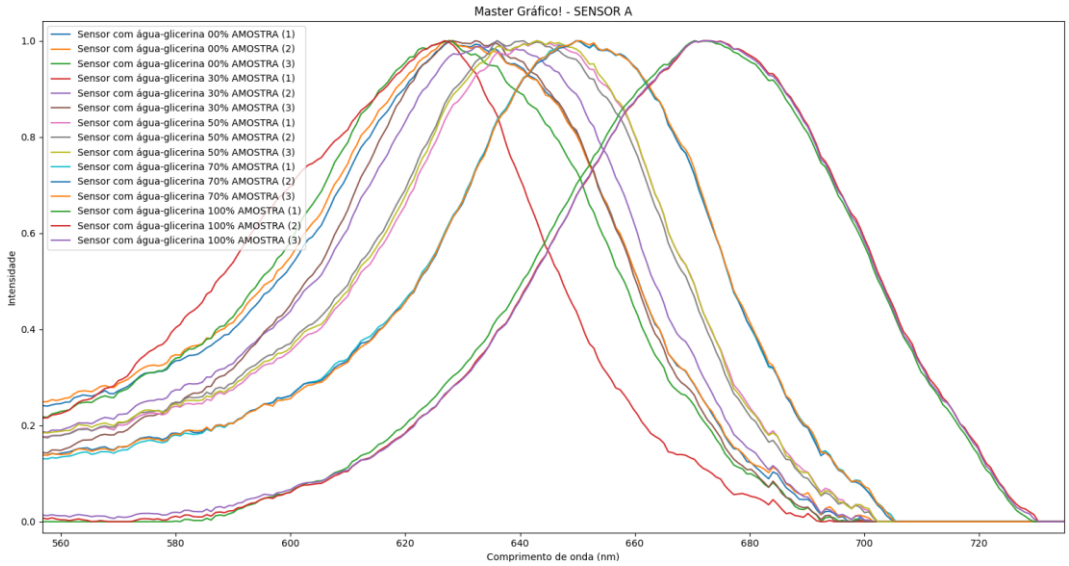


Fonte: Autoria própria (2025)

Para o sensor LMR utilizado em questão é possível observar que houve, quatro regiões de ressonância existentes, as quais sofrem um deslocamento para a direita conforme o aumento da concentração de glicerina nas amostras, resultando, também, em um aumento do comprimento de onda dos picos. Portanto, esse comportamento confirmou que o Sensor A é sensível às variações no meio e poderia ser utilizado no processo de calibração.

No entanto, é perceptível que houve a presença de ruídos do experimento que atrapalharam, principalmente, a visão do quarto pico (740nm à 1000nm). Porém conforme definido no processo de desenvolvimento do script, a análise é focada no pico de maior altura, que, neste caso, correspondeu ao terceiro pico de ressonância (550 nm a 740 nm) como demonstrado na Figura 42.

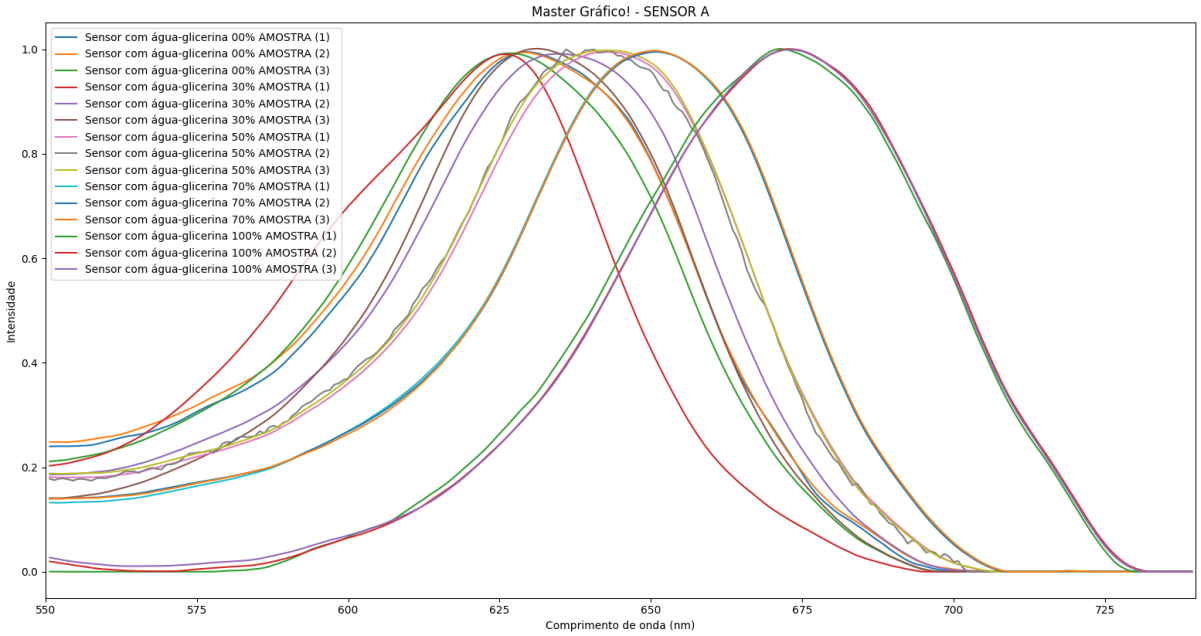
Figura 42 – “Master Gráfico!” detalhe na ressonância de foco (caracterização do Sensor A)



Fonte: Autoria própria (2025)

Para minimizar os efeitos dos ruídos na visualização, foi aplicada a função "***Filtro de Suavização***”. Em seguida, um novo "***Master Gráfico!***" foi gerado, agora com os comprimentos de onda delimitados por meio da função "***Aplicar range ao eixo x (Comprimento de onda [nm])***", permitindo focar exclusivamente na terceira ressonância LMR e analisá-la mais detalhadamente, conforme ilustrado na Figura 43.

Figura 43 – “Master Gráfico!” com filtros e restrição (caracterização do Sensor A)



Fonte: Autoria própria (2025)

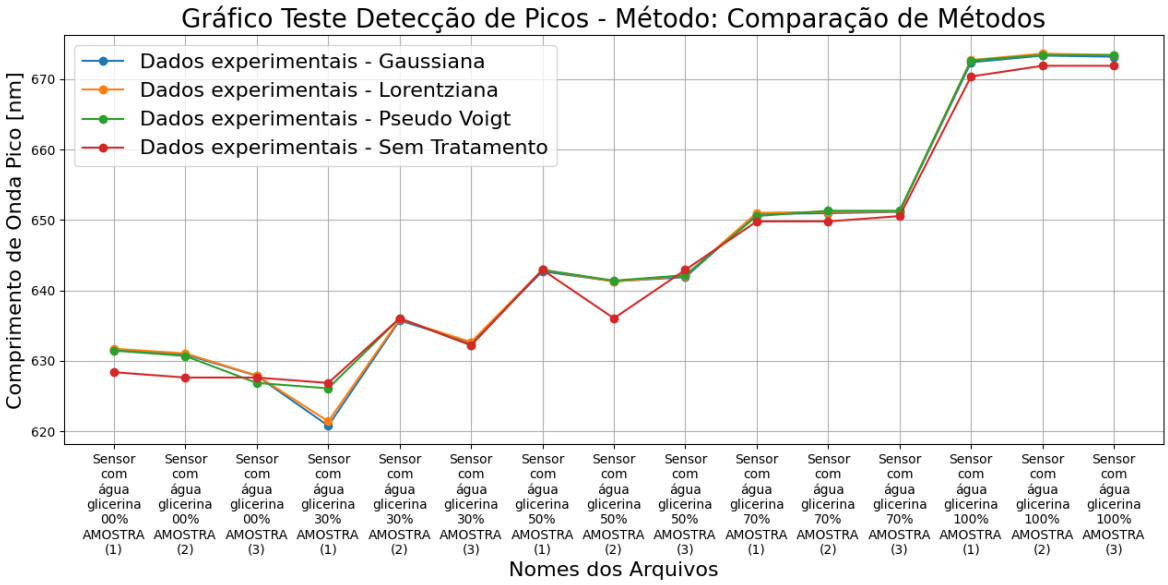
Para o conjunto de dados obtidos para o Sensor A é possível notar que algumas amostras demonstraram um comportamento mais ruidoso ou fora do padrão das

demais. Embora isso possa influenciar o processo de calibração, não impossibilitou a sua realização, permitindo ainda a retirada de informações relevantes para a caracterização do sensor LMR.

Após a análise inicial dos picos dos espectros experimentais, foi aplicada a função “**Teste de Detecção de Picos**” do script. Nessa etapa, foi necessário avaliar os diferentes métodos de detecção disponíveis para identificar aquele que melhor se adapta ao comportamento dos espectros obtidos.

Para uma comparação rápida entre todos os métodos, foi selecionado a opção “5” da função. A Figura 44 mostra os resultados para cada método, correlacionando os nomes dos arquivos e o comprimento de onda da posição do pico da ressonância determinada pelo tratamento definido.

Figura 44 – “Gráfico Teste Detecção de Picos” (caracterização do Sensor A)



Fonte: Autoria própria (2025)

Os resultados demonstraram que, para o conjunto de dados experimentais obtidos, todos os métodos de detecção de picos convergiram para pontos próximos aos dados sem tratamento. No entanto, observou-se variações no deslocamento dos picos, com alguns métodos apresentando menor desvio em relação aos dados originais, enquanto outros exibiram um deslocamento mais significativo.

Entretanto, é necessário definir um único método para o processo de calibração. Portanto, foi necessário consultar a segunda forma de saída de dados da

função de testes, a planilha de dados "*Valores testes de pico.xlsx*", onde contém as informações dos erros de aproximação dos dados reais com as funções projetadas.

Tabela 2 - "Valores testes de pico" (caracterização do Sensor A)

Amostras	00% média	30% média	50% média	70% média	100% média	Média R²
Pico [nm] Sem tratamento	627,87	631,70	640,63	650,05	671,40	
Pico [nm] Gaussiana	630,11	629,69	641,96	651,00	672,99	
Erro R² Gaussiana	0,99458	0,96483	0,98944	0,99482	0,99762	0,98826
Pico [nm] Lorentziana	630,23	629,99	642,09	651,13	673,27	
Erro R² Lorentziana	0,98975	0,96423	0,97942	0,98661	0,99428	0,98286
Pico [nm] Pseudo-Voigt	629,66	631,45	642,16	651,07	673,18	
Erro R² Pseudo-Voigt	0,99518	0,99533	0,98944	0,99482	0,99762	0,99448

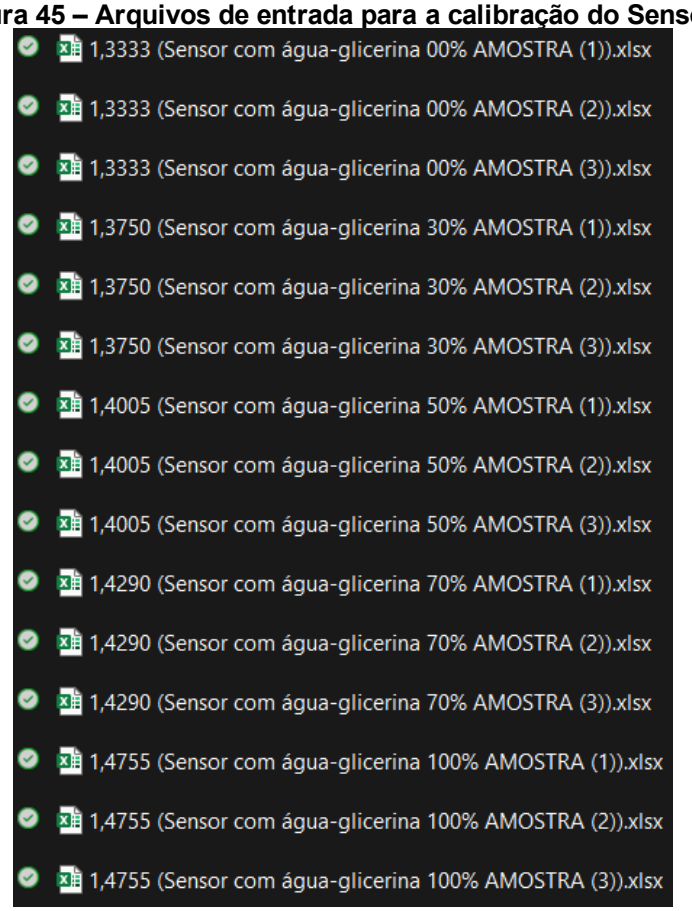
Fonte: Autoria própria (2025)

Após a análise dos resultados fornecidos pela função na planilha (Tabela 2), foi determinado que o método a ser utilizado para a calibração do Sensor A seria o Pseudo Voigt. Esse método apresentou os melhores valores de R^2 entre os avaliados, com uma média de 0,99448, superior à média obtida pelos métodos Gaussiano (0,98827) e Lorentziano (0,98286). Além disso, o Pseudo Voigt demonstrou um menor deslocamento das posições dos picos de ressonância em relação aos dados iniciais.

Definido o método de detecção, prosseguiu-se com o processo de calibração utilizando a última função do script, "Calibração do Sensor LMR". Para isso, foi necessário ajustar os dados de entrada, garantindo que a função pudesse operar corretamente.

Todos os dados amostrais, previamente salvos em planilhas, foram renomeados conforme seus respectivos valores de índice de refração da solução água-glicerina e alocados dentro da pasta "Dados para a Calibração" gerada pelo próprio script. Esses valores de IR foram obtidos anteriormente no processo experimental, conforme descrito na etapa de preparação da bancada. A Figura 45 ilustra como os arquivos foram organizados.

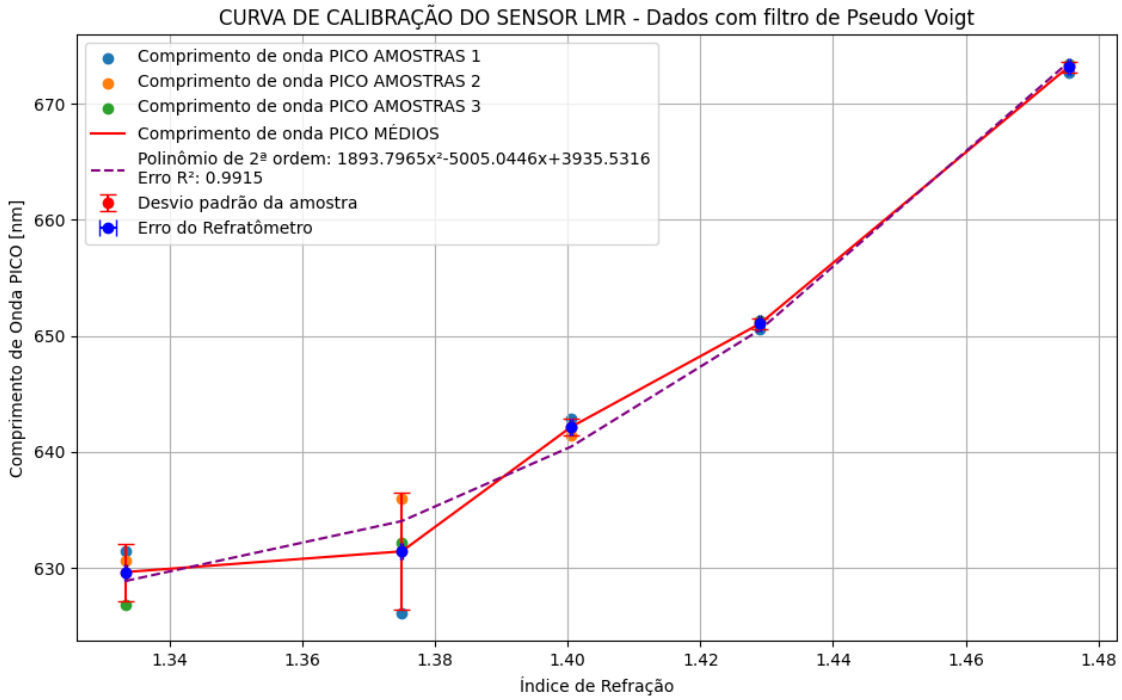
Figura 45 – Arquivos de entrada para a calibração do Sensor A



Fonte: Autoria própria (2025)

Com a função de calibração alimentada com os dados de entrada, foi necessário apenas selecioná-la e ajustá-la para o método de detecção de pico (opção 4). Além disso, foi informado o valor do erro do refratômetro, que, por padrão, já está configurado de acordo com o erro do equipamento utilizado (±0,0002).

Figura 46 –Gráfico da curva de calibração do Sensor A



Fonte: Autoria própria (2025)

Por fim, como pode ser observado na Figura 46, obteve-se a curva de calibração do Sensor A, apresentando os valores médios e os erros associados às medições e equipamentos utilizados. Nota-se que o coeficiente de determinação da equação polinomial de segunda ordem encontrada R^2 = 0,9915 está próximo do valor máximo, indicando que o processo convergiu para um resultado satisfatório, e através da equação **4.2** e a Tabela 3, foi possível determinar uma aproximação da sensibilidade do sensor como sendo de 306,02 nm/RIU.

Tabela 3 – Resultados da curva de calibração do Sensor A

IR	1.3333	1.3750	1.4005	1.4290	1.4755
Comprimento de onda PICO [nm] Filtro de Pseudo Voigt	631,45	626,09	642,92	650,56	672,67
Comprimento de onda PICO [nm] Filtro de Pseudo Voigt	630,68	636,04	641,39	651,32	673,43
Comprimento de onda PICO [nm] Filtro de Pseudo Voigt	626,85	632,21	642,16	651,32	673,43
Erro R^2 médio das aproximações Filtro Pseudo Voigt	0,99518	0,99533	0,98944	0,99482	0,99751
Comprimento de onda PICO [nm] Filtro de Pseudo Voigt - MÉDIO	629,66	631,4467	642,1567	651,0667	673,1767
DESVIO PADRÃO DA AMOSTRA	2,46380	5,01873	0,76501	0,43879	0,43879

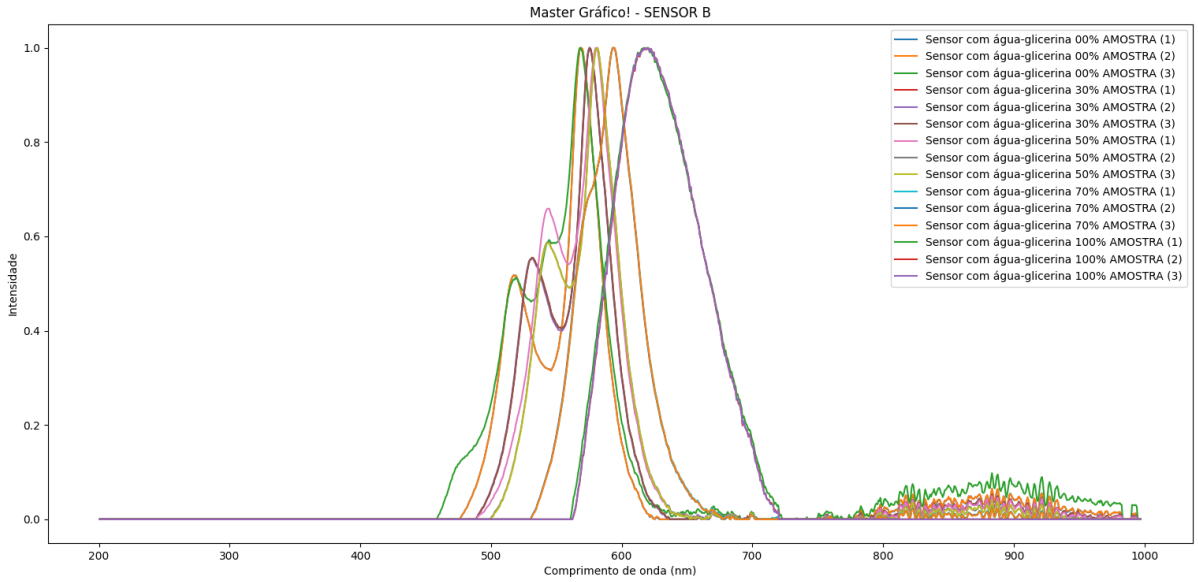
Fonte: Autoria própria (2025)

4.2 Resultados Sensor B

Seguindo a mesma metodologia aplicada ao Sensor A, o Sensor B foi caracterizado para avaliação do seu desempenho na calibração e resposta às variações do meio para posteriores comparações entre diferentes sensores LMR.

O processo teve início com a elaboração do gráfico por meio da função "***Master Gráfico!***", utilizando o conjunto de dados experimentais do Sensor B. Como mostrado na Figura 47, para o sensor LMR em questão, existe uma única região de ressonância bem definida em todas as amostras (550 nm a 720 nm), deslocando-se para a direita com o aumento da concentração de glicerina nas soluções. Esse comportamento indicou que o Sensor B estava sensível a variações em seu meio circundante, o que permitiu a sua continuidade no seu processo de caracterização.

Figura 47 – "Master Gráfico!" para o processo de caracterização do Sensor B



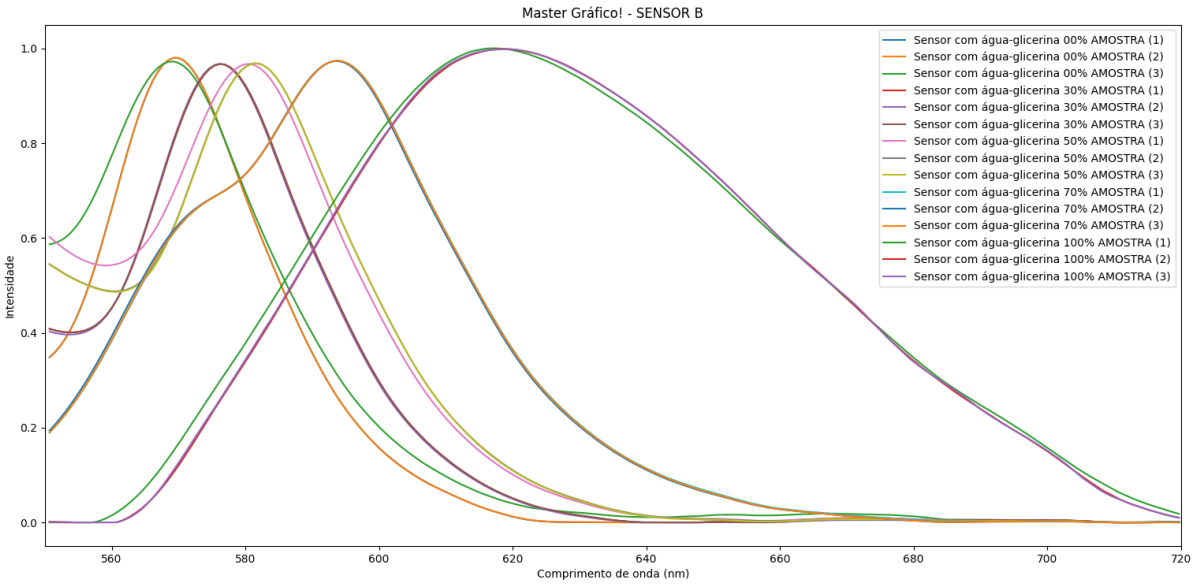
Fonte: Autoria própria (2025)

No entanto, observou-se que, em soluções com menores concentrações de glicerina, surgem picos de ressonância secundários, que desaparecem em amostras mais concentradas ou se confundem com os ruídos do experimento. Esta condição no comportamento deste sensor LMR não impediu que o processo de calibração fosse realizado, mas houve mais atenção a sua sensibilidade para a calibração.

Aproximando a visualização dos picos de ressonância em foco, é possível observar a diferença de largura entre os picos, assim como o pouco deslocamento entre as amostras da mesma solução e uma baixa presença de ruídos. A Figura 48

apresenta essa visualização mais detalhada, já configurada com o filtro de suavização e com as delimitações ajustadas no eixo dos comprimentos de onda.

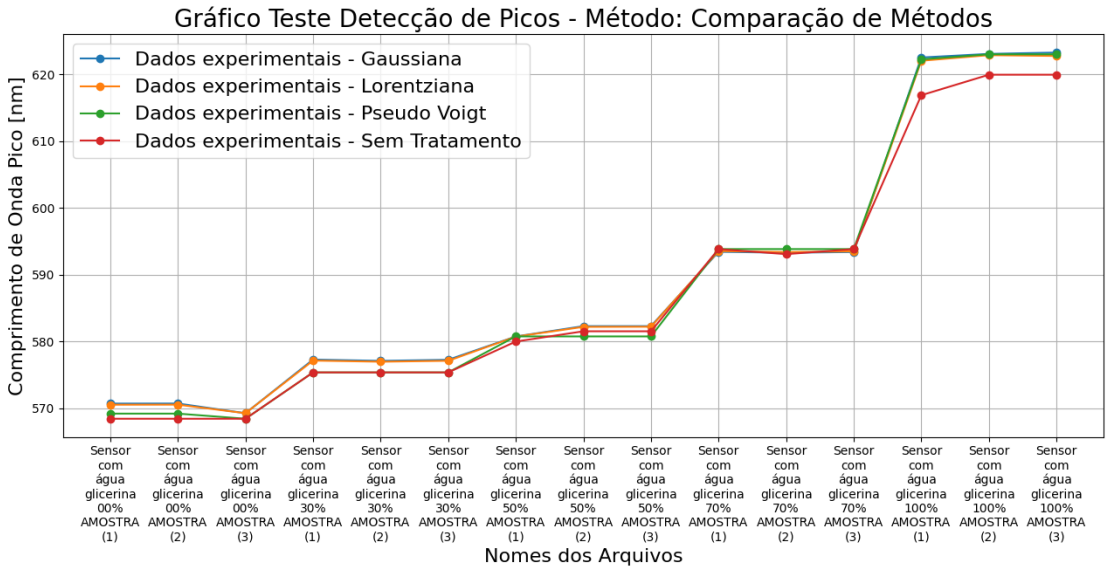
Figura 48 – "Master Gráfico!" com filtros e restrição (caracterização do Sensor B)



Fonte: Autoria própria (2025)

Após verificar as condições iniciais do comportamento do Sensor B, foi utilizado a função "***Teste de Detecção de Picos***" para comparar e determinar o melhor método de ajuste para definir a posição dos picos de ressonância.

Figura 49 – "Gráfico Teste Detecção de Picos" (caracterização do Sensor B)

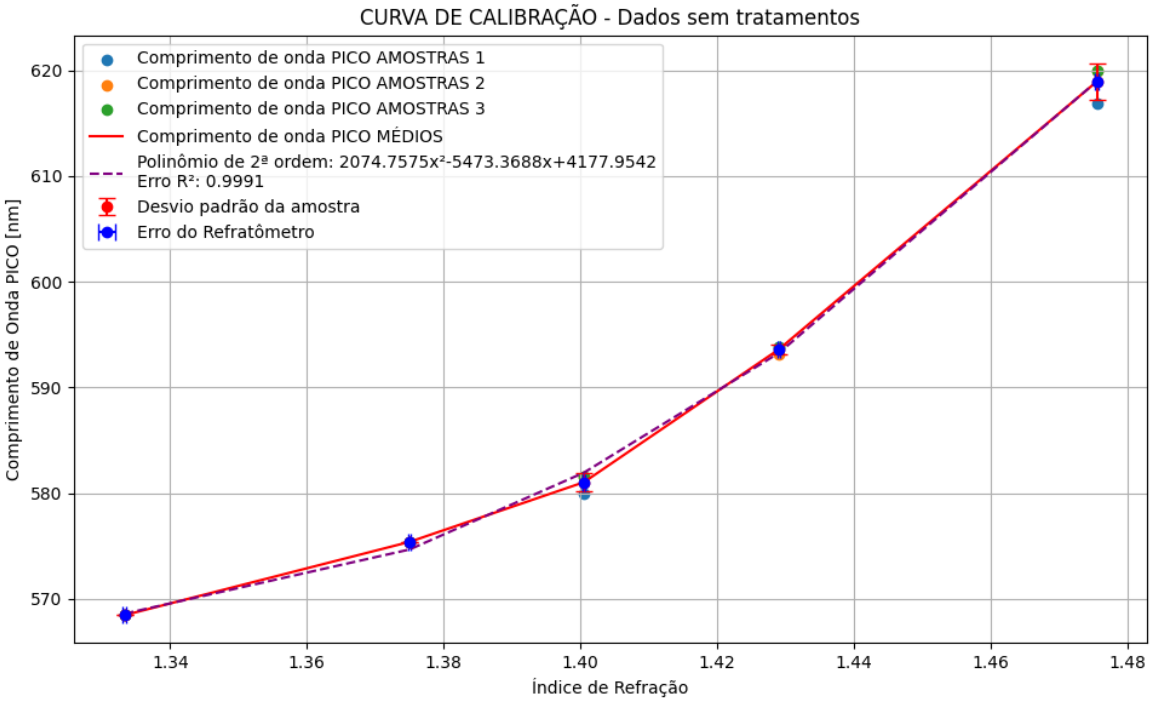


Fonte: Autoria própria (2025)

A Figura 49 mostra que todos os métodos convergiram para pontos próximos aos dados sem tratamento. No entanto, ao analisar a planilha de resultados gerada pela função, observou-se que para os dados sem tratamentos existiam poucas variações nas posições dos comprimentos de onda pico. Esses resultados, fizeram com que, para o conjunto de dados experimentais obtidos pelo Sensor B não existiria a necessidade da utilização de métodos adicionais para correção de ruídos nos picos de ressonância, os dados sem tratamento seriam suficientes para prosseguir com o processo de calibração.

Com o método de detecção de picos definido, os dados foram ajustados e organizados, sendo renomeados conforme seus respectivos valores de índice de refração assim como no processo anterior referente ao Sensor A. Então a função de calibração do sensor LMR do script foi utilizada obtendo com o resultado a curva de calibração representada na Figura 50.

Figura 50 – Gráfico da curva de calibração do Sensor B



Fonte: Autoria própria (2025)

A consistência dos dados experimentais e a escolha do método para detecção de picos se demonstraram eficazes, obtendo valores médios com baixos desvios padrões como se observa na Tabela 4. A função "***Calibração Sensor LMR***" obteve como resultado uma curva de calibração para o Sensor B com um coeficiente de determinação de R^2 =0,9991, um valor muito próximo do máximo para o coeficiente,

e através da equação **4.2** foi possível determinar aproximadamente a sensibilidade do sensor como sendo de 355,11 nm/RIU. Indicando que o sensor LMR utilizado corresponde para sua função de detecção de diferentes índices de refração.

Tabela 4 – Resultados da curva de calibração do Sensor B

IR	1.3333	1.3750	1.4005	1.4290	1.4755
Comprimento de onda PICO [nm] Sem tratamento	568,44	575,38	580	593,85	616,89
Comprimento de onda PICO [nm] Sem tratamento	568,44	575,38	581,54	593,09	619,96
Comprimento de onda PICO [nm] Sem tratamento	568,44	575,38	581,54	593,85	619,96
Comprimento de onda PICO [nm] Sem tratamento - MÉDIO	568,44	575,38	581,0267	593,5967	618,9367
DESVIO PADRÃO DA AMOSTRA	0	0	0,88912	0,43879	1,77247

Fonte: Autoria própria (2025)

4.3 Comparações entre os sensores “A” e “B”

Com os processos de calibração concluídos para ambos os sensores, tornou-se possível fazer uma análise comparativa entre os resultados obtido. O objetivo desta seção é destacar as diferenças e semelhanças encontradas no comportamento dos sensores “A” e “B”, considerando suas curvas de calibração, coeficientes de determinação e sensibilidade às variações no meio.

4.3.1 Diferenças nas respostas espectrais

Os sensores analisados demonstraram deslocamento das ressonâncias para comprimentos de onda maiores à medida que a concentração de glicerina nas amostras foram aumentando, confirmando a sensibilidade de ambos às variações do meio, adequando-os ao processo de calibração.

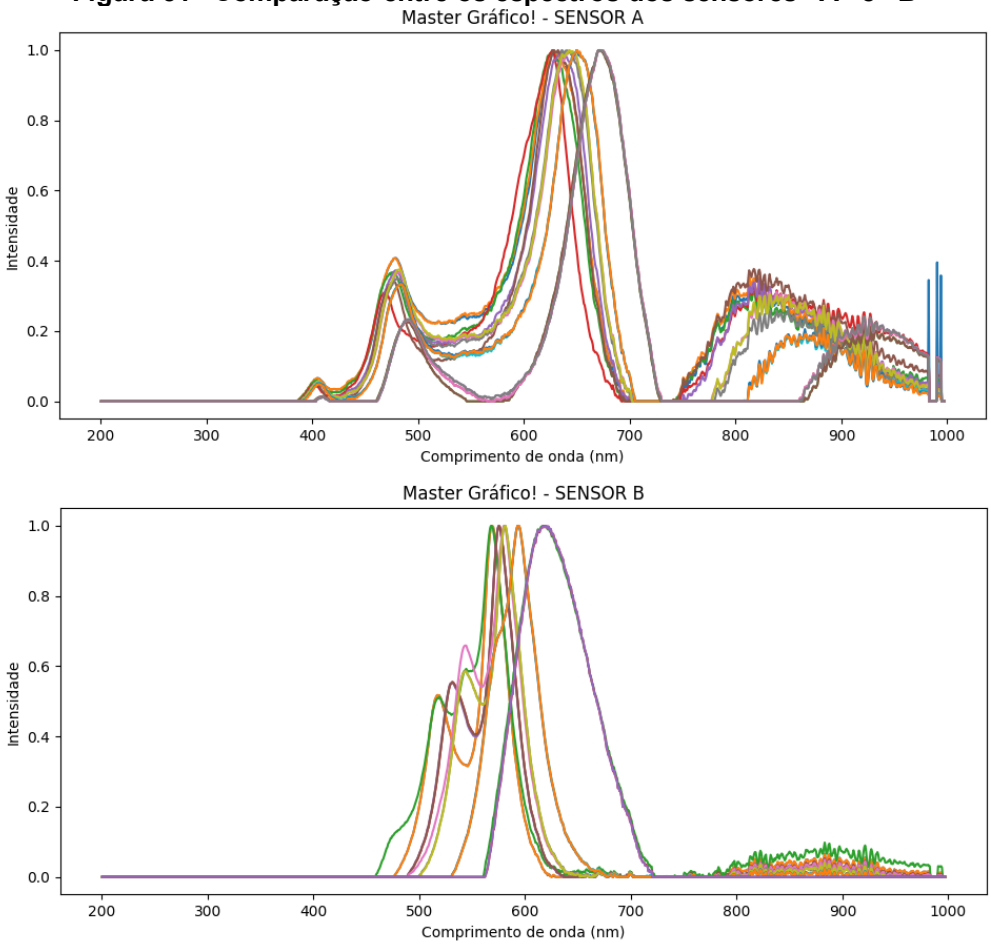
Contudo, diferenças foram constatadas nos espectros e nas ressonâncias apresentadas. Na Figura 51, percebe-se que, na faixa de comprimentos de onda analisados, os comportamentos espectrais dos sensores apresentaram diferenças. Enquanto, o Sensor A revelou quatro regiões de ressonância, o Sensor B exibiu apenas uma ressonância claramente definida.

Essa diferença pode ser atribuída a variações na espessura da camada de revestimento entre os sensores LMR, ambos foram projetados para possuir uma fina cobertura de SnO_2 de 200 nm (DREYER, 2018). Entretanto, como descrito por Arregui (2014), sensores LMR podem ter o ajuste fino das suas regiões de

ressonância com o aumento da espessura do revestimento. Assim, é provável que o Sensor A tenha uma camada ligeiramente mais espessa que o Sensor B, uma vez que ambos apresentam ao menos uma ressonância na mesma faixa de comprimento de onda como pico principal. A diferença na quantidade de ressonâncias, no entanto, aponta para uma espessura maior na cobertura do Sensor A.

Isso destaca a importância da precisão nos processos de fabricação desses sensores, evidenciando a baixa tolerância a variações nas etapas produtivas e a complexidade de replicar sensores LMR idênticos.

Figura 51 –Comparação entre os espectros dos sensores "A" e "B"



Fonte: Autoria própria (2025)

4.3.2 Diferenças na escolha do método de detecção de picos

O processo experimental para exposição das amostras de água e glicerina apresentou diferenças entre os sensores avaliados. Enquanto o Sensor A exibiu maior presença de ruídos em determinadas concentrações, o Sensor B demonstrou comportamento mais estável, com menor ocorrência de interferências ao longo das

medições. Essa disparidade pode ser atribuída tanto a diferenças de sensibilidade entre os sensores quanto a possíveis interferências externas durante a exposição das amostras. Tal diferença, também, impactou diretamente na necessidade da aplicação de métodos de correção durante o processamento dos dados.

Portanto, para a detecção de picos, foram avaliados diferentes métodos. O Sensor A demandou o uso do método Pseudo Voigt, o qual apresentou o melhor ajuste, devido ao ruido existente, e assim alcançando um coeficiente de determinação R^2 = 0,9915. Por outro lado, o Sensor B, não necessitou de métodos adicionais para correção dos dados, os quais já se mostraram adequados para a calibração. Esse aspecto resultou em um coeficiente de determinação ainda maior, R² = 0,9991.

Consequentemente o comportamento mais estável do Sensor B destaca a importância de minimizar ao máximo as interferências externas e os ruídos no ambiente experimental. Quando os dados podem ser utilizados sem a necessidade de métodos adicionais de suavização ou correção, como evidenciado pelos resultados do Sensor B, obtêm-se calibrações mais confiáveis e coeficientes de determinação elevados, reforçando a relevância de garantir condições ideais para medições precisas.

4.3.3 Diferenças nas curvas de calibração e sensibilidade

As curvas de calibração confirmam a aptidão de ambos os sensores para detectar variações no índice de refração, porém apresentam diferenças de desempenho. O Sensor B destacou-se por uma melhor linearidade, menor desvio padrão e maior sensibilidade (355,11 nm/RIU). Por outro lado, o Sensor A, embora tenha apresentado maior presença de ruídos e menor sensibilidade (306,02 nm/RIU), ainda demonstrou um ajuste consistente aos dados experimentais. Dessa forma, enquanto o Sensor B evidenciou estabilidade e dispensou correções adicionais, o Sensor A demandou mais do processamento de tratamento de dados.

Esses resultados reforçam a relevância do processo de calibração e da necessidade de escolher o sensor mais adequado, considerando as exigências de robustez, estabilidade e sensibilidade específicas para cada aplicação.

5 CONCLUSÕES

O Em síntese, este trabalho desenvolveu um método otimizado de calibração para sensores ópticos baseados em ressonância de modos de perdas (LMR), acompanhado pela criação de um script em Python capaz de automatizar o processamento dos dados espectrais, desde a importação até a detecção de picos e a geração das curvas de calibração. Com isso, buscou-se aprimorar a eficiência da caracterização desses sensores, reduzindo erros e tornando o processo de análise mais ágil e preciso.

A partir dos experimentos realizados, verificou-se que a metodologia proposta permitiu a calibração mais ágil dos sensores LMR, evidenciando suas respostas espectrais e a influência das variações do meio no deslocamento das ressonâncias. A comparação entre os dois diferentes sensores testados para os experimentos (Sensor “A” e “B”) demonstrou que fatores como a espessura do revestimento e a presença de ruídos podem impactar a sensibilidade e a repetibilidade dos resultados. Contudo, a aplicação dos métodos que incorporam ajustes matemáticos garantiu a realização da calibração em ambos os sensores, consolidando a viabilidade da abordagem desenvolvida.

O software criado mostrou-se eficiente ao executar automaticamente todas as etapas da análise, incluindo importação, detecção do comprimento de onda do pico das ressonâncias, aplicação de ajustes e geração das curvas de calibração. Os métodos de detecção de picos implementados funcionaram inclusive em sinais mais ruidosos, como os obtidos no processo de medição do Sensor A. Nesse caso, o método de ajuste Pseudo-Voigt, desenvolvido e incorporado ao software, apresentou bom desempenho, permitindo identificar com precisão o pico mesmo em condições espectrais desfavoráveis e garantindo que a calibração fosse possível para sensores com diferentes características.

Os resultados obtidos também demonstraram a viabilidade de calibrar sensores LMR e de extrair, por meio da equação de calibração gerada pelo software desenvolvido, o comportamento do deslocamento da ressonância em função do índice de refração, reforçando a possibilidade de utilizar esses sensores como refratômetros para medição do índice de refração. Entretanto, para validar sua aplicação em medições de óleos isolantes minerais de transformadores, voltadas a análises físico-químicas em condições reais de operação, torna-se necessário expandir os estudos

utilizando a metodologia desenvolvida neste trabalho, em especial com a calibração dos sensores na faixa típica desses óleos. Também é importante aprimorar o processo de escolha do sensor LMR, identificando aquele que apresente o melhor desempenho e maior sensibilidade, considerando que diferentes sensores podem responder de maneira distinta, como foi observado ao longo deste estudo.

Assim, este trabalho deixa como resultado uma metodologia desenvolvida que representa uma alternativa simples e de fácil replicação para a caracterização de sensores LMR. Os resultados obtidos abrem caminho para sua aplicação no processo de monitoramento preditivo de transformadores de potência, com potencial de implementação on-line e integração a sistemas de gestão de ativos, contribuindo para uma manutenção mais eficiente e segura dos equipamentos.

5.1 Limitações do espectrômetro e impactos na medição

Durante a caracterização dos sensores LMR, foi utilizado o espectrômetro QE65000 da *Ocean Optics*®, reconhecido por sua alta eficiência quântica e sensibilidade (OCEAN OPTICS INC., 2007). No entanto, algumas limitações pertencentes ao dispositivo podem impactar a precisão das medições e a calibração do sensor como refratômetro.

A resolução espectral do dispositivo, com precisão de ±0,05 nm e erro de calibração de ±0,1 nm, pode comprometer a sensibilidade das equações geradas, pois os sensores LMR apresentam deslocamentos espectrais sutis conforme mínimas variações do índice de refração. Qualquer incerteza na leitura pode limitar a precisão dos resultados obtidos.

Além disso, o fundo de escala espectral do *QE65000*, embora cubra de 200 nm a 1100 nm, foi limitado a 1000 nm no experimento devido às características do software e dos componentes envolvidos no processamento dos dados (OCEAN OPTICS INC., 2007). Mesmo que essa faixa seja relativamente ampla, os sensores LMR, dependendo das características do filme fino (espessura e tipo de material), podem apresentar ressonâncias em comprimentos de onda superiores a 1000 nm. Assim, eventos de ressonância que ocorram fora do limite superior do espectrômetro não serão detectados, levando à perda de informações valiosas ou até mesmo à impossibilidade de calibração adequada para certas configurações de sensores.

Essas limitações têm implicações práticas importantes, como a equação de calibração que estabelece a relação entre o deslocamento do comprimento de onda e o SRI, imprecisões nessa etapa podem comprometer as medições subsequentes, especialmente se o sensor LMR for utilizado em campo para estimativas precisas de propriedades físico-químicas de amostras.

Diante dessas limitações, considera-se que o espectrômetro utilizado poderia ser substituído por alternativas que ofereçam um intervalo espectral maior, permitindo a cobertura de comprimentos de onda superiores a 1000 nm. Essa melhoria possibilitaria uma melhor caracterização dos sensores LMR e ampliaria sua aplicabilidade em diferentes faixas de índice de refração. No entanto, qualquer substituição deve garantir que tais equipamentos forneçam os dados no mesmo padrão do software da *Ocean Optics*® (como o Spectra Suite), para manter a compatibilidade com o script de processamento desenvolvido neste trabalho.

Portanto, embora o espectrômetro utilizado tenha atendido aos objetivos propostos neste trabalho, a consideração dessas limitações é essencial para futuras aplicações práticas e para o aprimoramento da metodologia de calibração dos sensores LMR.

5.2 Trabalhos futuros e aplicabilidade dos sensores LMR

As principais linhas de pesquisa recomendadas para serem seguidas diante deste trabalho são:

- **Aprimoramento do script para monitoramento on-line**: integrar diretamente o software ao espectrômetro eliminando etapas manuais de importação de dados e assim desenvolver um protocolo de aquisição contínua que permita a monitoração em tempo real; e otimizar o desempenho computacional do código para torná-lo mais leve;
- **Detecção específica de gases dissolvidos**: associar os sensores LMR a outros métodos e sensores para análise de gases dissolvidos no óleo isolante dos transformadores, de modo a complementar a avaliação das condições do óleo isolante em uma única plataforma de sensoriamento;
- **Avaliação da resposta dos sensores LMR em óleos envelhecidos**: submeter os sensores a amostras de óleo isolante de transformadores com

histórico de falhas conhecidas ou envelhecidos de forma controlada, a fim de avaliar o desempenho e sensibilidade dos sensores para auxiliar na identificação destas falhas;

- **Calibração considerando variação de temperatura e análise de sensibilidade cruzada**: realizar calibrações dos sensores LMR sob diferentes temperaturas, investigando o efeito da variação térmica no comprimento de onda de ressonância e, consequentemente, na curva de calibração. Além disso, estudar a sensibilidade cruzada entre variações de índice de refração e temperatura. O script desenvolvido, no formato atual, não executa diretamente este tipo de análise cruzada, mas com etapas adicionais de desenvolvimento poderia ser adaptado para integrar essa funcionalidade.
- **Incorporação da propagação de incertezas no processo de calibração**: adicionar ao fluxo de calibração a estimativa e propagação dos erros associados às medições e ao ajuste matemático, de modo que o relatório final e as curvas de calibração apresentem, além dos parâmetros de sensibilidade e coeficiente de determinação, os intervalos de confiança e a incerteza expandida. Essa abordagem aumentaria a robustez metrológica do método e permitiria comparações mais precisas entre diferentes sensores ou condições de ensaio.

Tais trabalhos futuros não apenas aprofundariam a compreensão dos mecanismos dos sensores LMR em diferentes ambientes, mas também viabilizariam a aplicação prática destes sensores em sistemas de monitoração preditiva de transformadores, contribuindo para redução de custos operacionais e aumento da confiabilidade do sistema elétrico.

REFERÊNCIAS

ARREGUI, Francisco J.; et al. Fiber-optic lossy mode resonance sensors. **Procedia Engineering**, v. 87, p. 3-8, 2014.

ABNT. **ABNT NBR 5356**: Transformadores de Potência. Rio de Janeiro: ABNT, 2007.

ABNT. **ABNT NBR 7274:** Interpretação da análise dos gases de transformadores em serviço. Rio de Janeiro: ABNT, 2012.

BARBOSA, Fábio Rocha. **Monitoramento on-line e diagnóstico inteligente da qualidade dielétrica do isolamento líquido de transformadores de potência**. 2008. 147 f. Dissertação (Mestrado em Engenharia Elétrica) – Centro de Tecnologia, Universidade Federal do Ceará, Fortaleza, 2008.

BARBOSA, Fábio Rocha.; *et al*. Diagnóstico de transformadores de potência utilizando a dualidade entre os ensaios físico-químico e cromatográfico. In: ENCONTRO REGIONAL IBERO-AMERICANO DO CIGRÉ, 15., 2013, Foz do Iguaçu. **Anais [...]** Foz do Iguaçu: Itaipu Binacional, 2013. p. 1-8

BASSAN, Renato.; *et al*. Desenvolvimento de sistema de monitoramento óptico de multiparâmetros para transformadores de potência. *In*: SEMINÁRIO NACIONAL DE PRODUÇÃO E TRANSMISSÃO DE ENERGIA ELÉTRICA, 25., 10 a 13 de novembro de 2019, Belo Horizonte. **Anais [...]** Belo Horizonte: SNPTEE, 2019.

BASTOS, Gilson.; *et al*., Avaliação de Desempenho de Transformadores de Potência e Reatores no Sistema Elétrico Brasileiro. **CIGRE A2**, V.2, 2013.

BECHARA, Ricardo. **Análise de falhas de transformadores de potência**. 2010. 118 f. Dissertação (Mestrado em Sistemas de Potência) - Escola Politécnica, Universidade de São Paulo, São Paulo, 2010.

BENETTI, Daniel. **Estudo de viabilidade para aplicação de sensores distribuídos de deformação a fibra óptica em transformadores de potência**. 2016. 175 f. Dissertação (Doutorado em Mestre em Desenvolvimento de Tecnologia) – Institutos Lactec, Curitiba, 2016.

BIOSYSTEMS. Refratômetro Abbe modelo WYA-2WAJ DIGI. Disponível em: https://www.biosystemsimportadora.com.br/produtos/refratometro-abbe-com-escalas-para-indice-de-refracao-1-3000-1-7000-nd-com-precisao-de-00005-nd-e-brix-0-95-com-precisao-de-01-indicador-da-temperatura-por-termometro-digital-sem-compensacao-automatica-de-temperatura-modelo-wya-2waj-digi/. Acesso em: 17 maio 2025.

CARDOSO, P. M. **Adaptação de um sistema de medição de gases dissolvidos em óleo mineral isolante para monitoração de múltiplos transformadores de potência**. 2005. 123 f. Dissertação (Mestrado em Engenharia Elétrica) - Universidade Federal de Santa Catarina, Fortaleza, 2005.

CAVACO, Marco. A. M.; *et al*. Desenvolvimento e Avaliação de um Múltiplo Analisador de Gás Dissolvido em Óleo de Transformadores de Potência Utilizando

Comunicação GPRS1. In: SEMINÁRIO NACIONAL DE DISTRIBUIÇÃO DE ENERGIA ELÉTRICA, 18., 06 a 10 de outubro, Olinda, Pernambuco, Brasil, 2008. **Anais [...]** Olinda: SENDI, 2008.

CHAPMAN, S J. **Fundamentos de máquinas elétricas**.5 ed. Porto Alegre: AMGH, 2013.

DREYER, U. J. **Sensoriamento ótico distribuído e quase-distribuído para o setor de energia**. 2018. 162 f. Tese (Doutorado em Engenharia Elétrica e Informática Industrial) - Universidade Tecnológica Federal do Paraná, Curitiba, 2018.

DREYER, U. J.; *et al*. Gas Detection Using LMR-Based Optical Fiber Sensors. *In*: **Proceedings**. MDPI, 2018. p. 890.

FONTANA, E.; MARTINS FILHO, J. F. Sistemas Sensores em Desenvolvimento no Grupo de Fotônica do DES-UFPE. **Revista de Tecnologia da Informação e Comunicação**, v. 1, n. 1, p. 20-27, 2011.

FOROS, J.; ISTAD, M. Health index, risk and remaining lifetime estimation of power transformers. **IEEE Transactions on Power Delivery**, v. 35, n. 6, p. 2612–2620, 2020.

FUENTES, O.; *et al*. Simultaneous Measurement of Refractive Index and Temperature using LMR on planar waveguide. In: **2020 IEEE SENSORS**. IEEE, 2020. p. 1-4.

GODINHO, Mariana da Silva. **Avaliação da Degradação do Sistema Isolante Óleo-Papel usando Análise de Imagens e Técnicas Espectroscópicas combinadas com Métodos de Calibração Multivariada e Resolução de Curvas**. 2014. 274 f. Tese (Doutorado em Química) – Universidade Federal de Goiás, Goiânia, 2014.

GUIMARÃES, J. M. C. **Desenvolvimento de um sensor de descargas parciais e uma metodologia para priorização de investimento em sensores**. 2022. 117 f. Tese (Doutorado em Engenharia Elétrica) - Universidade Federal de Itajubá, Itajubá, 2022.

IEEE STANDARDS ASSOCIATION. ***Fiber Optics Sensors Standards Report***. New York: The Institute of Electrical and Electronics Engineers, 2017. Disponível em: https://standards.ieee.org. Acesso em: 31 maio 2025.

IMAS, J. J.; *et al*. Optical fiber sensors based on lossy mode resonances (LMRs): Fundamentals and recent developments. In: **2023 23rd International Conference on Transparent Optical Networks (ICTON)**. IEEE, 2023. p. 1-4.

INTERNATIONAL ELECTROTECHNICAL COMMISSION**. *IEC 61757:2018**: Fibre optic sensors – Generic specification***. Genebra: IEC, 2018.

JAIN, V.; *et al*. The Gaussian-Lorentzian Sum, Product, and Convolution (Voigt) functions in the context of peak fitting X-ray photoelectron spectroscopy (XPS) narrow scans. **Applied Surface Science**, v. 447, p. 548-553, 2018.

JESUS, E. O.; COSTA, R., Jr. **A utilização de filtros gaussianos na análise de imagens digitais**. SBMAC, 2015.

LIN, Y.; CHEN, L. Development of a Temperature-Controlled Optical Planar Waveguide Sensor with Lossy Mode Resonance for Refractive Index Measurement. In: **Photonics**. MDPI, 2021. p. 199.

MAHANTA, D. K.; LASKAR, S. Investigation of transformer oil breakdown using optical fiber as sensor. **IEEE Transactions on Dielectrics and Electrical Insulation**, v. 25, n. 1, p. 316-320, 2018.

MAMEDE, J. Filho. **Manual de Equipamentos Elétricos**.3 ed. Rio de Janeiro: LTC 2011.

MARTIGNONI, Afonso. **Transformadores**.6 ed. Rio de Janeiro: Globo, 1983.

MEITEI, S. N.; *et al*. Review on monitoring of transformer insulation oil using optical fiber sensors. **Results in Optics**, v. 10, p. 100361, 2023.

MORAES, R.; *et al*. A computational algorithm for determining the parameters of a spectral line. **NOTAS TÉCNICAS**, v. 12, n. 3, 2022.

MUKHERJEE, A.; *et al*. Characterization of a fiber optic liquid refractive index sensor. **Sensors and Actuators B: Chemical**, v. 145, n. 1, p. 265-271, 2010.

MUNIZ FILHO, J.; *et al*. Desenvolvimento de Sensor Infravermelho para Detecção on-line de Gases Dissolvidos em Óleo Isolante de Transformadores. *In*: V Congresso de Inovação Tecnológica em Energia Elétrica (V CITENEL), 22 a 24 de junho de 2009, Belém, PA. **Anais [...]** Belém: ANEEL, 2009.

NACARATO, R. L. **Investigação de Falha em Transformadores de Potência pela Análise dos Gases Dissolvidos no Óleo Isolante Utilizando Simulação em Redes Neurais Artificiais**. 2018. 82 f. Dissertação (Mestrado em Engenharia Mecânica) – Universidade Santa Cecília, Santos, 2018.

OCEAN OPTICS INC**. *QE65000 Scientific-grade Spectrometer: installation and operation manual***. Dunedin, FL: Ocean Optics Inc., 2007.

PALITÓ, T. T. C. **Metodologia acústica para análise de óleo de transformador por sensores piezoelétricos**. 2019. Tese (Doutorado em Sistemas Elétricos de Potência) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2019.

PALIWAL, N.; JOHN, J. Lossy mode resonance (LMR) based fiber optic sensors: A review. **IEEE sensors journal**, v. 15, n. 10, p. 5361–5371, 2015.

SAINI, R.; *et al*. Analysis of graphene coated optical fiber for visible range refractive index sensing. **Optics Communications**, v. 529, p. 129097, 2023.

SCHAFER, R. What is a Savitzky-Golay filter?. **IEEE signal processing magazine**, v. 28, n. 4, p. 111–117, 2011.

SCIPY. **SciPy Documentation**. 2025. Disponível em: https://docs.scipy.org/doc/scipy/index.html. Acesso em: 30 jan. 2025.

SILVA, L. T. S. **Diagnóstico de Falhas em Transformadores de Potência Utilizando Sensores de Gás Semicondutores**. 2012. 99 f. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal de Sergipe, São Cristovão, 2012.

VILLAR, Ignacio Del.; *et al*. Generation of lossy mode resonances by deposition of high-refractive-index coatings on uncladded multimode optical fibers. **Journal of Optics**, v. 12, n. 9, p. 095503, 2010.

APÊNDICE A - Código fonte do script desenvolvido em Python

```
import matplotlib.pyplot as plt
import tkinter as tk
from tkinter import filedialog, messagebox, simpledialog
import numpy as np
import pandas as pd
import os
from scipy.signal import savgol_filter, find_peaks, peak_widths
from scipy.optimize import minimize
import mplcursors
from sklearn.metrics import r2_score
from joblib import Parallel, delayed
import re


# Variáveis globais
dados_base = None
dados_reais = None
contador_cliques = 0
range_x_min = None
range_x_max = None
suavizar_dados = None
pico = 0 # Variável para determinar qual pico de ressonância será utilizado
grafico_check = pd.DataFrame()


# Função para plotar gráficos estáticos com legenda ordenada alfabeticamente
def plotar_grafico(dados, titulo):

    global grafico_check
    global range_x_min
    global range_x_max

    fig, ax = plt.subplots()
    fig.canvas.manager.set_window_title(titulo)

    # Lista para armazenar os dados antes de plotar
    series = []

    # Configurando os eixos do gráfico
    for column in dados.columns[1:]:
        # Substitui os nomes das colunas por descrições mais claras
        column_legenda = column.replace("SCA", "Sensor com água-
glicerina").replace("LBSS", "Fonte de luz (AQ4305) sem sensor")

        # Adiciona a legenda formatada na lista de séries
        series.append((column_legenda, dados.iloc[:, 0], dados[column]))

    # Ordena as séries pelos valores numéricos na legenda (se houver)
```

```
    series.sort(key=lambda x: [int(n) if n.isdigit() else n for n in
re.findall(r'\d+|\D+', x[0])])

    # Agora, plota as séries ordenadas corretamente
    for legenda, x, y in series:
        ax.plot(x, y, label=legenda)

    if range_x_min is not None and range_x_max is not None:
        ax.set_xlim(range_x_min, range_x_max)

    ax.set_title(f'{titulo}')
    ax.set_xlabel('Comprimento de onda (nm)')
    ax.set_ylabel('Intensidade')
    ax.legend()  # Adiciona a legenda ordenada corretamente

    plt.show()

    return fig


# Função para plotar gráficos de pontos conectados por linhas com legenda
ordenada
def plotar_grafico_pontos(dados, titulo):
    fig, ax = plt.subplots(figsize=(max(10, len(dados.columns)*0.5), 6))  #
Ajusta largura conforme quantidade de colunas
    fig.canvas.manager.set_window_title(titulo)

    legenda_metodos = {
        0: "Sem Tratamento",
        1: "Gaussiana",
        3: "Lorentziana",
        5: "Pseudo Voigt"
    }

    series = []
    colunas_ordenadas = sorted(dados.columns[1:], key=lambda x: [int(n) if
n.isdigit() else n for n in re.findall(r'\d+|\D+', x)])

    for index, row in dados.iterrows():
        y_values = [row[col] for col in colunas_ordenadas]
        x_curva = list(range(len(colunas_ordenadas)))
        metodo_legenda = legenda_metodos.get(index, "Outro")
        series.append((metodo_legenda, x_curva, y_values))

    series.sort(key=lambda x: x[0])

    legendas_adicionadas = set()
    for legenda, x, y in series:
        label_legenda = f'Dados experimentais - {legenda}'
```

```
        if label_legenda not in legendas_adicionadas:
            line, = ax.plot(x, y, marker='o', label=label_legenda)
            legendas_adicionadas.add(label_legenda)
        else:
            line, = ax.plot(x, y, marker='o')
        mplcursors.cursor(line, hover=True)

    # Verifica se os rótulos estão se sobrepondo
    ax.set_xticks(range(len(colunas_ordenadas)))
    ax.set_xticklabels(colunas_ordenadas, rotation=0, ha='center')
    fig.canvas.draw()
    labels = ax.get_xticklabels()

    overlap_detected = False
    for i in range(len(labels) - 1):
        bbox1 = labels[i].get_window_extent()
        bbox2 = labels[i + 1].get_window_extent()
        if bbox1.overlaps(bbox2):
            overlap_detected = True
            break

    # Se houver sobreposição, quebra os textos longos
    if overlap_detected:
        colunas_formatadas = []
        for label in colunas_ordenadas:
            if len(label) > 20:
                # Tenta quebrar o texto em blocos separados por espaços ou
hífens
                palavras = re.split(r'[\s\-]', label)
                colunas_formatadas.append("\n".join(palavras))
            else:
                colunas_formatadas.append(label)
        ax.set_xticklabels(colunas_formatadas, ha='center')

    ax.set_title(titulo)
    ax.set_xlabel('Nomes dos Arquivos')
    ax.set_ylabel('Comprimento de Onda Pico [nm]')
    ax.grid()
    ax.legend()

    fig.tight_layout()
    plt.show()
    return fig


# Função para plotar o gráfico da calibração do sensor
def plotar_dataframe(df, metodo):

    # Solicitar o erro do refratômetro, valor padrão de ±0,0002
```

```
    erro_refratometro = simpledialog.askstring("Erro do Refratômetro", "Insira
o erro do refratômetro (valor absoluto, ex: 0,0002):", initialvalue="0,0002")

    if erro_refratometro is None:
        return

    erro_refratometro = float(erro_refratometro.replace(',', '.'))

    # Extrai os índices do eixo X (colunas a partir da segunda)
    indices_x = df.columns[1:].astype(float)

    # Determinar título e subtítulo com base no método
    if metodo == '1':
        titulo = 'CURVA DE CALIBRAÇÃO - Dados sem tratamentos'
        aux_titulo = 'Sem tratamento'
    elif metodo == '2':
        titulo = 'CURVA DE CALIBRAÇÃO DO SENSOR LMR - Dados com filtro de
gaussiana'
        aux_titulo = 'Filtro de Gaussiana'
    elif metodo == '3':
        titulo = 'CURVA DE CALIBRAÇÃO DO SENSOR LMR - Dados com filtro de
lorentziana'
        aux_titulo = 'Filtro de Lorentziana'
    elif metodo == '4':
        titulo = 'CURVA DE CALIBRAÇÃO DO SENSOR LMR - Dados com filtro de
Pseudo Voigt'
        aux_titulo = 'Filtro de Pseudo Voigt'

    # Identificar linhas específicas no DataFrame
    linha_desvio_padrao = df[df.iloc[:, 0].str.contains("DESVIO", na=False,
case=False)]
    linha_comprimento_medio = df[df.iloc[:, 0].str.contains("- MÉDIO",
na=False, case=False)]
    sem_desvio = linha_desvio_padrao.empty or linha_desvio_padrao.iloc[0, 1]
== 'NaN'

    linhas_comprimento_pico = df[~df.iloc[:,
0].str.contains("MÉDIO|DESVIO|R²", na=False)]
    valores_comprimento_pico = linhas_comprimento_pico.iloc[:,
1:].astype(float).values
    valores_comprimento_medio = linha_comprimento_medio.iloc[:,
1:].astype(float).values.flatten()

    if not sem_desvio:
        valores_desvio_padrao = linha_desvio_padrao.iloc[:,
1:].astype(float).values.flatten()

    # Criar o gráfico
    plt.figure(figsize=(10, 6))
```

```
    for i, linha in enumerate(valores_comprimento_pico):
        legenda = 'Comprimento de onda PICO AMOSTRAS'
        plt.scatter(indices_x, linha, label=f'{legenda} {i + 1}')

    plt.plot(indices_x, valores_comprimento_medio, color='red',
label='Comprimento de onda PICO MÉDIOS')

    # Calcular a equação da curva de calibração dos pontos médios
    coefs = np.polyfit(indices_x, valores_comprimento_medio, 2)
    poly = np.poly1d(coefs)

    equacao = f"{coefs[0]:.4f}x²{'+' if coefs[1] >= 0 else
''}{coefs[1]:.4f}x{'+' if coefs[2] >= 0 else ''}{coefs[2]:.4f}"
    y_fit = poly(indices_x)
    r2 = r2_score(valores_comprimento_medio, y_fit)

    # Calcular o desvio padrão do eixo Y
    if not sem_desvio:
        erro_y = valores_desvio_padrao
        plt.errorbar(indices_x, valores_comprimento_medio, yerr=erro_y,
fmt='o', capsize=5, ecolor='red', markeredgecolor='red',
markerfacecolor='red', label='Desvio padrão da amostra')
    else:
        erro_y = None

    # Calcular o erro do refratômetro (valor absoluto)
    erro_indices_x = np.full_like(indices_x, erro_refratometro)
    plt.errorbar(indices_x, valores_comprimento_medio, xerr=erro_indices_x,
fmt='o', capsize=5, ecolor='blue', markeredgecolor='blue',
markerfacecolor='blue', label='Erro do Refratômetro')

    plt.plot(indices_x, y_fit, linestyle='--', color='purple',
label=f'Polinômio de 2ª ordem: {equacao}\nErro R²: {r2:.4f}')
    plt.xlabel('Índice de Refração')
    plt.ylabel('Comprimento de Onda PICO [nm]')
    plt.title(titulo)
    plt.legend()
    plt.grid(True)

    mplcursors.cursor(hover=True)

    pasta_relatorio = 'Relatório de Calibração'
    if not os.path.exists(pasta_relatorio):
        os.makedirs(pasta_relatorio)

    caminho_grafico = os.path.join(pasta_relatorio, 'grafico_calibracao.png')
    plt.savefig(caminho_grafico)
```

```
    caminho_equacao = os.path.join(pasta_relatorio, 'equacao_polinomial.txt')
    with open(caminho_equacao, 'w') as f:
        f.write(f'Polinômio de 2ª ordem: {equacao}\nErro R²: {r2:.4f}\n')
        f.write(f'Erro do Refratômetro: ±{erro_refratometro}\n')
        f.write(f'Utilizando o método de detecção de pico: {aux_titulo}\n')

    os.startfile(pasta_relatorio)

    plt.show()



# Função para normalizar os dados
def normalizar_dados(dados_base1):

    # Cria uma cópia do DataFrame para evitar a modificação do DataFrame
original
    dados_base_normalizados = dados_base1.copy()

    # Converte todo o DataFrame para float
    dados_base_normalizados = dados_base_normalizados.astype(float)

    # Calcula os valores mínimos e máximos uma vez para melhorar o desempenho
    minimo = np.min(dados_base_normalizados.iloc[:,1])
    maximo = np.max(dados_base_normalizados.iloc[:,1])

    # Normaliza os dados
    for i in range(len(dados_base_normalizados)):
        dados_base_normalizados.iloc[i,1] = (dados_base_normalizados.iloc[i,1]
- minimo) / (maximo - minimo)

    return dados_base_normalizados


# Função para verificar se os dados estão normalizados
def verficar_se_dados_sao_normalizados(dados):
    if (dados.iloc[:, 1] <= 1).all():
         return True
    else:
        return


# Função para definir/atualizar o sinal base
def definir_sinal_base():
    global dados_base
    caminho_pasta = os.path.join(os.getcwd(), 'Espectro Base (Espectro da
fonte de luz)')
    arquivos = os.listdir(caminho_pasta)
    arquivos_xlsx = [arq for arq in arquivos if arq.endswith('.xlsx')]
```

```
    if len(arquivos_xlsx) > 1:
        messagebox.showwarning("Sinal Base","Error!!!\nMais de um sinal base
detectado! Deve existir apenas um sinal na pasta!")

    if arquivos_xlsx:
        dados_base = pd.read_excel(os.path.join(caminho_pasta,
arquivos_xlsx[0]), index_col=None)

        normalizado=verficar_se_dados_sao_normalizados(dados_base)
        if normalizado:
            messagebox.showinfo("Sinal Base", "Sinal base carregado do arquivo
.xlsx (dados ja estão normalizados).")
        else:
            messagebox.showinfo("Normalizar Espectro base", "O arquivo .xlsx
não está normalizado. Normalizando...")
            dados_base = normalizar_dados(dados_base)

            messagebox.showinfo("Sinal Base", f"Sinal base carregado do
arquivo {os.path.basename(os.path.join(caminho_pasta, arquivos_xlsx[0]))}
agora normalizado.")
            return
    else:
        resposta = messagebox.askyesno("Carregar espectro base", "Adicione um
arquivo .xlsx na pasta 'Espectro Base (Espectro da fonte de luz)'\n\nDeseja
adicionar o arquivo .xlsx agora?")
        if resposta:
            caminho_pasta = os.path.join(os.getcwd(), 'Espectro Base (Espectro
da fonte de luz)')
            os.makedirs(caminho_pasta, exist_ok=True)
            filedialog.askopenfilename(initialdir=caminho_pasta,
title="Coloque o arquivo aqui",
                                       filetypes=[("Excel files", "*.xlsx")])
            messagebox.showinfo("Espectro Base", "Adicione o arquivo .xlsx na
pasta 'Espectro Base (Espectro da fonte de luz)' e selecione
'Definir/Atualizar Sinal Base' novamente para carregar.")


# Função para definir o sinal base automaticamente caso exista o sinal base ja
esteja no pasta base
def definir_sinal_base_autmatico ():
    global dados_base
    caminho_pasta = os.path.join(os.getcwd(), 'Espectro Base (Espectro da
fonte de luz)')
    arquivos = os.listdir(caminho_pasta)
    arquivos_xlsx = [arq for arq in arquivos if arq.endswith('.xlsx')]

    if arquivos_xlsx:
```

```
        dados_base = pd.read_excel(os.path.join(caminho_pasta,
arquivos_xlsx[0]), index_col=None)
        normalizado=verficar_se_dados_sao_normalizados(dados_base)
        if normalizado:
            return
        else:
            dados_base = normalizar_dados(dados_base)
            return
    else:
        return


# Função para aplicar o range ao eixo x
def aplicar_range_x():
    global range_x_min, range_x_max
    range_x_min_flash=None

    resposta = messagebox.askyesno("Range do Eixo X", "Deseja aplicar um range
personalizado ao eixo x? Caso selecione não volta para a situação Default")

    if resposta:
        range_x_min_flash=range_x_min
        range_x_min = simpledialog.askfloat("Range do Eixo X", "Valor
mínimo:\t\t\t\t", minvalue=0.0, maxvalue=10000.0)
        if range_x_min==None:
            return
        range_x_max = simpledialog.askfloat("Range do Eixo X", "Valor
máximo:\t\t\t\t", minvalue=range_x_min, maxvalue=10000.0)
        if range_x_max==None:
            range_x_min=range_x_min_flash
            return
    else:
        range_x_min = None
        range_x_max = None


# Função para criar o Master Gráfico!
def criar_grafico_mestre():

    global suavizar_dados
    global range_x_max
    global range_x_min
    global grafico_check

    resposta=messagebox.askyesno('Master Gráfico!','Deseja fazer o Master
Gráfico?\n\nEsta função tem como obejtivo pegar todos os sinais/dados
normalizados salvos, e os subtrir-los com sinal do Espectro Base, retornando
um único gráfico como resultado e os salvando na pasta "Resultados Master
Gráfico!"')
```

```
    if resposta:

        messagebox.showwarning('Master Gráfico!','inicializado...\n\nATENÇÃO!
Estão função apenas opera com dados normalizados!')

        nome_pasta = 'Dados salvos (normalizados)'
        caminho_pasta = os.path.join(os.getcwd(), nome_pasta)
        arquivos = os.listdir(caminho_pasta)
        arquivos_xlsx = [arq for arq in arquivos if arq.endswith('.xlsx')]

        # Verifica se existe um sinal base definido
        if dados_base is None:
            messagebox.showwarning("Aviso", "Defina um sinal base antes de
criar o Master Gráfico!.")
            return

        #Verifica se existe alguma base dados salvos na pasta selecionada
        if len(arquivos_xlsx)==0:
            messagebox.showwarning("Aviso", "Sem dados salvos para realizar o
Master Gráfico.")
            return


        # Prepara o DataFrame para o Master Gráfico!
        grafico_mestre = pd.DataFrame()
        grafico_mestre['Comprimento de onda [nm]'] = dados_base.iloc[:,0]


        for arquivo in arquivos_xlsx:
            dados = pd.read_excel(os.path.join(caminho_pasta, arquivo))
            dados.iloc[:,1] = dados_base.iloc[:,1] - dados.iloc[:,1]
            col_nome = dados.columns[1]
            dados[col_nome] = dados[col_nome].where(dados[col_nome] >= 0, 0)
            dados=normalizar_dados(dados)
            grafico_mestre[arquivo.replace('.xlsx', '')] = dados.iloc[:,1]


        grafico_check = grafico_mestre


        if suavizar_dados:

            grafico_mestre_suavizado , valor_r2 =
suavizar_grafico(grafico_mestre)

            resposta2=messagebox.askyesno('Master Gráfico!','Deseja salvar
dados suavizados?')
            if resposta2:
```

```
                if range_x_max and range_x_min:

                    grafico_mestre_reduzido = pd.DataFrame()
                    grafico_mestre_reduzido =
grafico_mestre_suavizado[grafico_mestre_suavizado[grafico_mestre_suavizado.col
umns[0]].between(range_x_min, range_x_max)].copy()

                    for column in grafico_mestre_reduzido.columns[1:]:
                        df = pd.DataFrame({grafico_mestre_reduzido.columns[0]:
grafico_mestre_reduzido[grafico_mestre_reduzido.columns[0]], column:
grafico_mestre_reduzido[column]})
                        df = normalizar_dados(df)
                        grafico_mestre_reduzido.loc[:, column] = df[column]


                # Salva o Master Gráfico! em um novo arquivo .xlsx
                caminho_pasta = os.path.join(os.getcwd(), 'Resultados Master
Gráfico!')
                os.makedirs(caminho_pasta, exist_ok=True)
                nome_arquivo_mestre = os.path.join(caminho_pasta, 'Master
Gráfico! (espectros resultantes).xlsx')
                if range_x_max and range_x_min:
                    grafico_mestre_reduzido.to_excel(nome_arquivo_mestre,
index=False)
                else:
                    grafico_mestre_suavizado.to_excel(nome_arquivo_mestre,
index=False)
            else:
                # Salva o Master Gráfico! em um novo arquivo .xlsx
                caminho_pasta = os.path.join(os.getcwd(), 'Resultados Master
Gráfico!')
                os.makedirs(caminho_pasta, exist_ok=True)
                nome_arquivo_mestre = os.path.join(caminho_pasta, 'Master
Gráfico! (espectros resultantes).xlsx')
                grafico_mestre.to_excel(nome_arquivo_mestre, index=False)
        else:
                # Salva o Master Gráfico! em um novo arquivo .xlsx
                caminho_pasta = os.path.join(os.getcwd(), 'Resultados Master
Gráfico!')
                os.makedirs(caminho_pasta, exist_ok=True)
                nome_arquivo_mestre = os.path.join(caminho_pasta, 'Master
Gráfico! (espectros resultantes).xlsx')
                grafico_mestre.to_excel(nome_arquivo_mestre, index=False)

        # Abre a pasta e mostra o arquivo salvo
        os.startfile(caminho_pasta)

        if suavizar_dados and range_x_max and range_x_min:
            # Plota o Master Gráfico!
```

```
            grafico_mestre_reduzido =
grafico_mestre_suavizado[grafico_mestre_suavizado[grafico_mestre_suavizado.col
umns[0]].between(range_x_min, range_x_max)].copy()
            plotar_grafico(grafico_mestre_reduzido, "Master Gráfico!")
        elif suavizar_dados:
            # Plota o Master Gráfico!
            plotar_grafico(grafico_mestre_suavizado, "Master Gráfico!")
        else:
            # Plota o Master Gráfico!
            plotar_grafico(grafico_mestre, "Master Gráfico!")
    else:
        return


# Função para criar as pastas que serão utilizadas pelo codigo caso elas não
existam
def criar_pasta_saves():
    # Criando Dados salvos (normalizados)
    caminho_pasta1 = os.path.join(os.getcwd(), 'Dados salvos (normalizados)')
    os.makedirs(caminho_pasta1, exist_ok=True)
    # Criando dados Dados para a Calibração
    caminho_pasta2 = os.path.join(os.getcwd(), 'Dados para a Calibração')
    os.makedirs(caminho_pasta2, exist_ok=True)
    # Criando Dados import
    caminho_pasta3 = os.path.join(os.getcwd(), 'Dados import')
    os.makedirs(caminho_pasta3, exist_ok=True)
    # Criando Espectro Base (Espectro da fonte de luz)
    caminho_pasta4 = os.path.join(os.getcwd(), 'Espectro Base (Espectro da
fonte de luz)')
    os.makedirs(caminho_pasta4, exist_ok=True)
    # Criando Relatorio final
    caminho_pasta5 = os.path.join(os.getcwd(), 'Relatório de Calibração')
    os.makedirs(caminho_pasta5, exist_ok=True)


# Função para importar dados
def importar_dados():
    caminho_pasta_import = os.path.join(os.getcwd(), 'Dados import')
    messagebox.showinfo("Importação de Dados", "Selecione os arquivos .xlsx
para importar. Eles podem ser normalizados ou não.")
    arquivos = filedialog.askopenfilenames(initialdir=caminho_pasta_import,
title="Selecione os arquivos",
                                           filetypes=[("Excel files",
"*.xlsx")])

    if  arquivos:
        for arquivo in arquivos:
            dados = pd.read_excel(os.path.join(caminho_pasta_import, arquivo))
            normalizado = verficar_se_dados_sao_normalizados(dados)
```

```
            if not normalizado:
                messagebox.showinfo("Normalização", f"O arquivo
{os.path.basename(arquivo)} não está normalizado. Normalizando...")
                dados = normalizar_dados(dados)

            # Salva o arquivo normalizado na pasta de Dados salvos
(normalizados)
            caminho_pasta_salvos = os.path.join(os.getcwd(), 'Dados salvos
(normalizados)')
            os.makedirs(caminho_pasta_salvos, exist_ok=True)
            dados.to_excel(os.path.join(f'{caminho_pasta_salvos}',os.path.base
name(arquivo)), index=False)

        # Apaga os arquivos da pasta de importação
        for arquivo in os.listdir(caminho_pasta_import):
            os.remove(os.path.join(caminho_pasta_import, arquivo))

        messagebox.showinfo("Importação de Dados", "Importação concluída com
sucesso. Os dados foram salvos em 'Dados salvos (normalizados)'.")
    else:
        messagebox.showwarning ("Importação de Dados","Erro na importação,
voltado ao menu...")
        return


# Função para suavizar as curvas (filtro Savitzky-Golay)
def suavizar_grafico(grafico):

    grafico_suavizado = grafico.copy()  # Cria uma cópia do DataFrame original

    valor_r2 = pd.DataFrame(columns=['Valores de R²'])

    # Aplicando o filtro Savitzky-Golay
    for column in grafico_suavizado.columns[1:]:
        tamanho_da_janela = 50  # Tamanho da janela do filtro
        polinomio = 4  # Ordem do polinômio
        col_y = grafico_suavizado[column]
        trava = 0

        while trava == 0:
            # Aplica o filtro de suavização
            try:
                grafico_suavizado[column] = savgol_filter(col_y,
window_length=tamanho_da_janela, polyorder=polinomio)
            except:
                polinomio -= 1
                tamanho_da_janela = 50  # Tamanho da janela do filtro

                if polinomio == 1:
```

```
                    messagebox.showinfo("Suavização", "Não foi possível
suavizar o gráfico. Tente novamente.")
                    print(f'{tamanho_da_janela}, {polinomio} : não conseguiu
suavizar a função não funcionou')
                    return grafico_suavizado, None

            # Tira os valores negativos
            grafico_suavizado[column] = np.where(grafico_suavizado[column] >=
0, grafico_suavizado[column], 0)

            # Verifica se o erro entre os gráficos é menor que a tolerância
            if erro_suavizacao(col_y, grafico_suavizado[column], grafico):
                trava = 1
            else:
                tamanho_da_janela -= 1  # Diminui o tamanho da janela do
filtro
                if tamanho_da_janela == 0:
                    messagebox.showinfo("Suavização", "Não foi possível
suavizar o gráfico. Tente novamente.")
                    print(tamanho_da_janela)
                    return grafico_suavizado, None

        valor_r2.loc[len(valor_r2)] = r2_score(col_y,
grafico_suavizado[column])



    if valor_r2.shape[0] == 1:
        valor_r2 = valor_r2.iloc[0, 0]
    elif valor_r2.empty:
        valor_r2 = None

    return grafico_suavizado , valor_r2


# Função para diminuinção do erro ao suavizar gráfico
def erro_suavizacao(dados_grafico_original, dados_grafico_suavizado, grafico,
tolerancia=0.8):

    # Calcula o erro entre os gráficos
    df_grafico_original = pd.DataFrame({grafico.columns[0]:
grafico[grafico.columns[0]], 'dados_grafico_orignal': dados_grafico_original})
    pico_original = encontrar_pico(df_grafico_original) # Encontra os picos do
gráfico original
    print(pico_original)

    df_grafico_suavizado = pd.DataFrame({grafico.columns[0]:
grafico[grafico.columns[0]], 'dados_grafico_suavizado':
dados_grafico_suavizado})
```

```
    pico_suavizado = encontrar_pico(df_grafico_suavizado) # Encontra os picos
do gráfico suavizado
    print(pico_suavizado)

    erro = np.abs(pico_original.loc[0,pico_original.columns[1]] -
pico_suavizado.loc[0,pico_suavizado.columns[1]]) # Calcula o erro entre os
picos

    print (erro)

    return None if erro > tolerancia else True


# Função para solicitar suavização dos resultados
def deseja_suavizar():
    global suavizar_dados
    if not suavizar_dados:
        resposta = messagebox.askyesno("Suavização do Resultados", "Deseja
aplicar um filtro de suavização aos resultados?\n\nNota: Esse filtro utiliza a
técnica de Savitzky-Golay")
        if resposta:
            suavizar_dados = 'True'
            return
        else:
            return
    if suavizar_dados:
        resposta2 = messagebox.askyesno("Suavização do Resultados", "Deseja
retirar o filtro de suavização aos resultados?")
        if resposta2:
            suavizar_dados = None
            return
        else:
            return


# Função para encontra os picos dos espectros para a suavização de gráficos
def encontrar_pico(df):

    pico_auxiliar = 0

    # Cria um novo DataFrame para armazenar os segundos picos
    segundos_picos = pd.DataFrame()

    # Capturando os comprimentos de onda
    comprimentos_de_onda = df['Comprimento de onda [nm]']

    # Itera por todas as colunas de intensidades
    segundos_picos.loc[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm]'
```

```
    for coluna in df.columns[1:]:  # Ignora a primeira coluna de comprimento
de onda
        intensidades = df[coluna]
        intensidades = intensidades.tolist()
        picos, _ =find_peaks(intensidades, height=0.90, prominence=0.5,
width=10, distance=10)
        # Pega o índice do segundo pico
        indice_segundo_pico = picos[pico_auxiliar]
        # Encontra o comprimento de onda correspondente ao segundo pico
        comprimento_de_onda_segundo_pico =
comprimentos_de_onda[indice_segundo_pico]
        segundos_picos[coluna] = comprimento_de_onda_segundo_pico


    return segundos_picos


# Função para encontra os picos de ressonância
def encontrar_pico_verdadeiro(df):

    # Verificar se df é uma tuple
    if isinstance(df, tuple):
        # Convertendo a tuple de volta para DataFrame
        # Supondo que a tuple tenha o DataFrame como primeiro elemento
        df = pd.DataFrame(df[0])

    # Cria um novo DataFrame para armazenar os segundos picos
    segundos_picos = pd.DataFrame()
    comprimentos_de_onda = pd.DataFrame()

    # Capturando os comprimentos de onda
    comprimentos_de_onda = df['Comprimento de onda [nm]']

    # Itera por todas as colunas de intensidades
    segundos_picos.loc[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm]'

    for coluna in df.columns[1:]:  # Ignora a primeira coluna de comprimento
de onda
        intensidades = df[coluna]
        intensidades = intensidades.tolist()
        picos, _ = find_peaks(intensidades, height=0.9, prominence=0.5,
width=10, distance=10) # height=0 para encontrar todos os picos acima de 0.007

        # Pega o índice do segundo pico
        indice_segundo_pico = picos[pico]
        # Encontra o comprimento de onda correspondente ao segundo pico
```

```
        comprimento_de_onda_segundo_pico =
comprimentos_de_onda[indice_segundo_pico]
        segundos_picos[coluna] = comprimento_de_onda_segundo_pico

    return segundos_picos


# Função para testes para a detecção de picos atraves dos metodos propostos
def teste_de_pico():

    resposta=messagebox.askyesno('Teste de detecção de picos','Deseja o teste
de detecção de picos dos dados salvos?\n\nEsta função tem como obejtivo pegar
todos os sinais/dados normalizados salvos, e os subtrir-los com sinal base, em
sequência é aplicado o filtro desejado e por final é será plotado um gráfico
demonstrando os picos detectados como resultado. Os dados, também, serão
salvos na pasta "indice de refracao" em formato .xlsx ')
    if resposta:

        # Buscando os dados dentro da pasta "Dados salvos (normalizados)"
        nome_pasta = 'Dados salvos (normalizados)'
        caminho_pasta = os.path.join(os.getcwd(), nome_pasta)
        arquivos = os.listdir(caminho_pasta)
        arquivos_xlsx = [arq for arq in arquivos if arq.endswith('.xlsx')]

        # Verifica se existe um sinal base definido
        if dados_base is None:
            messagebox.showwarning("Aviso", "Defina um sinal base antes o
Teste de detecção de picos.")
            return

        #Verifica se existe alguma base dados salvos na pasta selecionada
        if len(arquivos_xlsx)==0:
            messagebox.showwarning("Aviso", "Sem dados salvos para determinar
o Teste de detecção de picos.")
            return

        # Prepara o DataFrame para o gráfico teste de picos
        valores_de_pico = pd.DataFrame()
        resultados_auxliar = pd.DataFrame()
        resultados_auxliar['Comprimento de onda [nm]'] = dados_base.iloc[:,0]


        # Capturando os dados salvos na pasta 'Dados salvos (normalizados)'
        for arquivo in arquivos_xlsx:
            if arquivo:
                dados = pd.read_excel(os.path.join(caminho_pasta, arquivo))

                if not verficar_se_dados_sao_normalizados(dados):
```

```
                    messagebox.showinfo("Teste de detecção de picos",
f"Error!!!\n\nOs dados salvos na pasta 'Dados salvos (normalizados)' não estão
normalizados, encerrando a função e voltando ao menu...")
                    return

                dados = pd.read_excel(os.path.join(caminho_pasta, arquivo))
                dados.iloc[:,1] = dados_base.iloc[:,1] - dados.iloc[:,1]
                col_nome = dados.columns[1]
                dados[col_nome] = dados[col_nome].where(dados[col_nome] >= 0,
0)
                dados=normalizar_dados(dados)
                resultados_auxliar[arquivo.replace('.xlsx', '')] =
dados.iloc[:,1]


        # Pergunta ao usuário qual método de calibração deseja utilizar
        opcao = simpledialog.askstring("Teste de detecção de picos", "Digite o
número da opção do método de detecção desejado:\n1 - Determinação de picos sem
tratamento de dados\n2 - Determinação de picos através do filtro Gaussiano\n3
- Determinação de picos através do filtro Lorentziano\n4 - Determinação de
picos através do filtro de Pseudo Voigt (Gaussiano + Lorentziano)\n5 -
Comparação entre todos os metodos")

        if opcao not in ['1', '2', '3', '4', '5']:
            messagebox.showwarning("Aviso", "Método de detecção inválido.")
            return

        # Encontrando os valores de pico puros
        if opcao=='1':
            valores_de_pico = encontrar_pico_verdadeiro(resultados_auxliar)
            valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda
pico [nm] - Sem tratamento'
            titulo_aux = "Gráfico Teste Detecção de Picos - Método: Sem
tratamentos"


        # Encontrando os valores de picos atraves da extraploção com a
GAUSSIANAS
        if opcao=='2':
            valores_de_pico = gaussiana(resultados_auxliar)
            valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda
pico [nm] - Gaussiana'
            valores_de_pico.at[1,'Nomes dos arquivos'] = 'Erro R² - Gaussiana'
            titulo_aux = "Gráfico Teste Detecção de Picos - Método: Filtro de
Gaussiana"


        # Encontrando os valores de picos atraves da extraploção com a
LORENTZIANAS
```

```
        if opcao=='3':
            valores_de_pico = lorentziana(resultados_auxliar)
            valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda
pico [nm] - Lorentziana'
            valores_de_pico.at[1,'Nomes dos arquivos'] = 'Erro R² -
Lorentziana'
            titulo_aux = "Gráfico Teste Detecção de Picos - Método: Filtro de
Lorentziana"


        # Encontrando os valores de picos atraves da extraploção com a Pseudo
Voigt
        if opcao=='4':
            valores_de_pico = pseudo_voigt(resultados_auxliar)
            valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda
pico [nm] - Pseudo Voigt'
            valores_de_pico.at[1,'Nomes dos arquivos'] = 'Erro R² - Pseudo
Voigt'
            titulo_aux = "Gráfico Teste Detecção de Picos - Método: Filtro de
Pseudo Voigt"


        # Comparação entre os 3 metodos de encontrar picos
        if opcao=='5':

            valores_de_pico_5 = pseudo_voigt(resultados_auxliar)
            valores_de_pico_5.at[0,'Nomes dos arquivos'] = 'Comprimento de
onda pico [nm] - Pseudo Voigt'
            valores_de_pico_5.at[1,'Nomes dos arquivos'] = 'Erro R² - Pseudo
Voigt'

            valores_de_pico_1 = encontrar_pico_verdadeiro(resultados_auxliar)
            valores_de_pico_1.at[0,'Nomes dos arquivos'] = 'Comprimento de
onda pico [nm] - Sem tratamento'

            valores_de_pico_3 = gaussiana(resultados_auxliar)
            valores_de_pico_3.at[0,'Nomes dos arquivos'] = 'Comprimento de
onda pico [nm] - Gaussiana'
            valores_de_pico_3.at[1,'Nomes dos arquivos'] = 'Erro R² -
Gaussiana'

            valores_de_pico_4 = lorentziana(resultados_auxliar)
            valores_de_pico_4.at[0,'Nomes dos arquivos'] = 'Comprimento de
onda pico [nm] - Lorentziana'
            valores_de_pico_4.at[1,'Nomes dos arquivos'] = 'Erro R² -
Lorentziana'

            titulo_aux = "Gráfico Teste Detecção de Picos - Método: Comparação
de Métodos"
```

```
            valores_de_pico = pd.concat([valores_de_pico_1, valores_de_pico_3,
valores_de_pico_4, valores_de_pico_5], ignore_index=True)


        # Salva o gráfico com os indice de refração em um novo arquivo .xlsx
        caminho_pasta = os.path.join(os.getcwd(), 'Determinando picos')
        os.makedirs(caminho_pasta, exist_ok=True)
        nome_arquivo_picos = os.path.join(caminho_pasta, 'Valores testes de
pico.xlsx')
        valores_de_pico.to_excel(nome_arquivo_picos, index=False)


        # Abre a pasta e mostra o arquivo salvo
        os.startfile(caminho_pasta)


        if opcao in ['2', '3', '4']:
            valores_de_pico = valores_de_pico.drop(1)

        if opcao in ['5']:
            valores_de_pico = valores_de_pico.drop([2, 4, 6])

        # Plota o gráfico com os indices de refração
        plotar_grafico_pontos(valores_de_pico, titulo_aux)

        return
    else:
        return


# Função para a detecção de picos atraves da gaussiana
def gaussiana(df):

    global pico

    # Função para a curva gaussiana
    def gaussian(x, a, b, c):
        return a * np.exp(-((x - b) ** 2) / (2 * c ** 2))

    # Função de custo para a otimização
    def gauss_fit(params, x, y):
        amplitude, mean, stddev = params
        y_fit = gaussian(x, amplitude, mean, stddev)
        return -r2_score(y, y_fit)

    def process_column(x, y, pico, rel_height_range):
        # Encontrar os picos
```

```
        peaks, _ = find_peaks(y, height=1, prominence=0.5, width=10,
distance=10)

        # Calcular a prominência e a largura dos picos
        widths_result = peak_widths(y, peaks, rel_height=(1/2))
        left_ips = widths_result[2]
        right_ips = widths_result[3]

        # Dados do pico escolhido
        peak_x = x[peaks[pico]]
        peak_y = y[peaks[pico]]
        left_base = x[int(left_ips[pico])]
        right_base = x[int(right_ips[pico])]

        # Otimização da meia altura para minimizar o erro R²
        melhor_r2 = -np.inf
        melhor_params = None

        # Vetorização da criação da máscara e otimização para melhorar a
performance
        mask = np.zeros_like(x, dtype=bool)
        initial_guess = [peak_y, peak_x, (right_base - left_base) / 2]

        for rel_height in rel_height_range:  # Ajuste de 0.4 a 0.7
            half_height = peak_y * rel_height
            mask = (x >= left_base) & (x <= right_base) & (y >= half_height)
            x_above_half = x[mask]
            y_above_half = y[mask]

            if len(x_above_half) > 1:  # Verificar se existem pontos
suficientes para o ajuste
                try:
                    result = minimize(gauss_fit, initial_guess,
args=(x_above_half, y_above_half), method='L-BFGS-B')
                    popt = result.x
                    r2 = -result.fun

                    # Considerar a altura e posição do pico original no
cálculo do R²
                    if r2 > melhor_r2 and abs(popt[0] - peak_y) / peak_y < 0.1
and abs(popt[1] - peak_x) / peak_x < 0.05:
                        melhor_r2 = r2
                        melhor_params = popt
                except RuntimeError:
                    continue

        # Usar os parâmetros ajustados para definir a gaussiana
        gaussian_peak_x = melhor_params[1]
```

```
        return gaussian_peak_x, melhor_r2


    # Cria um novo DataFrame para armazenar os segundos picos
    segundos_picos = pd.DataFrame()

    # Capturando os comprimentos de onda
    comprimentos_de_onda = df['Comprimento de onda [nm]'].values

    # Itera por todas as colunas de intensidades
    segundos_picos.loc[0, 'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm]'
    segundos_picos.loc[1, 'Nomes dos arquivos'] = 'Erro R² (Gaussiana)'

    # Ajuste fino no intervalo de busca para manter precisão
    rel_height_range = np.linspace(0.40, 0.70, 75)

    # Utilizando joblib para paralelizar o processamento das colunas
    resultados = Parallel(n_jobs=-
1)(delayed(process_column)(comprimentos_de_onda, df[coluna].values, pico,
rel_height_range) for coluna in df.columns[1:])

    # Preencher os resultados no DataFrame
    for (coluna, (gaussian_peak_x, melhor_r2)) in zip(df.columns[1:],
resultados):
        segundos_picos.loc[0, coluna] = round(gaussian_peak_x, 2)
        segundos_picos.loc[1, coluna] = melhor_r2

    print(f"{segundos_picos}")

    return segundos_picos


# Função para a detecção de picos atraves da lorentziana
def lorentziana(df):

    global pico

    # Função para a curva Lorentziana
    def lorentzian(x, amplitude, mean, hwhm):
        return amplitude * (hwhm**2) / ((x - mean)**2 + hwhm**2)

    # Função de custo para a otimização
    def lorentz_fit(params, x, y):
        amplitude, mean, hwhm = params
        y_fit = lorentzian(x, amplitude, mean, hwhm)
        return -r2_score(y, y_fit)

    def process_column(x, y, pico, rel_height_range):
```

```
        # Encontrar os picos
        peaks, _ = find_peaks(y, height=1, prominence=0.5, width=10,
distance=10)

        # Calcular a prominência e a largura dos picos
        widths_result = peak_widths(y, peaks, rel_height=(1/2))
        left_ips = widths_result[2]
        right_ips = widths_result[3]

        # Dados do pico escolhido
        peak_x = x[peaks[pico]]
        peak_y = y[peaks[pico]]
        left_base = x[int(left_ips[pico])]
        right_base = x[int(right_ips[pico])]

        # Otimização da meia altura para minimizar o erro R²
        melhor_r2 = -np.inf
        melhor_params = None

        # Vetorização da criação da máscara e otimização para melhorar a
performance
        mask = np.zeros_like(x, dtype=bool)
        initial_guess = [peak_y, peak_x, (right_base - left_base) / 2]

        for rel_height in rel_height_range:  # Ajuste de 0.4 a 0.7
            half_height = peak_y * rel_height
            mask = (x >= left_base) & (x <= right_base) & (y >= half_height)
            x_above_half = x[mask]
            y_above_half = y[mask]

            if len(x_above_half) > 1:  # Verificar se existem pontos
suficientes para o ajuste
                try:
                    result = minimize(lorentz_fit, initial_guess,
args=(x_above_half, y_above_half), method='L-BFGS-B')
                    popt = result.x
                    r2 = -result.fun

                    # Considerar a altura e posição do pico original no
cálculo do R²
                    if r2 > melhor_r2 and abs(popt[0] - peak_y) / peak_y < 0.1
and abs(popt[1] - peak_x) / peak_x < 0.05:
                        melhor_r2 = r2
                        melhor_params = popt
                except RuntimeError:
                    continue

        # Usar os parâmetros ajustados para definir a Lorentziana
        lorentzian_peak_x = melhor_params[1]
```

```
        return lorentzian_peak_x, melhor_r2

    # Cria um novo DataFrame para armazenar os segundos picos
    segundos_picos = pd.DataFrame()

    # Capturando os comprimentos de onda
    comprimentos_de_onda = df['Comprimento de onda [nm]'].values

    # Itera por todas as colunas de intensidades
    segundos_picos.loc[0, 'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm]'
    segundos_picos.loc[1, 'Nomes dos arquivos'] = 'Erro R² (Lorentziana)'

    # Ajuste fino no intervalo de busca para manter precisão
    rel_height_range = np.linspace(0.40, 0.70, 75)

    # Utilizando joblib para paralelizar o processamento das colunas
    resultados = Parallel(n_jobs=-
1)(delayed(process_column)(comprimentos_de_onda, df[coluna].values, pico,
rel_height_range) for coluna in df.columns[1:])

    # Preencher os resultados no DataFrame
    for (coluna, (lorentzian_peak_x, melhor_r2)) in zip(df.columns[1:],
resultados):
        segundos_picos.loc[0, coluna] = round(lorentzian_peak_x, 2)
        segundos_picos.loc[1, coluna] = melhor_r2

    print(f"{segundos_picos}")

    return segundos_picos


# Função para a detecção de picos atraves da Pseudo Voigt
def pseudo_voigt(df):

    global pico

    # DataFrame para armazenar os segundos picos
    segundos_picos = pd.DataFrame(columns=['Nomes dos arquivos'] +
list(df.columns[1:]))
    segundos_picos.loc[0, 'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm]'
    segundos_picos.loc[1, 'Nomes dos arquivos'] = 'Erro R² (Pseudo Voigt)'

    # Captura os comprimentos de onda
    comprimentos_de_onda = df['Comprimento de onda [nm]'].values

    # Funções para Gaussiana, Lorentziana e Pseudo-Voigt internas
```

```
    def gaussian(x, amplitude, mean, stddev):
        return amplitude * np.exp(-((x - mean) ** 2) / (2 * stddev ** 2))

    def lorentzian(x, amplitude, mean, hwhm):
        return amplitude * (hwhm**2) / ((x - mean)**2 + hwhm**2)

    def pseudo_voigt_curve(x, amplitude_g, mean_g, stddev_g, amplitude_l,
mean_l, hwhm_l, m):
        return m * lorentzian(x, amplitude_l, mean_l, hwhm_l) + (1 - m) *
gaussian(x, amplitude_g, mean_g, stddev_g)

    def find_best_combination_fit(x, y, peak_y, peak_x, left_base,
right_base):
        melhor_r2 = -np.inf
        melhor_combined_y = None
        melhor_params = None

        # Filtra os dados dentro do intervalo [left_base, right_base]
        x_range_mask = (x >= left_base) & (x <= right_base)
        x_filtered = x[x_range_mask]
        y_filtered = y[x_range_mask]


        for rel_height in np.linspace(0.4, 0.7, 75): # Ajuste de 0.4 a 0.7
            half_height = peak_y * rel_height

            # Cria a máscara para os pontos acima da metade da altura
            mask = y_filtered >= half_height

            if np.sum(mask) > 1:  # Verifica se há pontos suficientes
                x_above_half = x_filtered[mask]
                y_above_half = y_filtered[mask]

                # Estimativa inicial dos parâmetros para a minimização
                initial_guess = [
                    peak_y,  # Amplitude da Gaussiana
                    (left_base + right_base) / 2,  # Média da Gaussiana
                    (right_base - left_base) / 4,  # Desvio padrão da
Gaussiana
                    peak_y,  # Amplitude da Lorentziana
                    (left_base + right_base) / 2,  # Média da Lorentziana
                    (right_base - left_base) / 4,  # Largura da Lorentziana
                    0.5  # Parâmetro de mistura 'm'
                ]

                # Limites para os parâmetros durante a minimização devido a
otimização do tempo de processamento
                bounds = [
                    (0.95 * peak_y, 1.2 * peak_y),  # Amplitude da Gaussiana
```

```
                    (left_base, right_base),        # Média da Gaussiana
                    (0.1 * (right_base - left_base), 2 * (right_base -
left_base)),  # Desvio padrão da Gaussiana
                    (0.95 * peak_y, 1.2 * peak_y),  # Amplitude da Lorentziana
                    (left_base, right_base),        # Média da Lorentziana
                    (0.1 * (right_base - left_base), 2 * (right_base -
left_base)),  # Largura da Lorentziana
                    (0, 1)                          # Parâmetro de mistura
                ]

                try:
                    # Executa a minimização para encontrar os melhores
parâmetros
                    result = minimize(
                        lambda params:
np.sum((pseudo_voigt_curve(x_above_half, *params) - y_above_half) ** 2),
                        initial_guess,
                        method='L-BFGS-B',
                        bounds=bounds
                    )
                    popt = result.x
                    mean_y_above_half = np.mean(y_above_half)
                    r2 = 1 - result.fun / np.sum((y_above_half -
mean_y_above_half) ** 2)

                    if (r2 > melhor_r2 and abs(popt[0] - peak_y) / peak_y <
0.1 and abs(popt[1] - peak_x) / peak_x < 0.05): # Diferença máxima de 10% na
altura e 5% na posição
                        melhor_r2 = r2
                        melhor_params = popt
                except RuntimeError:
                    continue

        if melhor_params is not None:
            melhor_combined_y = pseudo_voigt_curve(x, *melhor_params)

        return melhor_combined_y, melhor_r2, melhor_params

    def process_column(coluna, df, comprimentos_de_onda, pico):
        y = df[coluna].values

        # Encontrar os picos
        peaks, _ = find_peaks(y, height=1, prominence=0.5, width=10,
distance=10)

        if len(peaks) > pico:
            widths_result = peak_widths(y, peaks, rel_height=0.5)
            left_ips = widths_result[2]
            right_ips = widths_result[3]
```

```
            peak_x = comprimentos_de_onda[peaks[pico]]
            peak_y = y[peaks[pico]]
            left_base = comprimentos_de_onda[int(left_ips[pico])]
            right_base = comprimentos_de_onda[int(right_ips[pico])]

            combined_y_adjusted, melhor_r2, _ = find_best_combination_fit(
                comprimentos_de_onda, y, peak_y, peak_x, left_base, right_base
            )

            peak_voigt_ajusted =
float(comprimentos_de_onda[np.argmax(combined_y_adjusted)]) if
combined_y_adjusted is not None else None

            return coluna, peak_voigt_ajusted, melhor_r2

        return coluna, None, None

    # Paralelização com joblib para otimização de processamento
    results = Parallel(n_jobs=-1)(delayed(process_column)(coluna, df,
comprimentos_de_onda, pico) for coluna in df.columns[1:])

    # Atualizando os resultados no DataFrame
    for coluna, comprimento_pico, erro_r2 in results:
        if comprimento_pico is not None:
            segundos_picos.loc[0, coluna] = comprimento_pico
            segundos_picos.loc[1, coluna] = erro_r2

    print(segundos_picos)
    return segundos_picos



# Função para elaborar o gráfico de calibração do sensor
def calibracao():
    resposta = messagebox.askyesno('Calibração do Sensor LMR!', 'Deseja
definir a calibração do sensor LMR?\n\nEsta função tem como objetivo,
determinar uma equação polinomial de segunda ordem que melhor descreva o
comportamento do sensor LMR correlacionado o índice de refração medido no
processo de calibração e o comprimento de onda pico obtido através do
processamento dos dados do sensor LMR.')
    if resposta:
        # Pergunta ao usuário qual método de calibração deseja utilizar
        metodo = simpledialog.askstring("Método de Calibração", "Digite o
número da opção do método de detecção para realizar a calibração, desejado:\n1
- Determinação de picos sem tratamento de dados\n2 - Determinação de picos
através do filtro de Gaussiana\n3 - Determinação de picos através do filtro de
Lorentziana\n4 - Determinação de picos através do filtro de Pseudo Voigt
(Gaussiana + Lorentziana)")
```

```
        if metodo not in ['1', '2', '3', '4']:
            messagebox.showwarning("Aviso", "Método de calibração inválido.")
            return

        nome_pasta = 'Dados para a Calibração'
        caminho_pasta = os.path.join(os.getcwd(), nome_pasta)
        arquivos = os.listdir(caminho_pasta)
        arquivos_xlsx = [arq for arq in arquivos if arq.endswith('.xlsx')]

        # Verifica se existe um sinal base definido
        if dados_base is None:
            messagebox.showwarning("Aviso", "Defina um sinal base antes de
realizar a calibração do sensor LMR.")
            return

        # Verifica se existe algum Espectro Base (Espectro da fonte de luz)
salva na pasta selecionada
        if len(arquivos_xlsx) == 0:
            messagebox.showwarning("Aviso", "Sem dados salvos para realizar a
calibração do sensor LMR.")
            return

        # Dicionário para armazenar os dataframes por índice de refração
        dataframes_por_indice = {}

        # Percorre todos os arquivos .xlsx na pasta
        for nome_arquivo in arquivos_xlsx:
            caminho_arquivo = os.path.join(caminho_pasta, nome_arquivo)
            df = pd.read_excel(caminho_arquivo, usecols=[0, 1], header=0)  #
Lendo apenas as primeiras duas colunas

            nome_sem_extensao = os.path.splitext(nome_arquivo)[0]  # Remove a
extensão .xlsx
            indice_ref = nome_sem_extensao.split(' ')[0]  # Pega o índice de
refração (parte antes do espaço)
            indice_ref = indice_ref.replace(',', '.')

            if indice_ref not in dataframes_por_indice:
                dataframes_por_indice[indice_ref] = []  # Inicializa a lista
para o índice de refração
            dataframes_por_indice[indice_ref].append(df)  # Adiciona o
dataframe à lista correspondente

        # Inicializar DataFrames corretamente
        colunas = ['IR'] + list(dataframes_por_indice.keys())
        valores_de_pico_medios = pd.DataFrame(columns=colunas)
        valores_de_r2 = pd.DataFrame(columns=colunas)
```

```
        if metodo == '1':
            metodo_aux = 'Comprimento de onda PICO [nm] - Sem tratamento'
            metodo_aux2 = 'Comprimento de onda PICO [nm] - Sem tratamento -
MÉDIO'
            metodo_aux3 = 'Erro R² médio das aproximações - Sem tratamento'
            titulo = 'Curva de Calibração do Sensor LMR - MÉTODO: SEM
TRATAMENTOS'
        elif metodo == '2':
            metodo_aux = 'Comprimento de onda PICO [nm] - Filtro de Gaussiana'
            metodo_aux2 = 'Comprimento de onda PICO [nm] - Filtro de Gaussiana
- MÉDIO'
            metodo_aux3 = 'Erro R² médio das aproximações - Filtro Gaussiana'
            titulo = 'Curva de Calibração do Sensor LMR - MÉTODO: FILTRO DE
GAUSSIANA'
        elif metodo == '3':
            metodo_aux = 'Comprimento de onda PICO [nm] - Filtro de
Lorentziana'
            metodo_aux2 = 'Comprimento de onda PICO [nm] - Filtro de
Lorentziana - MÉDIO'
            metodo_aux3 = 'Erro R² médio das aproximações - Filtro
Lorentziana'
            titulo = 'Curva de Calibração do Sensor LMR - MÉTODO: FILTRO DE
LORENTZIANA'
        elif metodo == '4':
            metodo_aux = 'Comprimento de onda PICO [nm] - Filtro de Pseudo
Voigt'
            metodo_aux2 = 'Comprimento de onda PICO [nm] - Filtro de Pseudo
Voigt - MÉDIO'
            metodo_aux3 = 'Erro R² médio das aproximações - Filtro Pseudo
Voigt'
            titulo = 'Curva de Calibração do Sensor LMR - MÉTODO: FILTRO DE
PSEUDO VOIGT'

        # Função auxiliar para paralelizar o processamento de cada dataframe
        def process_df(df, indice):
            resultado_pico, resultado_R2 =
calibracao_determinando_pico(metodo, df)
            return indice, resultado_pico, resultado_R2

        # Preparar os dados para processamento paralelo
        all_dataframes = [(df, indice) for indice, dataframes in
dataframes_por_indice.items() for df in dataframes]

        # Usar joblib para paralelizar o processamento
        resultados = Parallel(n_jobs=-1)(delayed(process_df)(df, indice) for
df, indice in all_dataframes)

        print (resultados)
```

```
        passo = 0
        i = 0

        # Processar os resultados e preencher os DataFrames
        indices_unicos = []

        for r in resultados:
            if r[0] not in indices_unicos:
                indices_unicos.append(r[0])

        colunas = ['IR'] + indices_unicos
        valores_de_pico_medios = pd.DataFrame(columns=colunas)

        passo = 0  # controla a linha do DataFrame


        # Agrupamento dos dados
        agrupado_pico = {indice: [] for indice in indices_unicos}
        agrupado_r2 = {indice: [] for indice in indices_unicos}
        for indice, valor, r2 in resultados:
            agrupado_pico[indice].append(valor)
            if r2 is not None:
                agrupado_r2[indice].append(r2)

        max_linhas = max(len(v) for v in agrupado_pico.values())

        # Construir linhas com os valores de pico por índice
        for i in range(max_linhas):
            nova_linha = {'IR': metodo_aux}
            for indice in indices_unicos:
                if i < len(agrupado_pico[indice]):
                    nova_linha[indice] = agrupado_pico[indice][i]
            valores_de_pico_medios.loc[passo] = nova_linha
            passo += 1

        # Adicionar linha de R² médio (se houver)
        if metodo_aux != 'Comprimento de onda PICO [nm] - Sem tratamento':
            linha_r2 = {'IR': metodo_aux3}
            for indice in indices_unicos:
                r2s = agrupado_r2[indice]
                if r2s:
                    linha_r2[indice] = sum(r2s) / len(r2s)
            valores_de_pico_medios.loc[passo] = linha_r2
            passo += 1

        # Linha de média
        linha_media = {'IR': metodo_aux2}
        for indice in indices_unicos:
            linha_media[indice] = np.mean(agrupado_pico[indice])
```

```
        valores_de_pico_medios.loc[passo] = linha_media
        passo += 1

        # Linha de desvio padrão
        linha_std = {'IR': 'DESVIO PADRÃO DA AMOSTRA'}
        for indice in indices_unicos:
            linha_std[indice] = np.std(agrupado_pico[indice], ddof=1) if
len(agrupado_pico[indice]) > 1 else 0.0
        valores_de_pico_medios.loc[passo] = linha_std


        print(valores_de_pico_medios)


        # Buscando os dados dentro da pasta "Relatório de Calibração"
        nome_pasta = 'Relatório de Calibração'
        caminho_pasta = os.path.join(os.getcwd(), nome_pasta)
        arquivos = os.listdir(caminho_pasta)
        arquivos_xlsx = [arq for arq in arquivos if
arq.endswith('_media.xlsx')]

        # Salva o gráfico com os índices de refração em um novo arquivo .xlsx
        os.makedirs(caminho_pasta, exist_ok=True)
        nome_arquivo_picos = os.path.join(caminho_pasta, 'Relatório de
Calibração.xlsx')
        valores_de_pico_medios.to_excel(nome_arquivo_picos, index=False)

        plotar_dataframe(valores_de_pico_medios, metodo)

        return
    else:
        return


#Função para determinar os picos para a calibração do sensor
def calibracao_determinando_pico(metodo, dados):

    if not verficar_se_dados_sao_normalizados(dados):
        dados=normalizar_dados(dados)

    dados.iloc[:,1] = dados_base.iloc[:,1] - dados.iloc[:,1]
    col_nome = dados.columns[1]
    dados[col_nome] = dados[col_nome].where(dados[col_nome] >= 0, 0)
    dados=normalizar_dados(dados)

    dados.columns = ["Comprimento de onda [nm]", 'Intensidades']

    resultado_R2 = None
```

```
    # Encontrando os valores de pico puros
    if metodo=='1':
        valores_de_pico = encontrar_pico_verdadeiro(dados)
        valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm] - Sem tratamento -medio'
        resultado_pico = valores_de_pico.iloc[0,1]

    # Encontrando os valores de picos atraves da extraploção com a GAUSSIANAS
    if metodo=='2':
        valores_de_pico = gaussiana(dados)
        valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm] - Gaussiana'
        resultado_pico = valores_de_pico.iloc[0,1]
        resultado_R2 = valores_de_pico.iloc[1,1]

    # Encontrando os valores de picos atraves da extraploção com a
LORENTZIANAS
    if metodo=='3':
        valores_de_pico = lorentziana(dados)
        valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm] - Lorentziana'
        resultado_pico = valores_de_pico.iloc[0,1]
        resultado_R2 = valores_de_pico.iloc[1,1]

    # Encontrando os valores de picos atraves da extraploção com a PSEUDO
VOIGT
    if metodo=='4':
        valores_de_pico = pseudo_voigt(dados)
        valores_de_pico.at[0,'Nomes dos arquivos'] = 'Comprimento de onda pico
[nm] - Pseudo Voigt'
        resultado_pico = valores_de_pico.iloc[0,1]
        resultado_R2 = valores_de_pico.iloc[1,1]


    return resultado_pico,resultado_R2


# Função para mostrar os créditos do programa
def mostrar_creditos(event):
    global contador_cliques
    contador_cliques += 1
    if contador_cliques == 3:
        messagebox.showinfo("Créditos", "Programa elaborado por Matheus
Osvaldo Camargo\n\nMOC SYSTENS\nEsse código não tem garantias de nada :)")
        contador_cliques = 0



# Cria a janela principal
```

```
root = tk.Tk()
root.title("Menu do Espectrômetro")

# Ajuste do tamanho da janela para mostrar o título completo e todo conteudo
necessário
quantidade_de_funcoes = 8 # Ajuste para dimensionamento das dos botões das
funções
largura = 400
altura = quantidade_de_funcoes*25 + 25 # +25 por conta dos créditos

# Calculo das coordenadas para centralizar a janela
largura_tela = root.winfo_screenwidth()
altura_tela = root.winfo_screenheight()
x = (largura_tela - largura) // 2
y = (altura_tela - altura) // 2

# Defina a posição da janela
root.geometry(f"{largura}x{altura}+{x}+{y}")

# Cria os botões e chama as funções
btn_definir_base = tk.Button(root, text="Definir Fonte de Luz Base",
command=definir_sinal_base)
btn_definir_base.pack(fill=tk.X)

btn_importar_dados = tk.Button(root, text="Importar Dados",
command=importar_dados)
btn_importar_dados.pack(fill=tk.X)

btn_aplicar_range_x = tk.Button(root, text="Aplicar Range ao Eixo x
(Comprimento de onda [nm])", command=aplicar_range_x)
btn_aplicar_range_x.pack(fill=tk.X)

btn_suavizar_dados_x = tk.Button(root, text="Filtro de Suavização",
command=deseja_suavizar)
btn_suavizar_dados_x.pack(fill=tk.X)

btn_criar_grafico_mestre = tk.Button(root, text="Master Gráfico!",
command=criar_grafico_mestre)
btn_criar_grafico_mestre.pack(fill=tk.X)

btn_teste_de_pico = tk.Button(root, text="Teste de Detecção de Picos",
command=teste_de_pico)
btn_teste_de_pico.pack(fill=tk.X)

btn_teste_de_pico_CALIBRACAO = tk.Button(root, text="Calibração Sensor LMR",
command=lambda: calibracao())
btn_teste_de_pico_CALIBRACAO.pack(fill=tk.X)

btn_sair = tk.Button(root, text="Sair", command=root.destroy)
```

```
btn_sair.pack(fill=tk.X)

# Label para MOC SYSTENS
label_moc = tk.Label(root, text="MOC SYSTENS", font=("Arial", 7))
label_moc.pack(side=tk.BOTTOM)
label_moc.bind("<Button-1>", mostrar_creditos)

# Criando pastas nescessárias
criar_pasta_saves()

# Testando se ja existe dados de base na pasta
definir_sinal_base_autmatico()

# Inicia a GUI
root.mainloop()
```