UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO DE CIÊNCIAS FÍSICAS E MATEMÁTICAS
PROGRAMA DE PÓS-GRADUAÇÃO EM FÍSICA

Beatriz Nattrodt D'Avila

Desenvolvimento de programa para análise automatizada de dados: Um estudo de caso do desbastamento eletrônico de filmes poliméricos por espectrometria de massa

Florianópolis
2024

Beatriz Nattrodt D'Avila

Desenvolvimento de programa para análise automatizada de dados: Um estudo de caso do desbastamento eletrônico de filmes poliméricos por espectrometria de massa

Dissertação submetida ao Programa de Pós-Graduação em Física da Universidade Federal de Santa Catarina para a obtenção do título de mestre em Física.
Orientador: Prof. Dr. Igor Alencar Vellame

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

D'Avila, Beatriz Nattrodt
Desenvolvimento de programa para análise automatizada de dados : um estudo de caso do desbastamento eletrônico de filmes poliméricos por espectrometria de massa / Beatriz Nattrodt D'Avila ; orientador, Igor Alencar Vellame, 2024.
118 p.

Dissertação (mestrado) - Universidade Federal de Santa Catarina, Centro de Ciências Físicas e Matemáticas, Programa de Pós-Graduação em Física, Florianópolis, 2024.

Inclui referências.

1. Física. 2. Software. 3. Python. 4. Danos por irradiação. 5. Física Atômica e Molecular. I. Vellame, Igor Alencar. II. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Física. III. Título.

Beatriz Nattrodt D'Avila

Desenvolvimento de programa para análise automatizada de dados: Um estudo de caso do desbastamento eletrônico de filmes poliméricos por espectrometria de massa

O presente trabalho em nível de mestrado foi avaliado e aprovado por banca examinadora composta pelos seguintes membros:

Prof. Dr. Igor Alencar Vellame
Universidade Federal de Santa Catarina

Prof. Dr. Marcelo Girardi Schappo
Instituto Federal de Santa Catarina

Prof. Dr. Henrique Trombini
Universidade Federal de Ciências da Saúde de Porto Alegre

Certificamos que esta é a **versão original e final** do trabalho de conclusão que foi julgado adequado para obtenção do título de mestre em Física.

Documento assinado digitalmente
Paulo Henrique Souto Ribeiro
Data: 17/06/2024 10:36:58-0300
CPF: ***.886.506-**
Verifique as assinaturas em https://v.ufsc.br

Coordenação do Programa de
Pós-Graduação

Documento assinado digitalmente
Igor Alencar Vellame
Data: 17/06/2024 10:09:41-0300
CPF: ***.145.918-**
Verifique as assinaturas em https://v.ufsc.br

Prof. Dr. Igor Alencar Vellame
Orientador

Florianópolis, 2024.

À minha esposa e aos meus gatos.

AGRADECIMENTOS

Chegar ao final desta dissertação foi um processo de muita importância na minha vida profissional, mas sem o apoio de pessoas importantes, seria para mim uma missão pouco possivel. E a elas dedico este trabalho e meus sinceros agradecimentos.

Primeiramente à minha esposa Karolayne, por toda ajuda e por todo apoio durante todos esses anos que nos conhecemos. Escrever esta dissertação foi uma jornada desafiadora, e nos momentos de dúvida e exaustão, você esteve ao meu lado, oferecendo palavras de encorajamento. Obrigada por estar comigo, meu amor por você só cresce a cada dia.

Aos meus gatos, Judith, Foucault, Simone e Madalena, pela companhia durante todos os momentos importantes dos últimos anos. Vocês estiveram sempre presentes, com suas ronronadas reconfortantes e seus olhares curiosos.

Aos meus pais por acreditarem nos meus sonhos e por me deram o apoio necessário para concluí-los e por se fazerem presentes mesmo na distância. Sem vocês essa realização seria apenas uma vontade.

Ao meu orientador Prof. Dr. Igor Alencar Vellame, por aceitar me orientar, pela paciência, ajuda e incentivo durante o processo de todas as escritas que fizemos, em especial esta dissertação.

Aproveito a oportunidade para agradecer ao Programa de Pós-Graduação em Física da UFSC pelo auxílio e facilitações nas burocracias acadêmicas, assim como à FAPESC, agência financiadora do meu mestrado.

Por fim, agradeço aos professores componentes da banca avaliadora por disponibilizarem seu tempo e conhecimento para contribuir com a ciência brasileira.

RESUMO

A física moderna se depara com fenômenos complexos que necessitam de experimentação detalhada e precisa, produzindo experimentos que geram uma quantidade massiva de dados. A coleta, armazenamento e análise dos dados gerados por esses experimentos são essenciais para os avanços tecnológicos que auxiliam na compreensão do universo. Estudos sobre os efeitos da radiação sobre os materiais são essenciais para entender os mecanismos e aplicações da radiação. Entretanto, devido a imensa quantidade de dados a serem analisados em um curto período de tempo disponível, torna-se inviável analisar individualmente cada dado gerado. Diante dessa realidade, o presente trabalho buscou desenvolver um programa em linguagem computacional PYTHON que fosse capaz de avaliar experimentos de degradação de polímeros por feixes de íons no regime eletrônico de deposição de energia. Como estudo de caso, investigou-se experimentos com a técnica de espectrometria de massa de íons secundários em medidas do tempo-de-voo, realizados com íons primários de Cloro e Cobre com energias cinéticas variando na faixa entre 6,0 e 12,0 MeV incididos sobre amostras de filmes poliméricos de Poliestireno, Polimetilmetacrilato e Cloreto de Polivinila. O programa foi construído utilizando Programação Orientada a Objeto, abstraindo conceitos matemáticos e de linguagem computacional, buscando tornar a experiência do usuário mais eficiente e intuitiva. O programa é capaz de identificar picos nos espectros, encontrar suas contagens e tempos de voo, calcular a área sob cada pico, além de permitir a identificação das moléculas iônicas emitidas por meio de comparação com o tempo-de-voo esperado através de calibração de massa, recebendo como parâmetro apenas o caminho nos quais os espectros a serem analisados estão salvos no computador. Para este estudo de caso em questão, a identificação das moléculas permitiu também observar três grupos moleculares nos diferentes alvos: hidrocarbonetos, óxidos e cloretos. Em sequência, o programa determinou a seção de choque para danos e seu respectivo raio ajustando a quantidade de íons secundários coletados em função do fluxo de íons primários em espectros de contagem por tempo-de-voo adquiridos sequencialmente. Assim, através dos resultados obtidos pelo programa foi possível realizar análises desses valores para a seção de choque nos três grupos moleculares estudados, permitindo identificar o grupo com maior e menor resistência à radiação, sendo estes os grupos dos óxidos e cloretos, respectivamente. Outra comparação possibilitada pelos resultados do programa foi o valor médio dos raios em cada grupo de elementos. Uma vez comparado com trabalhos que utilizaram técnicas de Microscopia de Varredura por Força, os valores de raios obtidos no presente estudo são uma ordem de grandeza inferiores aos raios reportados na literatura indicando uma baixa eficiência na ejeção de íons secundários. Por fim, comparando os valores de seção de choque encontrados com os reportados com a técnica Espectroscopia de Fotoelétrons excitados por Raios X, pode-se notar que os valores obtidos estão uma ordem de magnitude acima, indicando assim que o polímero permanece inerte quimicamente com exceção dos cloretos. Um estudo mais aprofundado para compreender a variação dessas grandezas com a perda de energia dos íons primários no regime eletrônico é necessária para confirmar tal hipóteses.

Palavras-chave: Programa; PYTHON; Danos por irradiação; Desbastamento eletrônico; Espectrometria de massa de íons secundários por tempo-de-voo; Polímeros.

ABSTRACT

Modern physics is faced with complex phenomena that require detailed and precise experimentation, producing experiments that generate massive amounts of data. Collecting, storing and analyzing the data generated by these experiments is essential for technological advances that help us understand the universe. Studies on the effects of radiation on materials are essential for understanding the mechanisms and applications of radiation. Due to the immense amount of data to be analyzed in a short period of available time available, it becomes unfeasible to analyze each piece of data generated individually. In view of this reality, the present research sought to develop a software in PYTHON language that would be capable of evaluating polymer degradation experiments using ion beams in the electronic energy deposition regime. As a case study, experiments with the time-of-flight secondary ion mass spectrometry technique were investigated. They were carried out with primary Chlorine and Copper ions with kinetic energies varying between 6.0 and 12.0 MeV, impinging on samples of polystyrene, polymethylmethacrylate and polyvinyl chloride polymer films. The software was built using Object-Oriented Programming, abstracting mathematical and computer language concepts in order to make the user experience more efficient and intuitive. The software is capable of identifying peaks in the spectra, finding their counts and times-of-flight, calculating the area under each peak, as well as allowing the identification of the ionic molecules emitted by comparing them with the time-of-flight expected through mass calibration, receiving as a parameter only the path in which the spectra to be analyzed are saved on the computer. For this case study, the molecular identification also made it possible to observe the behaviour of three molecular groups present in the different targets: hydrocarbons, oxides and chlorides. The software then determined the cross section for damage and its respective radius by modeling the amount of secondary ions collected as a function of the primary ion fluxes in sequentially recorded time-of-flight spectra. Thus, using the results obtained by the software, it was possible to analyze these cross section values for the three molecular groups studied, making it possible to identify the group with the highest and lowest resistance to radiation, these being the oxide and chloride groups, respectively. Another comparison made possible by the software results was the average value of the radii. When compared with studies using Scanning Force Microscopy techniques, the radius values obtained in this study are an order of magnitude lower than the radii reported in the literature, indicating a low efficiency in the ejection of secondary ions. Finally, comparing the cross section values found with those reported in the literature for X-ray Photoemission Spectroscopy, it can be seen that the results are an order of magnitude higher, thus indicating that the polymer remains chemically inert with the exception of chlorides. A deeper investigation in order to understand the relation of these parameters withe the ion energy loss in the electronic regime is necessary to validate these hypotheses.

Keywords: Software; PYTHON; Radiation damage; Electronic sputtering; Time-of-flight secondary ion mass spectrometry; Polymers.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

C	Carbono
Cl	Cloro
H	Hidrogênio
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LII	Laboratório de Implantação Iônica
MS	Espectrometria de Massa (*Mass Spectrometry*)
O	Oxigênio
PMMA	Polimetilmetacrilato
POO	Programação Orientada a Objeto
PS	Poliestireno
PUC-RS	Pontifícia Universidade Católica do Rio Grande do Sul
PVC	Cloreto de Polivinila
SFM	Microscopia de Varredura por Força (*Scanning Force Microscopy*)
Si	Silício
SIMS	Espectrometria de Massa de Íons Secundários (*Secondary Ion Mass Spectrometry*)
ToF-SIMS	Espectrometria de Massa de Íons Secundários por Tempo-de-voo (*Time-of-flight Secondary Ion Mass Spectrometry*)
UFRGS	Universidade Federal do Rio Grande do Sul
XPS	Espectroscopia de Fotoelétrons excitados por Raios X (*X-ray Photoelectron Spectroscopy*)

LISTA DE SÍMBOLOS

eV	Elétron-Volt
keV	Quilo elétron-Volt
MeV	Mega elétron-Volt
m	Massa
z	Carga
$^{\circ}$C	Graus Celsius
g	Grama
cm	Centímetro
nm	Nanomêtro
mm	Milimetro
μs	Micro segundo
ns	Nano segundo
s	Segundo
σ	Seção de Choque
φ	Fluxo

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem por motivação a construção de um programa para investigação da degradação causada por feixes de íons no regime eletrônico de deposição como consequência do processo de desbastamento para três tipos diferentes de polímeros – Poliestireno (PS) , Polimetilmetacrilato (PMMA) e Cloreto de Polivinila (PVC) – através da análise de espectros gerados em experimentos de espectrometria de massa por íons secundários em medidas do tempo-de-voo realizados no Laboratório de Implantação Iônica (LII) da Universidade Federal do Rio Grande do Sul (UFRGS). Os experimentos de degradação consistem em adquirir sequencialmente espectros de tempo-de-voo durante a irradiação contínua dos alvos com íons primários pesados de alta energia, porém não relativísticos, gerando centenas de espectros contendo centenas de picos de emissão de íons secundários. Deste modo, evidência-se a necessidade de automatizar a análise para lidar com tal quantidade de dados e, portanto, um programa para esta finalidade foi desenvolvido.

O programa desenvolvido no presente trabalho foi empregado na análise de irradiações realizadas em oito condições diferentes, descritas no Quadro 1, e para cada uma delas seguiu-se o fluxo: (i) identificação dos picos de emissão de íons secundários em todos os espectros adquiridos; (ii) comparação dos picos de emissão observados com uma série de picos escolhidos como referência; (iii) ajuste de função exponencial para a curva gerada pela comparação; (iv) cálculo da seção de choque para danos por irradiação e seu respectivo raio utilizando os parâmetros da função exponencial ajustada; e, por fim, (v) identificação das possíveis moléculas emitidas emcada picos e agrupamento dos parâmetros obtidos (seção de choque e raio) em grupos moleculares (hidrocarbonetos, óxidos e cloretos).

Deste modo, para uma apresentação mais concisa e objetiva, estruturou-se esta dissertação da maneira descrita a seguir. O segundo capítulo apresenta uma breve revisão bibliografica sobre desbastamento, espectrometria de massa e degradação em polímeros, além da motivação e objetivo. O terceiro capítulo traz uma descrição

Quadro 1 – Irradiações de filmes poliméricos com feixe de íons.

Experimento	Amostra	Energia (MeV)	Feixe
PMMA_6_MeV_Cu	polimetilmetacrilato	6,0	Cobre
PS_12_MeV_Cl	poliestireno	12,0	Cloro
PMMA_12_MeV_Cl	polimetilmetacrilato	12,0	Cloro
PS_9_MeV_Cl	poliestireno	9,0	Cloro
PMMA_9_MeV_Cl	polimetilmetacrilato	9,0	Cloro
PS_6_MeV_Cl	poliestireno	6,0	Cloro
PS_6_MeV_Cu	poliestireno	6,0	Cobre
PVC_6_MeV_Cu	cloreto de polivinila	6,0	Cobre

das amostras, dos experimentos realizados e da linguagem computacional utilizada na construção do software. O quarto capítulo apresenta a documentação do programa desenvolvido para a análise automatizada dos espectros. O quinto capítulo discuste os resultados obtidos. Finalmente, o sexto e último capítulo mostra conclusões e perspectivas futuras do trabalho. Cabe notar que, nas figuras de autoria própria, a fonte foi omitida.

2 BREVE REVISÃO BIBLIOGRÁFICA

2.1 DESBATAMENTO

Dentre inúmeros fenômenos que ocorrem ao bombardear um sólido com partículas, como a retrodifusão de íons incidentes com átomos-alvo, implantação de íons incidentes no sólido e danos por radiação na superfície do sólido, encontra-se o fenômeno de desbastamento (BEHRISCH; ECKSTEIN, 2007).

Este processo consiste na ejeção de partículas de um material, podendo estas serem neutras ou carregadas, ocasionando a modificação da morfologia da superfície do material a partir do bombardeamento de sua superfície por radiação, utilizando como projéteis íons, átomos ou aglomerados (URBASSEK, 1997).

De acordo com Behrisch e Eckstein (2007), diferentes faixas de energia causam efeitos distintos no processo de desbastamento. Para energias na faixa de 100 eV a keV, o processo de desbastamento físico é causado pelo efeito cascata – transferência de momento e energia do íon incidente a partir de sua distribuição entre os átomos do alvo até que um átomo perto da superfície tenha energia e momento suficientemente maiores que a energia de ligação superficial, conseguindo assim ser ejetado da superfície do material. Já para faixas de energia de MeV, a energia depositada nos elétrons do alvo também auxilia no processo de desbastamento.

As colisões entre as partículas incidentes com energias na faixa de keV com os átomos da superfície do material ocorrem longe do equilíbrio térmico, conforme descrevem Behrisch e Eckstein (2007). Ou seja, o sistema não tem tempo de se equilibrar termicamente entre as colisões, logo o processo de desbastamento para estas energias não é auxiliado pelo processo de evaporação. No caso de partículas com energia na faixa de MeV, uma enorme quantidade de energia é depositada no sólido, aumentando significativamente a temperatura local, levando a evaporação direta de átomos da superfície, auxiliando assim, o processo de desbastamento.

Caso os íons incidentes formem ligações químicas com os átomos do material, têm-se o que é conhecido como desbastamento químico. Ao se formarem moléculas gasosas o processo de desbastamento é aumentado, já na formação de um óxido ou carboneto o processo é diminuído (BEHRISCH; ECKSTEIN, 2007).

Diferentes regimes de colisão no processo de desbastamento físico foram identificados de acordo com a energia das partículas incididas e da seção de choque da colisão, dentre os quais estão:

1. **Regime de impacto único**: Ocorre com a incidência de íons leves ou íons pesados com baixa energia. O átomo-alvo recebe energia suficiente apenas para ser ejetado, sem desencadear outras colisões.
2. **Regime de cascata linear**: O átomo-alvo recebe energia suficiente para

gerar uma cascata de colisões.

3. **Regime de pico**: Ocorre com a incidência de íons pesados, moléculas ou aglomerados de átomos. Devido à alta densidade de átomos de recuo, a maioria dos átomos em um certo volume é posta em movimento.
4. **Regime eletrônico**: Ocorre para energias na faixa de MeV. Parte da energia dos íons pesados incidentes é transferida para os elétrons ao longo do trajeto do íon incidente.

O processo de desbastamento físico tem sido estudado utilizando vários modelos diferentes de simulação computacional, buscando através destas recriar o processo de desbastamento. Dentre os modelos conhecidos encontra-se a teoria da taxa de reação, cálculos de neutrônica de Monte Carlo, métodos de elementos finitos e dinâmica molecular (NORDLUND, 2019).

A dinâmica molecular é uma técnica de simulação computacional para análise dos movimentos de átomos e moléculas através de algoritmos que realizam o cálculo numérico das equações de movimento de Newton para um sistema de partículas clássicas. Segundo Nordlund (2019), no caso de desbastamento físico, a dinâmica molecular pode ser aplicada para a obtenção das trajetórias das partículas ejetadas.

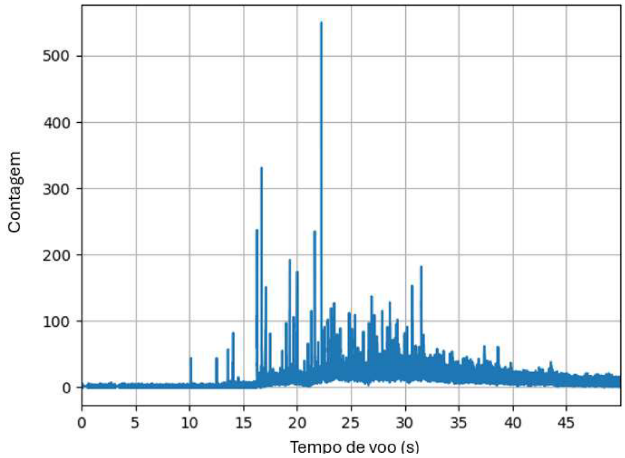
Já no caso do processo de desbastamento ocorrido no regime eletrônico – conhecido como desbastamento eletrônico – Behrisch e Eckstein (2007) dizem que sua ocorrência se dá tanto para feixes com energias na faixa de MeV quanto para íons com energias mais baixas, porém altamente carregados. Este último processo, às vezes, é também denominado como desbastamento potencial.

> Para energias de bombardeio na faixa de MeV, processos adicionais como o desbastamento eletrônico contribuem para a erosão da superfície. Uma parte significativa da energia dos íons pesados incidentes é transferida para elétrons ao longo da trilha do íon. Um acoplamento entre os elétrons e os fônons causa um aquecimento local intenso em um volume cilíndrico. Átomos da superfície podem ser removidos por evaporação em um jato do volume aquecido. Para materiais isolantes, esse processo de erosão é muito maior do que para amteriais metálicos. O desbastamento eletrônico também foi relatado para a incidência de íons lentos e altamente carregados. Nesse caso, a grande energia potencial dos íons incidentes é dissipada no subsistema eletrônico, causando grandes eficiência no processo de desbastamento para materiais semicondutores e isolantes. (Behrish e Eckstein, 2007, p.02, tradução nossa)

2.2 ESPECTROMETRIA DE MASSA DE IONS SECUNDÁRIOS

Amplamente utilizada em diversas áreas do conhecimento, a Espectrometria de Massa (MS, *Mass Spectrometry*) é de grande aplicação nos estudos da área de física atômica e molecular. Segundo Maher, Jjunju e Taylor (2015) é uma técnica analítica que permite identificar, quantificar e explorar – através da análise da relação massa-carga (m/z) – os compostos e moléculas presentes em uma amostra de interesse.

Figura 1 – Exemplo típico de espectro do número de contagens pelo tempo-de-voo registrado num experimento de ToF-SIMS. Este espectro faz parte do experimento **PMMA_12_MeV_Cl**, analisado neste trabalho.



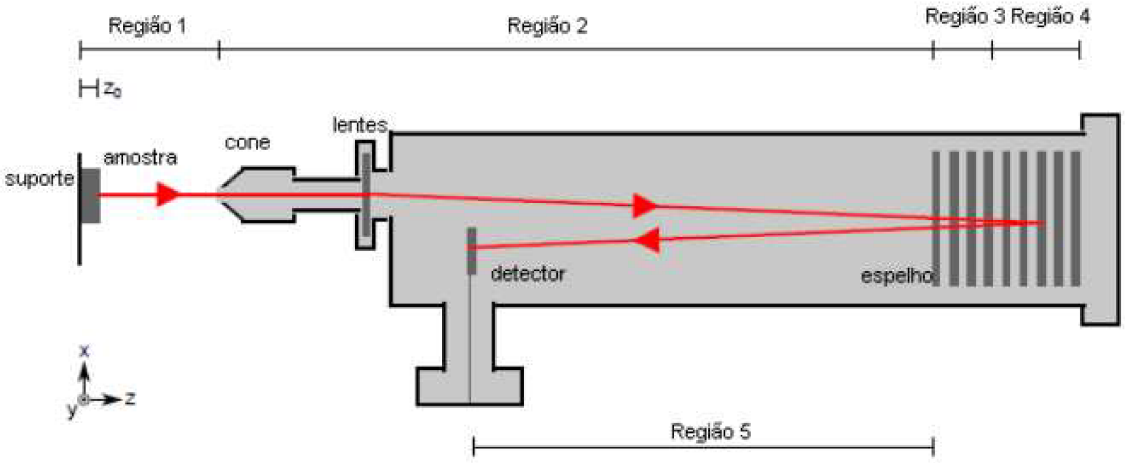
A análise é feita a partir da ionização das amostras, sua separação com base na relação m/z e a detecção dos íons resultantes. O resultado desse processo é um espectro de massa em forma de gráfico de abundância por relação m/z (MAHER; JJUNJU; TAYLOR, 2015).

No caso da Espectrometria de Massa de Íons Secundários (SIMS, *Secondary Ion Mass Spectrometry*), os materiais analisados são ejetados de uma amostra após a incidência de um feixe de íons primário na superfície do alvo (LÓPEZ FERNÁNDEZ, 2012). A discussão da seção anterior sobre o desbastamento causado por um feixe de íons primários indica que o processo de emissão de moléculas é mais eficiente no regime eletrônico. Tal técnica foi originalmente proposta com uso de fragmentos de fissão oriundos de uma fonte de Califórnio (MACFARLANE; TORGERSON, 1976).

O LII conta com uma linha para espectrometria de massa em que a diferenciação para os íons secundários – gerados por meio do processo de desbastamento eletrônico – é realizada através do método do tempo-de-voo. Esta técnica conhecida como Espectrometria de Massa de Íons Secundários por Tempo-de-voo (ToF-SIMS, *Time-of-flight Secondary Ion Mass Spectrometry*), baseia-se na medida de tempo em que um íon leva para atravessar o tubo do espectrômetro até alcançar o detector após a ação de um campo elétrico uniforme, gerando espectros em forma de gráficos do número de contagens pelo tempo-de-voo. Para ilustração, um exemplo de espectro de tempo-de-voo típico é apresentado na Figura 1.

A Figura 2 apresenta um esquema do espectrômetro de massa de íons secundários por tempo-de-voo utilizado para obter os espectros analisados neste estudo. Tal espectrômetro é um típico tubo-de-voo com espelho eletrostático de duplo estágio,

Figura 2 – Esquema de um espectrometro de massa por tempo-de-voo com espelho eletrostático de duplo estágio. As diferentes regiões da representação não estão representadas em escala.



Fonte: Adaptado de ALENCAR *et al.* (2024)

que pode ser dividido em cinco regiões cujos tempo-de-voo são calculados através de cinemática simples.

1. **Região de aceleração**: Nela uma diferença de potencial é aplicada, gerando um campo elétrico que acelera o íon secundário ejetado da amostra. O tempo-de-voo nessa região é dado pela Equação (1).

$$t_1 = \sqrt{\left((v_0)^2 + \frac{2zV_1}{md_1(d_1 - z_0)}\right)}\frac{md_1}{zV_1} = \frac{d_1}{v_1} \tag{1}$$

2. **Região do tubo de voo**: Região livre de campo elétrico que o íon percorre até ser refletido no espelho. O tempo-de-voo nessa região é dado pela Equação (2).

$$t_2 = \frac{d_2}{v_1} \tag{2}$$

3. **Região de retardo**: Região em que o íon é desacelerado pela aplicação de um potencial eletrotático. O tempo-de-voo nessa região é dado pela Equação (3).

$$t_3 = 2d_3\left(\frac{mv_1}{zV_3} - \sqrt{\left(\frac{mv_1}{zV_3}\right)^2 - \frac{2}{zV_3}}\right) = \frac{2d_3}{v_3} \tag{3}$$

4. **Região de deflexão**: Após ser totalmente freado, o íon altera sua direção, inverte o sentido de movimento e acelera em direção ao detector devido a presença do campo elétrico. O tempo-de-voo nessa região é dado pela Equação (4).

$$t_4 = \frac{2md_4v_3}{zV_4} \tag{4}$$

5. **Região do tubo de voo**: Região livre de campo elétrico que o íon percorre até atingir o detector. O tempo-de-voo nessa região é dado pela Equação (5).

$$t_5 = \frac{d_5}{v_1} \tag{5}$$

O tempo-de-voo total será a soma do tempo-de-voo de cada região.

2.3 DEGRADAÇÃO EM POLÍMEROS

O processo de degradação em polímeros ocorre quando sua cadeia polimérica sofre alterações em suas propriedades físicas e químicas – como quebra de cadeia, fragmentação molecular e alteração da composição química – devido a influência de fatores como calor, luz ou força aplicados ao polímero (SPEIGHT, 2011).

Polímeros a base de hidrocarbonetos são mais sensíveis à degradação térmica, enquanto polímeros compostos por cadeias aromáticas são mais vulneráveis à degradação por radiação ultravioleta. Além disso, o processo de degradação com a formação de moléculas menores pode ocorrer por cisão aleatória – quebra aleatória das ligações do polímero – ou por cisão específica – quebra do polímero nas extremidades, formando seus monômeros (SPEIGHT, 2011). Como resultado de sua degradação, os polímeros podem perder sua integridade estrutural. O processo de desbastamento é um dos responsáveis por tal degradação.

Em Thomaz *et al.* (2018) e (2023) o processo de degração em polímeros foi estudado para filmes finos compostos por PMMA e PVC através da técnica Espectroscopia de Fotoelétrons excitados por Raios X (XPS, *X-ray Photoelectron Spectroscopy*). Em Papaléo *et al.* (2006), a degradação em PMMA também foi investigada utilizando-se a técnica Microscopia de Varredura por Força (SFM, *Scanning Force Microscopy*). Os resultados dos experimentos apresentados nesse trabalho serão comparados aos valores de seção de choque obtidos por estes estudos.

2.4 MOTIVAÇÃO E OBJETIVO

Diferente de outras técnicas analíticas, a técnica SIMS permite a caracterização molecular de uma amostra sob estudo. Em combinação com uma varredura espacial

seria possível realizar o imageamento molecular da amostra. Entretanto, feixes de íons são capazes de degradar tecidos orgânicos. Nesse trabalho, construimos um programa para investigar como filmes orgânicos compostos por polímeros se degradam sob exposição contínua de um feixe de íons primários pesados no regime eletrônico não relativístico através da determinação de seção de choque para danos por irradiação. Além disso, o programa também busca identificar possíveis emissões de moléculas a partir da degradação destes polímeros. A fragmentação molecular do alvo é um importante fenômeno para compreender os efeitos da radiação ionizante em materiais orgânicos.

3 MATERIAIS E MÉTODOS

Chamam-se polímeros os materiais orgânicos, inorgânicos, sintéticos ou naturais compostos de monômeros – pequenas unidades estruturais repetidas formadas por moléculas simples contendo uma ligação dupla ou um mínimo de dois grupos funcionais ativos (CHANDA; ROY, 2008).

A existência deste tipo de ligação é responsável por um processo de atração entre uma molécula de monômero sobre a outra repetidamente, o que resulta em ligações primárias e estáveis, originando assim um polímero. E esse processo foi nomeado polimerização (RODRIGUES, 2018).

No contexto deste trabalho, o programa desenvolvido foi empregado para a análise de degradação em três polímeros distintos – Poliestireno (PS), Polimetilmetacrilato (PMMA) e o Cloreto de Polivinila (PVC), formados essencialmente por Carbono (C) e Hidrogênio (H) no caso do PS, Carbono (C), Higrogênio (H) e Oxigênio (O) no caso do PMMA e Carbono (C), Hidrogênio (H) e Cloro (Cl) no caso do PVC.

3.1 POLIESTIRENO

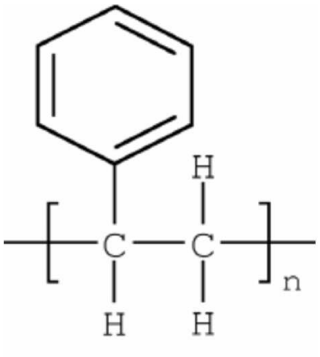
O Poliestireno (C_8H_8) – Figura 3 – é um polímero linear de alto peso molecular, rígido e duro, descoberto em 1839 por Eduard Simon, químico alemão, que o obteve primeiramente como uma substância oleosa, proveniente da destilação de storax, uma resina que se origina a partir de uma árvore nativa da América do Norte chamada LIQUIDAMBAR STYRACIFLUA (GREGOR-SVETEC, 2022).

Possui ponto de fusão de 239°C e temperatura vítrea de 100°C. Devido à sua tendência a constituir uma estrutura amorfa e ausência de reticulação, é um material transparente e facilmente moldável. Além disso, não se dissolve em ácidos, bases ou alcoois, apenas em hidrocarbonetos aromáticos, benzeno e ésteres (RODRIGUES, 2018).

Também possui como características importantes a densidade relativamente baixa (1.05 g/cm^3), boa resistência mecânica, estabilidade térmica, resistência à radiação, capacidade de reciclagem e resistência à absorção de água. Tais características, segundo Hashim e Abbas (2019), o tornam um material extremamente comercializável e amplamente empregado no cotidiano.

De acordo com Montenegro e Serfaty (2002), o PS atualmente é fabricado comercialmente a partir da polimerização do estireno e pode ser encontrado em diferentes utilizações – como na fabricação de embalagens e borrachas, componentes eletrônicos, peças automotivas e artigos domésticos. Além disso, também é empregado na construção civil, sendo usado como isolante térmico, na substituição de argamassa, entre outros.

Figura 3 – Estrutura do polímero Poliestireno.



Fonte: Extraído de FINDLAY (2018)

3.2 POLIMETILMETACRILATO

O Polimetilmetacrilato ($C_5H_8O_2$) – Figura 4 – é um polímero sintético amorfo de peso molecular variado, descoberto na década de 1930 por dois químicos britânicos, Rowland Hill e John Crawford. É produzido a partir do monômero de metacrilato de metila, utilizando diferentes técnicas de polimerização, dentre as quais encontra-se o método de iniciações via radical livre ou aniônica, utilizando técnicas como massa, solução, suspensão e emulsão (ALI; ABD KARIM; BUANG, 2015).

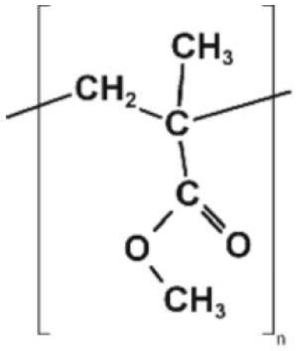
Dentre suas propriedades, destaca-se sua alta rigidez, resistência à tração, arranhões e dureza. Segundo Ali, Abd Karim e Buang (ALI; ABD KARIM; BUANG, 2015) é um material com uma densidade de 1.2 g/cm^3, ponto de fusão em 130°C e temperatura vítrea entre 100°C e 130°C, alta resistência no ultravioleta e estabilidade térmica, suportando temperaturas que variam entre –70°C a 100°C.

Também é um material isolante, porém pode ser copolimerizado quimicamente ou misturado fisicamente a outros polímeros condutores, produzindo assim um novo produto com propriedades sinérgicas entre ambos os monômeros (ABU HASSAN SHAARI *et al.*, 2021).

Apesar de ser amplamente conhecido e utilizado como substituto do vidro, o PMMA também apresenta outras funcionalidades igualmente consideráveis, principalmente na área da medicina – de fato, é um material padrão para objetos simulatores utilizados no controle de qualidade em radiodiagnóstico, medicina nuclear e radioterapia. Devido a sua não toxicidade, baixo custo de produção e compatibilidade com o tecido humano, é aplicado como cimento ósseo para implantes e remodelação de ossos perdidos (HASHIM; ABBAS, 2019).

É também utilizado em lentes, microscópios, fibras ópticas e filmes ópticos devido a sua transparência e resistência no ultravioleta. Além de ser encontrados em produtos do cotidiano como acrílicos para móveis, *displays* de publicidade, luminárias

Figura 4 – Estrutura do polímero Polimetilmetacrilato.



Fonte: Extraído de BARKUSKY *et al.* (2009)

e revestimentos protetores (HASHIM; ABBAS, 2019).

3.3 CLORETO DE POLIVINILA

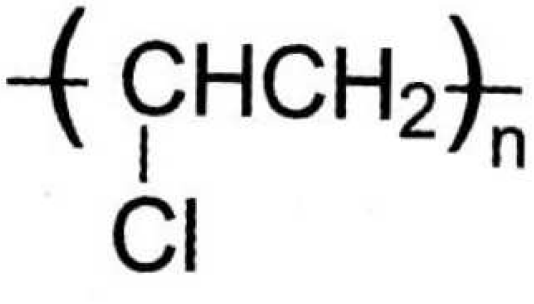
O Cloreto de Polivinila (C_2H_3Cl) – Figura 5 – é obtido a partir da polimerização do cloreto de vinila. Este processo foi feito pela primeira vez por Baumann em 1872, quando notara mudanças induzidas pela luz no monômero de cloreto de vinila de um pó para um sólido branco (EL-GHARBAWY, 2022; CHANDA; ROY, 2008).

Na sua forma pura é quebradiço, não flexível e com pouca estabilidade térmica, o que o tornaria inapto para usos comerciais. Entretanto, de acordo com El-Gharbawy (2022), a partir de processos de tratamentos com aplicação de aditivos – como estabilizadores térmicos, lubrificantes, pigmentos e plastificadores –, misturados a ele através de um processo de composição, as propriedades térmicas, mecânicas e elétricas são aperfeiçoadas, o tornando apto para o uso comercial.

A utilização de aditivos, como estabilizadores térmicos e de luz, é realizada para impedir a degradação do PVC quando este entra em contato com altas temperaturas e luz violeta, visto que suas moléculas não são estáveis a este tipo de interação. Já a aplicação de modificadores de impacto serve para aumentar a resistência dele a impactos, uma vez que as moléculas desses modificadores absorvem a energia do impacto sofrido, protegendo assim o PVC. Pode-se utilizar pigmentos, que também são necessários para se ter resistência térmica e resistência à luz ultravioleta, para alterar a cor do PVC (EL-GHARBAWY, 2022).

Segundo El-Gharbawy (2022) a partir da aplicação destes aditivos separa-se o PVC em formas de utilização, a primeira sendo rígido, uma vez que não recebe o aditivo de plastificantes. O que possibilita o emprego em atividades de construção, como pisos, material para telhados, peças rígidas, placas de sinalização, tubos e perfis. Além de apresentar boa resistência a óleos e gorduras e ser resistente a ácidos.

Figura 5 – Estrutura do polímero Cloreto de polivinila.



Fonte: Extraído de RODRIGUES (2018)

Já o PVC flexível, com adição de plastificantes, apresenta maior resistência e durabilidade, sendo utilizado em fabricação de roupas, calçados, estofados, compostos para cabos elétricos, embalagens, além do *design* de produtos médicos (EL-GHARBAWY, 2022; CHANDA; ROY, 2008).

3.4 AMOSTRAS

As amostras – compostas por filmes finos de Poliestireno, polimetilmetacrilato e Cloreto de Polivinila – foram produzidas utilizando a técnica de *spin coating* sobre substratos de silício. As amostras foram preparadas na Pontifícia Universidade Católica do Rio Grande do Sul (PUC-RS), entre os anos de 2015 e 2018.

3.4.1 Poliestireno

As amostras de PS, fornecidas pela Polymer Standards Service, utilizadas na produção dos filmes finos foram primeiramente dissolvidas em xilano ou tolueno para formar uma solução adequada à utilização da técnica de *spin coating* (PAPALÉO *et al.*, 1999).

Uma solução com 10 g/l foi depositada em um filme de dióxido de silício nativo – aproximadamente 2 nm de espessura – crescido sobre um substrato de Silício (Si).

3.4.2 Polimetilmetacrilato

As amostras de PMMA utilizadas na produção de filmes finos foram primeiramente dissolvidas em anisol para formar uma solução adequada à utilização da técnica de *spin coating* (THOMAZ *et al.*, 2021).

Uma solução com 10 g/l foi depositada em um filme de dióxido de silício nativo – aproximadamente 2 nm de espessura – crescido sobre um substrato de Si. Após

a deposição do PMMA nos substratos, os filmes gerados foram submetidos a um tratamento térmico em uma placa quente a 60°C. Isso foi feito para permitir a relaxação do filme e remover qualquer solvente residual presente.

3.4.3 Cloreto de Polivinila

As amostras de PVC, fornecidas pela Sigma Aldrich, utilizadas na produção dos filmes finos foram primeiramente dissolvidas em ciclohexanona para formar uma solução adequada à utilização da técnica de *spin coating* (TEE *et al.*, 2021).

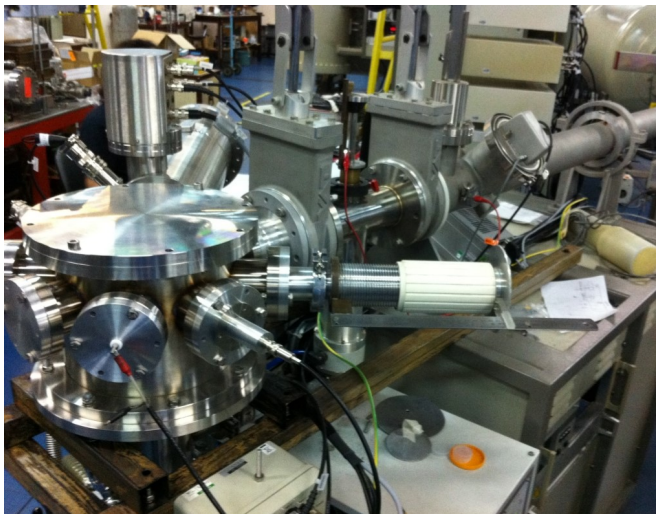
Uma solução com 30 g/l foi depositada em um filme de dióxido de silício nativo – aproximadamente 2 nm de espessura – crescido sobre um substrato de Si.

3.5 EXPERIMENTOS

Os filmes foram irradiados com íons primários positivos de Cloro (Cl) e Cobre (Cu) com energias entre 6,0 MeV e 12,0 MeV na linha para espectrometria de massa de íons secundários do LII-UFRGS.

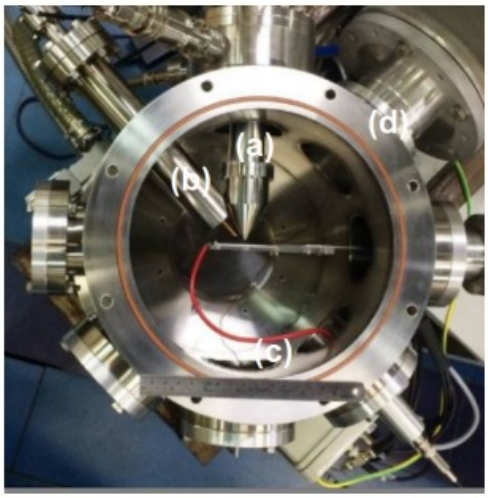
O sistema consiste em uma câmara de vácuo – com capacidade de produção de vácuo da ordem de 10^{-7} mbar – contendo um colimador para o feixe de íons primários, um porta-amostras com possibilidade de movimento horizontal, um tubo-de-voo e um canhão de elétrons (BATTÚ, 2019). Visões externa e interna do sistema podem ser observadas nas Figura 6 e Figura 7, respectivamente.

Figura 6 – Visão externa da linha de espectrometria de massa presente no LII-UFRGS.



Fonte: Extraído da apresentação “Time-of-flight Secondary Ion Mass Spectrometry with MeV primary ions” ALENCAR (2017)

Figura 7 – Visão interna da câmara de irradiação da linha de espectrometria de massa presente no LII-UFRGS. Pode-se encontrar na imagem: (a) analisador por tempo-de-voo, (b) canhão de elétrons, (c) cabo responsável pelo pulso de extração e (d) entrada do feixe primário.



Fonte: Extraído de BATTÚ (2019)

O feixe de íons foi gerado através de um acelerador de partículas, de modelo Tandetron 3MV da HIGH VOLTAGE ENGINEERING EUROPA BV, que para gerar tal aceleração aplica uma alta diferença de potencial entre as extremidades.

> O responsável pela criação do feixe que gera os íons primarios utilizado no *sputtering* é o acelerador de partículas disponível no LII-UFRGS modelo Tandetron 3MV da HIGH VOLTAGE ENGINEERING EUROPA BV. O feixe é obtido após a aceleração de íons a partir da aplicação de um alto potencial entre uma das extremidades e o terminal positivo do acelerador. A partir dai, o feixe obtém características positivas devido a perda de elétrons por causa de um *stripper* com gás de nitrogênio, e então sofre mais uma aceleração direcionada para linha de análise desejada. Essa inversão de cargas do íon permite que um feixe de maior energia seja gerado. (BATTÚ, 2019, p.10).

O colimador na entrada do feixe de íons primários garante que o feixe seja circular com diâmetro aproximado de 1,2 mm e permite monitorar o fluxo de íons na amostra durante os experimentos. O porta-amostras contém uma placa de cobre acoplada, garantindo a conexão elétrica com o gerador responsável pelo pulso de extração. O tubo-de-voo, com uma extensão próxima a um metro, consiste em uma região livre de campos eletromagnéticos, na qual uma extremidade possui um espelho eletrostático de duplo estágio e a outra extremidade apresenta assimetricamente o nariz-de-cone responsável pela coleta dos íons secundários e o detector. A necessidade da utilização de um espelho eletrostático vem das diferentes energias cinéticas que um mesmo íon pode adquirir a partir de sua emissão. Para um período fixo de tempo, íons com maior energia cinética percorrem um maior caminho dentro do tubo-de-voo, enquanto

íons com uma energia cinética menor percorrem um menor caminho. Portanto, para fazer com que os mesmos íons com diferentes energias cinéticas cheguem ao detector simultaneamente, utiliza-se o espelho eletrostático.

> O principal motivo para sua utilização deve-se às diferentes velocidades que um mesmo íon pode receber a partir do pulso de extração dependendo de inúmeros fatores tanto da amostra, quanto das ligações com o material. Dessa forma, é obtida uma distribuição de energia cinética para os íons secundários na direção de sua extração, que é compensada a partir da aplicação de campos elétricos entre as placas do reflectômetro, permitindo que os íons façam a volta e sejam enviados em direção ao detector. Quanto maior a energia cinética da espécie iônica que ali chega, maior será sua penetração ao longo do reflectômetro, ou seja, irá tomar um caminho levemente mais longo em direção a sua detecção. Por outro lado, quando o contrário acontece, o íon irá percorrer uma distância curta nesse ambiente e terá um caminho menor em direção ao detector quando comparado com suas contrapartes mais energéticas. (BATTÚ, 2019, p.13).

O detector, por sua vez, é composto por um empilhamento de duas placas de microcanais, a qual uma voltagem de ganho é aplicada. A passagem de um íon pelas placas pode emitir um pulso de corrente, que é utilizado para contagem. A aquisição de dados é feita através da repetição de ciclos. Um ciclo consiste num período em que há um sinal de *start* e cada pulso de contagem representa um sinal de *stop* do ciclo, ou seja, há múltiplos sinais de *stop*. Os ciclos possuem um tempo mínimo de 100 μs e o detector possui uma alta resolução temporal de 0,5 ns e um tempo-morto entre 4 e 8 ns. O canhão de elétrons permite o descarregamento da amostra, já que o feixe é formado por íons primários positivos. Nos experimentos, as voltagens foram otimizadas para aquisição sequencial de espectros de contagem por tempo-de-voo em passos de 1, 5 ou 10 s.

O sistema de registro do fluxo de íons utiliza um integrador de corrente, uma placa tipo Arduino, responsável por se comunicar com o módulo integrador de carga do experimento, e uma rotina para Laboratory Virtual Instrument Engineering Workbench (LabVIEW) – uma linguagem de programação gráfica voltada para engenharia de sistemas. O fluxo de íons primários pode alcançar a faixa dos 10^{10} íons cm^{-2} s^{-1}, o que permite uma fluência total (fluxo integrado) na ordem de grandeza de 10^{14} íons cm^{-2}.

Os resultados destes experimentos foram centenas de espectros de tempo-de-voo para cada um dos filmes, feixes e passos utilizados, o que evidenciou a necessidade de pensar em estratégias para conseguir fazer a análise de dados em tempo hábil, bem como na intenção de ser compreensível e replicável em outros experimentos correlatos que apresentem variedade e volume de dados igual ou superior.

3.6 PROGRAMAÇÃO

3.6.1 PYTHON

Devido à grande quantidade de espectros resultantes dos experimentos realizados, e após considerar as possibilidades, decidiu-se desenvolver um programa que analisasse de forma automatizada e individual todos os espectros resultantes do mesmo experimento a fim de avaliar a degradação sofrida pelo polímero utilizado em cada experimento. Desta maneira, é possível aplicar de forma consistente e rápida a mesma padronização em termos de parâmetros em todos os dados, aumentando a eficiência e reprodutibilidade dos resultados apresentados nesse trabalho.

Para automatização, optou-se pela linguagem de programação PYTHON, criada em 1991 por Guido van Rossum. Segundo Dhruv, Patel e Doshi (2020), devido sua sintaxe simples e legível, esta linguagem permite programar utilizando poucas linhas de código de forma simplificada, o que facilita a leitura, o desenvolvimento e manutenção do código escrito.

Dentre as vantagens do PYTHON, destaca-se a flexibilidade quanto a erros na construção do código, permitindo que o programa continue a ser executado até que um erro seja encontrado, o que facilita seu desenvolvimento. De acordo com Ranjan *et al.* (2023), essa característica da linguagem também auxilia no tratamento e localização dos erros de forma mais eficiente, o que influencia no ganho de tempo para ser empregado em outros processos.

Além disso, por ser gratuito para uso e distribuição e possuir código aberto, a linguagem PYTHON possui grande, ativa e diversa comunidade de desenvolvedores, facilitando o encontro de recursos, documentação e suporte *online*. Do mesmo modo, por ser suportado em uma ampla variedade de sistemas operacionais, incluindo WINDOWS, MACOS e LINUX, apresenta a característica de migrar programas escritos de um sistema para outro sem a necessidade de alteração no código.

O PYTHON possui uma vasta coleção de bibliotecas – conjuntos de códigos pré-escritos que contém métodos, classes e variáveis – que podem ser utilizadas na realização das mais variadas tarefas.

Além das bibliotecas pré-existentes/padrões – que já vem inclusas em sua instalação padrão – existem diversas outras desenvolvidas por terceiros, o que permite abranger uma gama de funcionalidades especializadas, desde aprendizado de máquina, desenvolvimento *web* e visualização de dados, até o processamento de dados científicos em áreas diversas de conhecimento (SRINATH, 2017).

A linguagem fornece estruturas de dados de alto nível, o que permite a organização, gerenciamento e armazenamento de dados de forma rápida, eficiente e segura. Além de ser uma linguagem interpretativa, interativa e que suporta diversos paradigmas. Dentre eles, a programação orientada a objeto, programação imperativa

ou funcional e estilos processuais (DHRUV; PATEL; DOSHI, 2020).

Considerando todas as características e vantagens do PYTHON em relação à natureza do experimento, optou-se pelo uso do paradigma de programação orientada a objeto para estruturar o desenvolvimento do algoritmo de análise de espectros.

3.6.2 Paradigma de Progração Orientada a Objeto

Em 1966, Alan Kay criou os conceitos do paradigma de programação orientada a objeto, com o intuito de superar as limitações ao tentar transformar ideias matemáticas em códigos de computador de uma maneira de fácil compreensão pelo ser humano. A partir de unidades pré-definidas – como classes, objetos e métodos – consegue-se transcrever a complexidade do mundo real para os códigos de computador de forma simples, estruturada e reutilizável.

Os principais pilares da Programação Orientada a Objeto (POO) são encapsulamento, herança, polimorfismo e abstração. A ideia do encapsulamento é restringir o acesso a certas partes do código, fornecendo apenas o necessário para sua utilização e tornando-o mais seguro quanto ao seu uso pelo usuário. A herança é chave para o processo de reutilização de métodos de uma classe por outra, tornando o processo de escrita de códigos mais rápido, uma vez que não será necessário escrever o mesmo método várias vezes. O polimorfismo está ligado à capacidade do objeto poder ser representado de diferentes formas. Já a abstração é uma forma de simplificar o código para o usuário, permitindo se preocupar em apenas executar um método, sem precisar entender toda a programação por detrás dele.

Assim, optou-se por usar o POO através da construção de classes que encapsulam métodos abstratos (NZERUE-KENNETH *et al.*, 2023), buscando utilizar o algoritmo criado para automatizar a análise de degradação dos polímeros – baseada na investigação de espectros de tempo-de-voo obtidos sequencialmente durante a irradiação – de forma mais rápida, fácil e simples de ser programada e executada.

4 RESULTADOS E ANÁLISE

Valendo-se dos princípios da POO, estruturou-se o algoritmo por detrás do programa a partir de uma hierarquia de diretórios, conforme observa-se na Figura 8. Cada pasta contém um arquivo PYTHON composto por classes, responsáveis por encapsular o código. Cada classe possui métodos próprios, cujas funcionalidades são semelhantes e que podem ser chamadas nas mais variadas etapas do código, evitando a redundância de ter que escrevê-los novamente. Nos Quadro 2 e Quadro 3 encontram-se informações referentes a cada diretório e uma breve descrição sobre a função principal da classe presente em seu arquivo.

Figura 8 – Hierarquia de diretórios do algoritmo de análise automatizada dos espectros.

Quadro 2 – Descrição dos diretórios.

Diretório	**Arquivo**	**Classe**	**Número Métodos**	**Apêndice**	**Descrição**
calculation	CALC.PY	CALC	6	APÊNDICE A	Abriga todos os métodos referentes a cálculos e operações a serem realizadas
files	FILES.PY	FILES	7	APÊNDICE B	Gerencia todas interações com arquivos
errors	ERRORS.PY	ERRORS	3	APÊNDICE C	Calcula erros de todas as operações
utils	UTILS.PY	UTILS	3	APÊNDICE D	Armazena métodos gerais utilizados pelo algoritmo
graphics	GRAPHICS.PY	GRAPHICS	2	APÊNDICE E	Gera e salva os gráficos gerados pelo algoritmo
function	FUNCTION.PY	FUNCTION	2	APÊNDICE F	Guarda as funções utilizadas no ajuste do pico
flow	FLOW.PY	—	1	APÊNDICE L	Organiza o fluxo de execução do algoritmo
script	RUN.PY	FUNCTION	1	APÊNDICE M	Torna o algoritmo executável em terminal

4.1 PICOS DE EMISSÃO

O programa para análise de degradação de polímeros durante a irradiação com íons primários de Cobre e Cloro é iniciado pelo estudo dos picos de emissão encontrados nos espectros de contagem por tempo-de-voo resultantes do experimento. Para isto, escreveu-se um *script* em PYTHON que pudesse automatizar todo este

Quadro 3 – Descrição do diretório *implement*.

Arquivo	Classe	Número Métodos	Apêndice	Descrição
IMPLEMENT.PY	DATAPROCESSOR	1	APÊNDICE G	Encontra picos
IMPLEMENT_2.PY	DATAPROCESSOR2	1	APÊNDICE H	Compara picos
IMPLEMENT_3.PY	DATAPROCESSOR3	1	APÊNDICE I	Ajusta picos
IMPLEMENT_4.PY	DATAPROCESSOR4	1	APÊNDICE J	Compara massas
IMPLEMENT_5.PY	DATAPROCESSOR5	1	APÊNDICE K	Determina seção de choque
IMPLEMENT_6.PY	DATAPROCESSOR6	1	APÊNDICE L	Determina possíveis moléculas

Quadro 4 – Diretórios criados para armazenar arquivos gerados pelo *script* de análise automatizada de espectros.

Diretório	Descrição
plot	Arquivos de imagem contendo os espectros com seus respectivos picos em destaque
peaks	Arquivos de texto contendo parâmetros referentes aos picos encontrados
adjusted_functions	Arquivos de imagem contento os dados de picos ajustados para um conjunto de funções

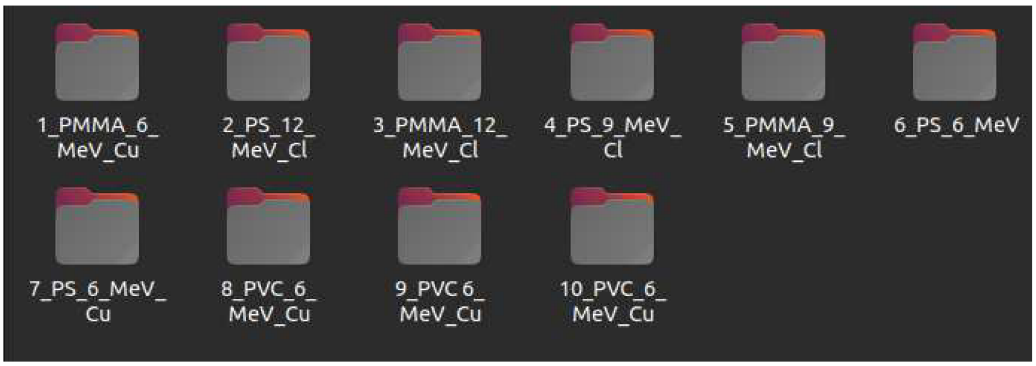
processamento.

O *script* inicia com a tentativa de encontrar todos os picos de todos os espectros gerados a cada rodada de experimento. Devido ao grande número de espectros gerados decidiu-se por implementar um método que analisasse todos os espectros salvos na pasta do experimento de uma única vez, evitando assim ter que executar manualmente um *script* para cada espectro individualmente.

Cria-se, caso não exista, três diretórios distintos que armazenarão os arquivos gerados em cada etapa do *script* de acordo com a sua finalidade. Estes diretórios são criados dentro da pasta onde se encontram os arquivos de espectros que serão analisados. Através do método CREATE_PATHS – presente na classe FILES – estes diretórios são criados. No Quadro 4 é possível identificar o nome e a finalidade de cada diretório criado.

Em seguida, buscando automatizar o processo de análise de espectros, executa-se o método PROCESS_MULTIPLE_FILE – encontrado da classe DATAPROCESSOR – responsável por gerenciar todo o fluxo referente a esta etapa. São necessários apenas

Figura 9 – Pastas nomeadas de acordo com o tipo de polímero e informações do feixe de íons primários, contendo os espectros a serem analisados.



quatro parâmetros, a saber: (i) caminho até a pasta onde se encontram os espectros a serem analisados; (ii) caminho até a pasta onde serão salvas as imagens geradas; (iii) caminho até a pasta onde serão salvos arquivos de texto gerados; e (iv) uma *string* como FLAG indicando se os arquivos gerados pelo método devem ser salvos ou não.

No que se refere aos arquivos de espectros disponibilizados, estes encontram-se em diferentes pastas, nomeadas de acordo com o tipo de polímero existente na amostra e pela espécie e energia do feixe de íons primários. Na Figura 9 pode-se notar esta subdivisão em pastas.

Dentro destas pastas encontram-se os espectros adquiridos sequencialmente durante a exposição ao feixe contínuo de íons primários como resultados de cada experimento. Eles possuem uma padronização quanto ao seu nome, facilitando o processo de leitura por parte do *script*, sendo assim nomeados:

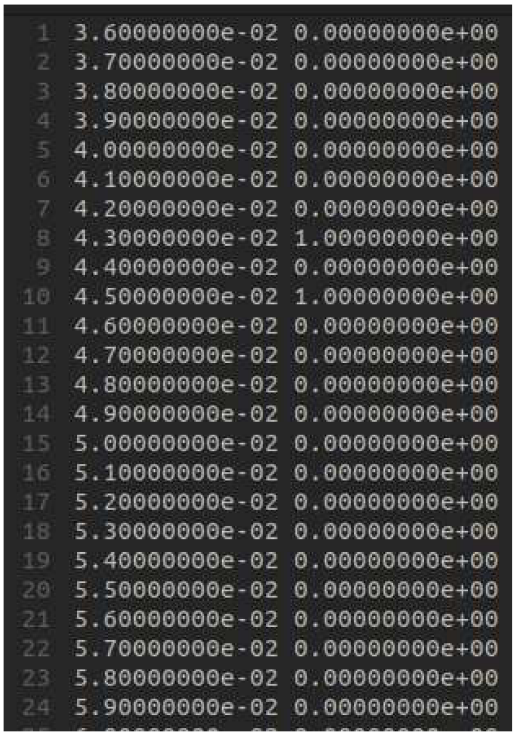
SPEC<XXX>-<POLÍMERO>-<TTTT>S-DEFLECT41.TXT

Onde os valores XXX são sustituidos pelo número do espectro, POLÍMERO pelo tipo de polímero irradiado e TTTT pelo tempo, em segundos, da duração da aquisição de dados por parte do experimento.

Portanto, o primeiro passo do *script* para encontrar os picos de emissão é listar – em ordem numérica – todos os espectros presentes na pasta fornecida como parâmetro. Para isto, chama-se outro método – denominado LIST_TXT_FILES – que encontra-se presente na classe FILES. Recebendo como argumento o caminho completo até uma pasta de espectros, este método é responsável por avaliar quais arquivos existentes nesta pasta terminam com a extensão .TXT e por retornar uma lista – ordenada numericamente – com o nome de cada arquivo encontrado.

A seguir, todas as próximas operações são realizadas para cada item da lista individualmente, ou seja, para cada espectro. Iniciando-se pela construção do caminho completo até o espectro em questão, para isto, chama-se o método CONSTRUCT_-

Figura 10 – Exemplo do conteúdo de um arquivo de espectro de contagens por tempo-de-voo.



```
1  3.60000000e-02 0.00000000e+00
2  3.70000000e-02 0.00000000e+00
3  3.80000000e-02 0.00000000e+00
4  3.90000000e-02 0.00000000e+00
5  4.00000000e-02 0.00000000e+00
6  4.10000000e-02 0.00000000e+00
7  4.20000000e-02 0.00000000e+00
8  4.30000000e-02 1.00000000e+00
9  4.40000000e-02 0.00000000e+00
10 4.50000000e-02 1.00000000e+00
11 4.60000000e-02 0.00000000e+00
12 4.70000000e-02 0.00000000e+00
13 4.80000000e-02 0.00000000e+00
14 4.90000000e-02 0.00000000e+00
15 5.00000000e-02 0.00000000e+00
16 5.10000000e-02 0.00000000e+00
17 5.20000000e-02 0.00000000e+00
18 5.30000000e-02 0.00000000e+00
19 5.40000000e-02 0.00000000e+00
20 5.50000000e-02 0.00000000e+00
21 5.60000000e-02 0.00000000e+00
22 5.70000000e-02 0.00000000e+00
23 5.80000000e-02 0.00000000e+00
24 5.90000000e-02 0.00000000e+00
```

PATH_FILES – também disponível na classe FILES. Por meio do caminho até a pasta onde se encontra salvo este espectro e do nome deste espectro, proveniente do resultado do método anterior – passados como argumento –, constrói-se o caminho completo até este espectro em específico. Utiliza-se este caminho como parâmetro para os próximos dois métodos, EXTRACT_SPEC_NUMBER e EXTRACT_SPEC_NAME – ambos encontrados na classe UTILS. Com eles, extrai-se, respectivamente, o número do espectro e seu nome, salvando estas informações em duas variáveis.

Também usa-se o caminho completo do espectro para trazer suas informações para dentro do *script*. Conforme observa-se na Figura 10, estes arquivos são formados por duas colunas e centenas de linhas, sendo a primeira coluna referente ao tempo de voo em microssegundos e a segunda ao seu respectivo número de contagem. Ambas informações são lidas em forma de um ARRAY NUMPY – estrutura de dados central da biblioteca NUMPY do PYTHON – responsável por representar matrizes e vetores. Esta representação foi escolhida devido a sua facilidade no tratamento de dados. Assim, armazena-se o tempo de voo em uma variável chamada DATA_TOF e o número de contagem em outra variável chamada DATA_COUNT. A leitura do arquivo em variáveis é realizada através do método READ_FILES – disponível pela classe FILES –, recebendo como argumentos o caminho até o arquivo e o valor numérico da coluna desejada.

Ambas informações de tempo-de-voo e número de contagem serão os *inputs*

do próximo método chamado FIND_DATA_MULTIPLE_PEAKS – encontrado na classe CALC – juntamente com o número e o nome do espectro sendo analisado, bem como os caminhos onde serão salvos os arquivos de texto e de imagens gerados como retorno da aplicação deste método. Além destes parâmetros ainda é passada uma FLAG, indicando se os arquivos retornados pelo método devem ser salvos. O método FIND_DATA_MULTIPLE_PEAKS também é responsável por realizar a detecção dos picos presentes em um espectro de tempo-de-voo. Para isso, ele utiliza a função FIND_PEAKS – encontrada na biblioteca SCIPY do PYTHON. Como observado pelos contribuidores (SCIPY CONTRIBUTORS, s.d.), é necessário passar alguns argumentos para que seja possível encontrar picos em um determinado sinal.

Após a realização de testes com conjuntos de valores diferentes como parâmetros, chegou-se a valores que resultaram em um número significativo de picos em comparação com o ruído. No Quadro 5 encontram-se os parâmetros que foram utilizados para encontrar os picos com uma breve descrição. Já na Tabela 1 estão os valores utilizados para cada parâmetro em cada experimento. O argumento *count* é dado pelo tempo-de-voo em cada espectro.

Quadro 5 – Argumentos passados ao método FIND_PEAKS, presente na biblioteca SCIPY do PYTHON, dedicada a encontrar picos em um determinado sinal.

Argumento	**Descrição**
Count	Série de dados onde os picos serão procurados
Height	Altura mínima do pico para ser detectado
Prominence	Distância vertical entre um pico e sua linha de contorno mais baixa
Width	Largura mínima de um pico
Distance	Minima distância horizontal entre picos

Tabela 1 – Valores passados como argumentos ao método FIND_PEAKS, presente na biblioteca SCIPY do PYTHON, dedicada a encontrar picos nos experimentos de degradação de polímeros.

Experimento	***Height***	***Prominence***	***Width***	***Distance***
PMMA_6MeV_Cu	h_1	10	0.01	400
PS_12MeV_Cl	h_1	10	0.01	400
PMMA_12MeV_Cl	h_1	10	0.01	400
PS_9MeV_Cl	h_1	10	0.01	400
PMMA_9MeV_Cl	h_1	10	0.01	400
PS_6MeV_Cl	h_1	10	0.01	400
PS_6MeV_Cu	h_1	8	0.01	400
1_PVC_6MeV_Cu	h_1	2.5	0.01	400
2_PVC_6MeV_Cu	h_1	10	0.01	400
3_PVC_6MeV_Cu	h_1	10	0.01	400

A variável h_1 é determinada dinamicamente para cada espectro, a partir da Equação (6). Ela é calculada como sendo 0.025 vezes o valor do maior pico encontrado neste espectro.

$$h1 = 0.025 \times \text{Máxima Contagem do Espectro} \tag{6}$$

O método FIND_PEAKS retorna dois resultados, o primeiro sendo o valor do índice da posição em que o pico se encontra e o segundo um dicionário de elementos, estando dentre estes elementos a altura de cada pico encontrado. Em posse do índice da posição do pico é possível encontrar o valor do tempo-de-voo que representa esta posição, e com isso é possível agrupar o valor do tempo-de-voo com seu respectivo pico de emissão.

Cria-se então, um arquivo de texto no formato .TXT onde será armazenado o tempo-de-voo de cada pico encontrado e seu respectivo valor de contagem. Caso a FLAG responsável por salvar os arquivos seja passada, este arquivo de texto é salvo na pasta passada como parâmetro.

Após salvar o arquivo, outro método é chamado, PLOT_PEAKS – presente na classe GRAPHICS. Este método recebe como *inputs* os dados de tempo-de-voo e contagem do espectro, a lista de tempo-de-voo e contagem dos picos, a altura mínima que um pico tem de ter, o caminho onde as imagens geradas serão salvas, e o número e nome do espectro analisado. Com estas informações o método plota a imagem do espectro destacando todos os picos encontrados anteriormente, salvado-a no outro caminho passado também como parametro. As Figura 11, Figura 12, Figura 13, Figura 14, Figura 15, Figura 16, Figura 17, Figura 18, Figura 19 e Figura 20 representam um espectro de cada experimento e sua respectiva imagem retornada pelo método.

Figura 11 – Comparação entre o espectro número 257 do experimento **PMMA_6MeV_-Cu** sem o destaque de picos e com o destaque de picos.

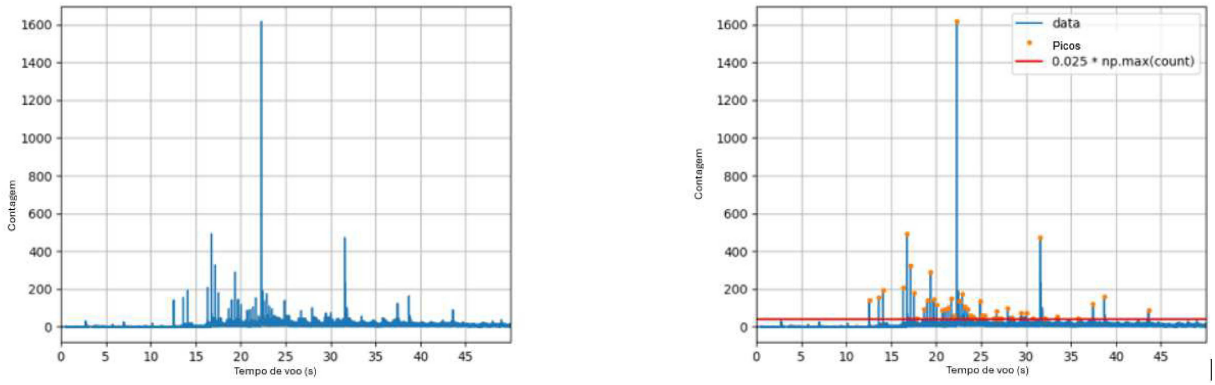


Figura 12 – Comparação entre o espectro número 513 do experimento **PS_12MeV_CI** sem o destaque de picos e com o destaque de picos.

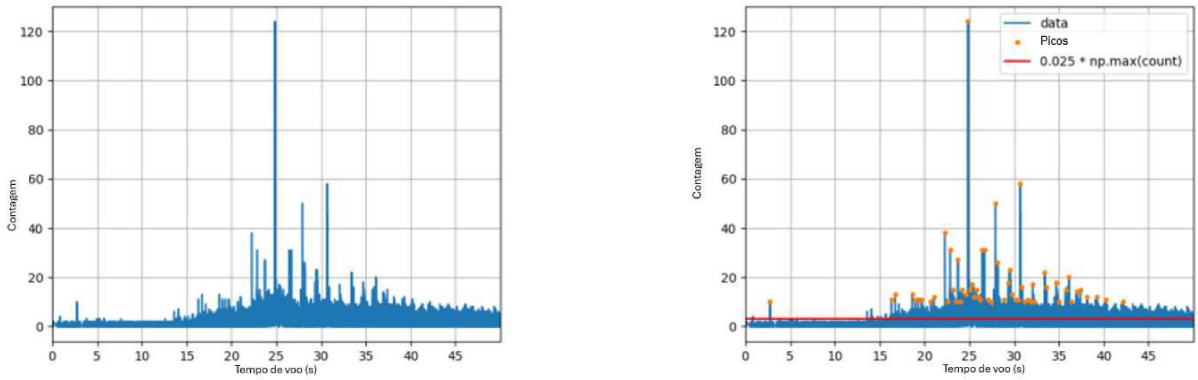


Figura 13 – Comparação entre o espectro número 959 do experimento **PMMA_-12MeV_CI** sem o destaque de picos e com o destaque de picos.

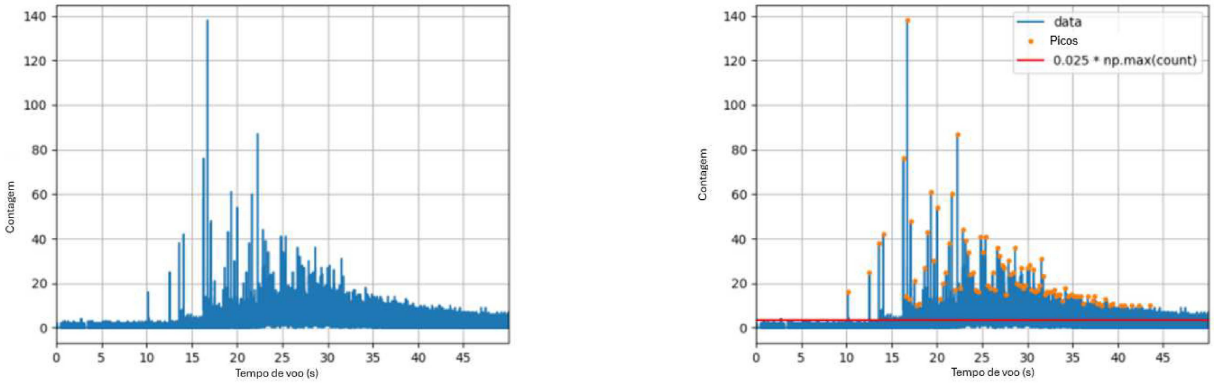


Figura 14 – Comparação entre o espectro número 1186 do experimento **PS_9MeV_CI** sem o destaque de picos e com o destaque de picos.

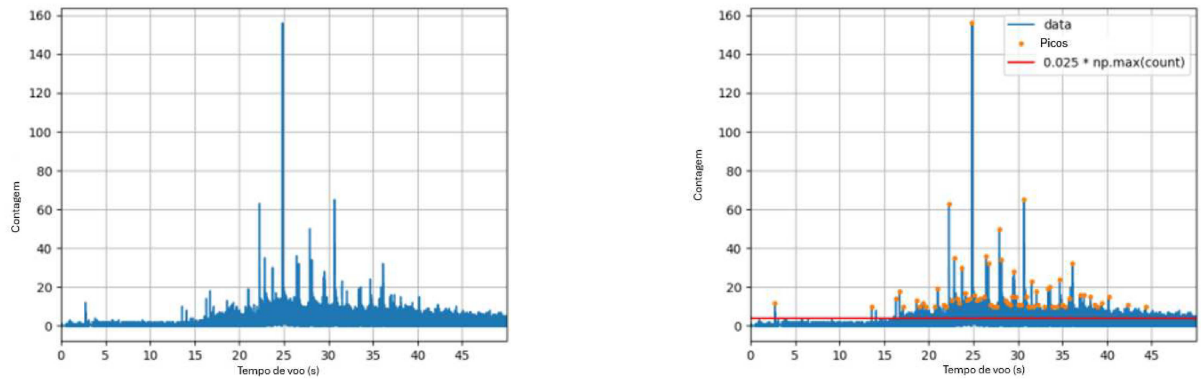


Figura 15 – Comparação entre o espectro número 1644 do experimento **PMMA_-9MeV_Cl** sem o destaque de picos e com o destaque de picos.

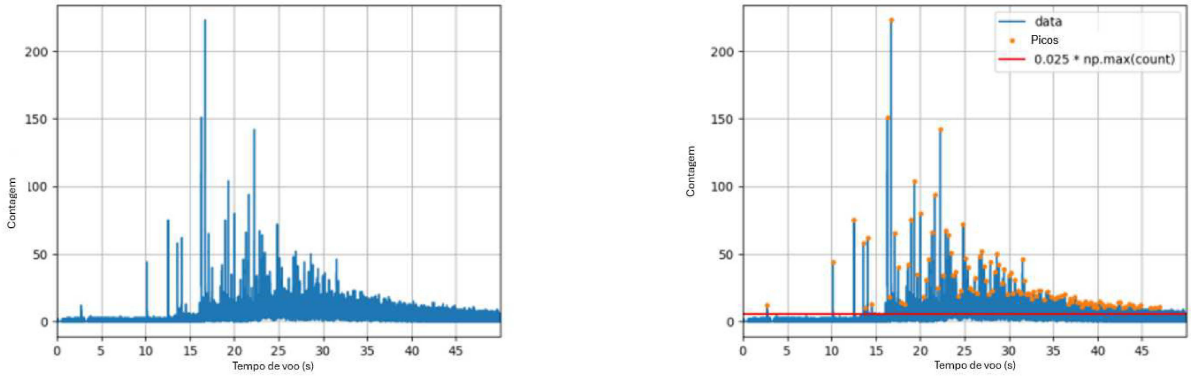


Figura 16 – Comparação entre o espectro número 2042 do experimento **PS_6MeV_Cl** sem o destaque de picos e com o destaque de picos.

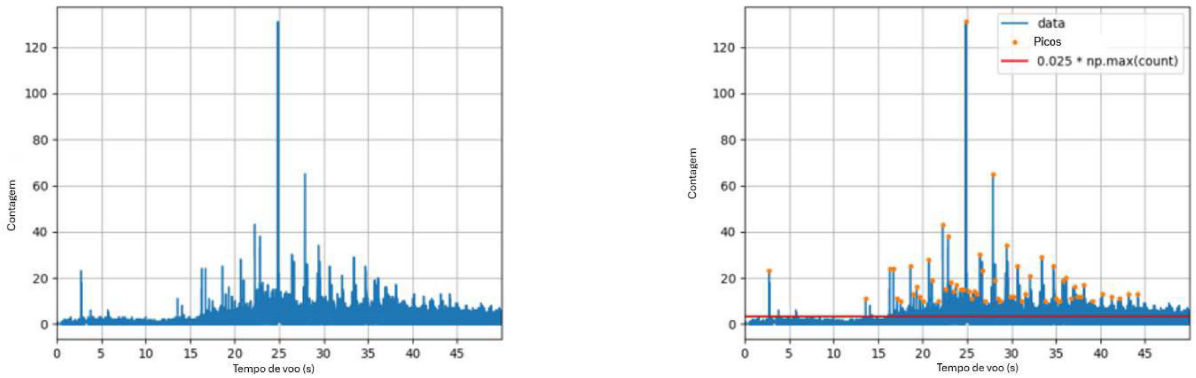


Figura 17 – Comparação entre o espectro número 10 do experimento **PS_6MeV_Cu** sem o destaque de picos e com o destaque de picos.

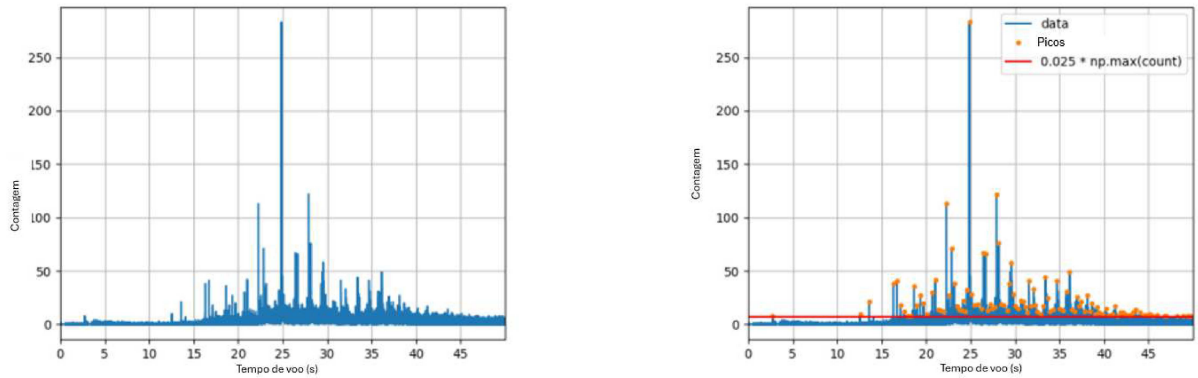


Figura 18 – Comparação entre o espectro número 630 do experimento **PVC_6MeV_Cu** sem o destaque de picos e com o destaque de picos.

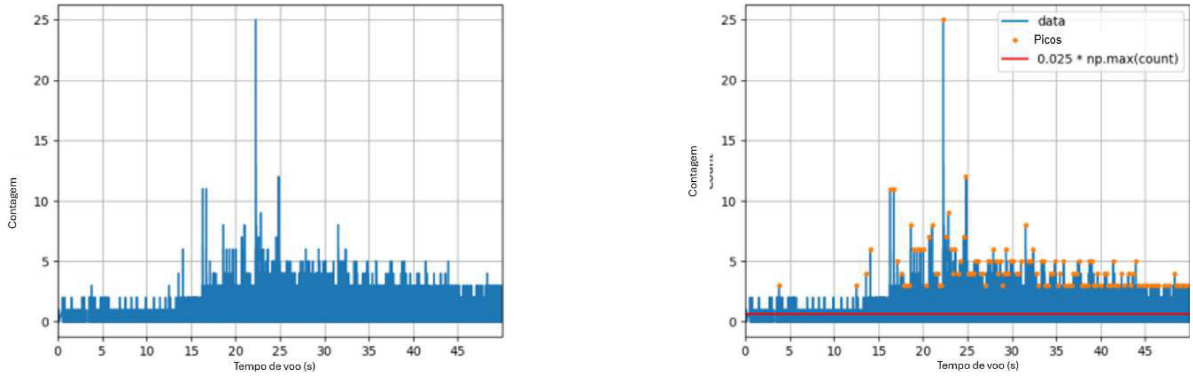


Figura 19 – Comparação entre o espectro número 2420 do experimento **PVC_6MeV_-Cu** sem o destaque de picos e com o destaque de picos.

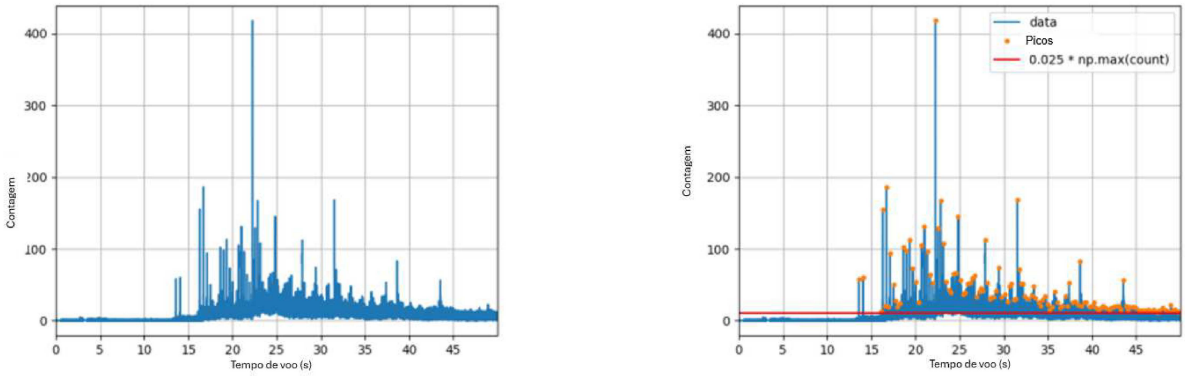
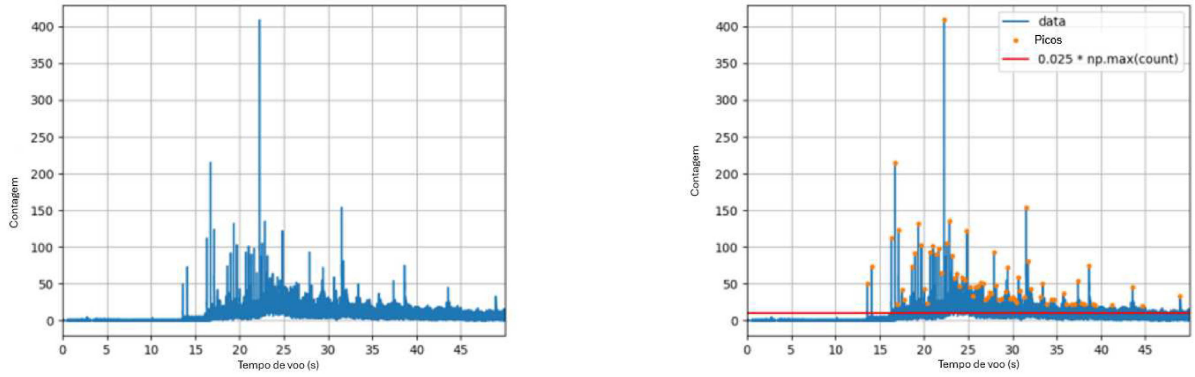


Figura 20 – Comparação entre o espectro número 3007 do experimento **PVC_6MeV_-Cu** sem o destaque de picos e com o destaque de picos.



4.1.1 Máximos e Áreas

Seguindo com o fluxo do *script*, ainda dentro do método FIND_MULTIPLES_PEAKS, encontra-se o real valor máximo e a área de cada pico. Logo após a finalização do procedimento anterior para identificar todos os picos de emissão num espectro, entra-se em uma nova etapa do método, responsável por encontrar estes valores para cada pico. Para isto, lê-se em duas outras variáveis os valores de tempo-de-voo e contagem dos picos provenientes do arquivo de texto gerado anteriormente. É importante notar que alguns espectros não geraram picos, devido a estrutura de seu sinal, conforme mostrado pela Figura 21. Para estes espectros as análises de contagem máxima e área realizadas a seguir não serão executadas.

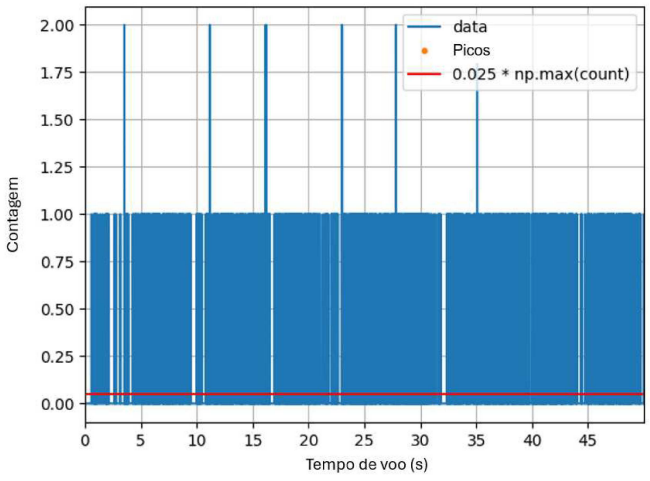
Usando-se o método READ_FILES_2 – encontrado na classe FILES –, o processo de leitura do tempo-de-voo dos arquivos de texto contendo os picos de cada espectro é realizado na variável chamada PEAK_TOF. Caso o espectro não possua picos, este método retornará o valor *None*.

O *script* checa se a variável PEAK_TOF é nula, caso seja, o espectro em que essa variável se refere não será analisado e o *script* passará para o próximo passo. Caso contrário, o processo de encontro de contagem máxima e área é continuado.

Ao continuar o fluxo acima descrito, chama-se o método READ_FILES novamente e este lê os valores de contagem presentes no arquivo de texto dos picos encontrados em uma variável chamada PEAK_COUNT. Em posse dessas duas variáveis e das duas anteriores – DATA_TOF e DATA_COUNT – iniciam-se os cálculos necessários para encontrar a área e a contagem máxima real de cada pico.

Primeiramente encontra-se o valor máximo real de cada pico. E para isso, faz-se

Figura 21 – Exemplo de um espectro do experimento **PVC_6MeV_Cu** em que não é possivel encontrar picos de emissão.



uma análise estatística encontrando o valor da média e do desvio padrão do fundo para cada pico, aplicando-se o método ANALYZE_STATISTICS_BKGD – disponibilizado na classe CALC. Este método recebe como parâmetros os ARRAYS de tempo-de-voo de todo o espectro, de contagem de todo o espectro e de tempo-de-voo dos picos.

Para realizar esta análise, delimita-se uma região anterior a cada pico e filtra-se o valor de contagem de todo o espectro para apenas valores que estão dentro desta região, salvando estes valores em uma lista de listas. Para cada item dessa lista utiliza-se seus elementos para calcular a média e o desvio padrão para aquele pico em questão. O retorno deste método é outra lista com todas as médias e desvio padrão para todos os picos encontrados. O próximo passo é subtrair os valores de contagem dentro do pico pelo valor da média de contagens antes do pico específico (fundo). Para isso, aplica-se o método COUNT_INSIDE_REGION_OF_INTEREST_MINUS_MEAN_BKGD – presente na classe CALC – passando como argumentos: (i) o tempo-de-voo de todo o espectro; (ii) a contagem de todo o espectro; (iii) o tempo-de-voo dos picos; (iv) os limites inferior e superior para calcular a região de interesse dentro do pico; e (v) a média de fundo.

Uma interação em cada pico é realizada, encontrando-se a região de interesse para o pico em questão. Subtrai-se todos os valores de contagem dentro desta região de interesse pelo valor da média do fundo para aquele pico em específico, retornando-se duas listas, uma contendo o tempo-de-voo para cada ponto dentro da região de interesse e outra com o valor da contagem após subtração.

Utilizando as variáveis retornadas pelo método anterior, encontra-se o novo valor máximo de contagem para cada pico, através do método NEW_MAX_TOF_COUNT_-INSIDE_PEAK_REGION – encontrado na classe CALC – e cria-se um novo arquivo de texto no formato .TXT, que assim como o anterior, possui duas colunas, sendo a primeira o tempo-de-voo de todos os picos e a segunda seu respectivo novo valor de contagem.

Para calcular a área de cada, utiliza-se o método CALC_AREA_FOR_PEAKS – disponível na classe CALC –, passando os tempos-de-voo dentro da região que representa o pico e a variável com os novos valores para a contagem dos picos dentro desta região como parâmetros.

Finalizando esta etapa, calcula-se o erro para a contagem, área e tempo-de-voo a partir do métodos ERROR_AREA, ERROR_COUNT e ERROR_TOF, respectivamente – todos presentes na classe ERRORS. Para o cálculo de erro da área, é usada a Equação (7), já para o erro de contagem é utilizada a Equação (8). ambos supondo uma distribuição de Poisson para o sinal.

$$\text{erro}_{\text{area}} = \sqrt{\text{area}} \tag{7}$$

$$\text{erro}_{\text{contagem}} = \sqrt{\text{contagem}} \tag{8}$$

Já para o cálculo do erro do tempo-de-voo, cria-se uma lista de funções interpoladas para cada região de interesse, Equação (9).

$$f_i(x) = \left\{ \text{interp1d}(x, y, \text{bounds\_error=False}, \text{fill\_value} = 0) \quad \text{para } i = 1, 2, \ldots, n \right. \tag{9}$$

na qual f_i representa a função interpolada para a i-ésima região de interesse, x e y os valores de tempo-de-voo e contagem para esta região.

Em seguida, gera-se novos valores de x em torno de cada valor máximo de tempo-de-voo por pico, de acordo com a Equação (10).

$$x_{\text{new}_i} = [x_{\text{min}_i}, x_{\text{min}_i} + \Delta x, x_{\text{min}_i} + 2\Delta x, \ldots, x_{\text{max}_i}] \tag{10}$$

onde x_{min} e x_{max} representam os novos valores mínimos e máximos de x para a i-ésima região, respectivamente, e Δx o incremento.

Calcula-se os valores interpolados de y para cada conjunto de novos valores de x através da Equação (11).

$$y_{\text{interpolado}_i} = f_i(x_{\text{new}_i}) \tag{11}$$

Encontra-se os valores de x para metade dos valores máximos de y, Equação (12).

$$x_{\text{half\_max}_i} = \begin{cases} \text{None} & \text{se } |y_{\text{interpolado}_i}| = 0 \\ \text{ARGMAX}(|y_{\text{interpolado}_i}|) & \text{caso contrário} \end{cases} \tag{12}$$

onde a função ARGMAX retorna o índice do maior elemento em uma lista.

Finalmente, calcula-se a diferença – sendo esta o erro estimado para o tempo-de-voo – entre os valores máximos de tempo-de-voo e os valores x correspondentes aos valores da metade do valor máximo de y, Equação (13).

$$\text{difference}_i = \begin{cases} \text{max\_tof}_i - x_{\text{half\_max}_i} & \text{se } x_{\text{half\_max}_i} \neq \text{None} \\ \text{None} & \text{caso contrário} \end{cases} \tag{13}$$

Assim, com os valores de tempo-de-voo, nova contagem máxima, área, erro do tempo-de-voo, erro da contagem nova e erro da área para cada pico, cria-se um arquivo de texto no formato .TXT com estas informações. Estes arquivos serão usados na etapa subsequente do algoritmo. Na Figura 22 está um exemplo dos arquivos de texto gerados para um dos espectros analisados.

Figura 22 – Exemplo de arquivo de texto gerado com valores de tempo-de-voo, nova contagem máxima, área, erro do tempo-de-voo, erro nova contagem máxima e erro área para cada pico do espectro 1169 do experimento **PS_-9MeV_Cl**.

Max_ToF	Max_Count	Area	Error_Area	Error_Count	Error_ToF
16.31	23.57	194.02999999999975	13.929465172791083	4.854894437575342	0.000370000000366133
16.7235	25.494949494949495	279.10101010100976	16.706316473149002	5.049252369900865	0.00172000000315025
17.125	19.8989898989899	212.0202020202019	14.560913502256716	4.460828387081249	0.0004500000003631044
22.2665	76.83838383838383	1509.232323232323	38.84883940650381	8.765750614658383	0.001980000000305182
22.8655	31.767676767676768	603.6464646464655	24.56921782732339	5.636282176016099	0.0041900000002215165
23.734	27.363636363636363	515.7272727272731	22.709629515412026	5.231026320296655	0.0010400000003407683
24.844	159.33333333333334	3247.333333333336	56.98537824155716	12.622730819174325	0.0014300000003260038
26.423	29.259999999999998	717.5399999999996	26.786937114944656	5.409251334519408	0.0031700000002601314
26.6775	32.33	542.0699999999998	23.282396783836493	5.6859475903318	0.0015100000003229752
27.9105	70.94	1347.2600000000016	36.705040525791574	8.422588675698226	0.0013100000003305468
28.153	25.83	469.5699999999996	21.669563908856116	5.0823223038292245	0.0016800000003165394
29.556	23.8989898989899	421.0202020202019	20.518776815887488	4.888659315087307	0.0013700000003282753
30.6785	75.16161616161617	1978.767676767677	44.483341564766434	8.669579929939868	0.0026100000002813317
31.537	24.22	537.3800000000008	23.181458107720506	4.921381919745713	0.0009800000003430398
33.4045	19.282828282828284	406.6262626262629	20.164976137507896	4.391221730091557	0.0021699999975126616
33.606	15.690000000000001	35.199999999999974	5.932958789676528	3.961060464067672	0.0002499999969032274
34.6925	20.31	427.80000000000007	20.683326618317473	4.5066617356975	0.0030599999977951597
36.119	21.03030303030303	471.3939393939388	21.71160840181903	4.5858808347255415	0.0020899999974872685

4.2 AJUSTE DE FUNÇÃO

A próxima etapa do *script* de análise de degradação de polímeros após a irradiação com íons primários de Cobre e Cloro é descobrir quais picos encontrados na etapa anterior são de emissão e quais são apenas ruídos. Esta etapa é crucial para descobrir quais moléculas foram emitidos pela amostra após a irradiação do feixe de íons – cada molécula possuirá um tempo-de-voo característico – que será utilizado para associá-lo aos picos encontrados. Com uma sequência de informações sobre uma molécula ao longo da exposição ao feixe de íons, é possível determinar sua seção de choque para danos por irradiação.

Para realizar a separação entre pico e ruído, o *script* é separado em duas partes. A primeira parte é responsável por mapear os picos encontrados em todos os arquivos de espectros e verificar picos semelhantes entre os espectros. A segunda parte é responsável pela plotagem destes picos semelhantes em um gráfico e por tentar ajustar um conjunto de funções a estes dados.

4.2.1 Comparação Entre Picos

A primeira parte é realizada através do método COMPARING_PEAKS – disponibilizada pela classe DATAPROCESSOR2. Este método recebe como parâmetro apenas o caminho do diretório onde estão salvos os arquivos de texto gerados pela etapa anterior. Ele, então, lista todos os arquivos presentes neste diretório e os filtra, criando uma lista apenas com os arquivos que possuam a paralvra RESULT em seu nome.

Esta lista é ordenada numericamente utilizando o método CUSTOM_SORT – encontrado na classe UTILS. Então, seleciona-se o décimo arquivo desta lista para ser o arquivo de referência – arquivo com os picos que serão utilizados para mapear

os demais.

Em seguida, lê-se o arquivo de texto em um DATAFRAME – uma estrutura de dados rotulada bidimensional com colunas de tipos potencialmente diferentes, semelhantes a uma tabela ou a um banco de dados (MCKINNEY, s.d.) – onde cada coluna do DATAFRAME representa uma coluna do arquivo de texto de referência.

Um dicionário – estrutura de dados que mapeia chaves a valores (PYTHON SOFTWARE FOUNDATION, s.d.) – vazio é inicializado para armazenar informações sobre os picos já mapeados. E assim começa então o processo de mapeamento de picos entre os arquivos listados. Para isto, itera-se sobre cada elemento da lista de arquivos transformando-os também em um DATAFRAME. Para cada pico no arquivo de referência, procura-se por picos correspondentes dentro do arquivo sendo atualmente iterado. Esta procura é feita comparando os tempos-de-voo entre os picos.

A correspondência entre picos não precisa ser exata, sendo admitido um valor de 0.05 μs de diferença entre os tempos-de-voo. Caso um pico correspondente seja encontrado, suas informações – como tempo de voo máximo, contagem máxima e área – são armazenadas no dicionário inicializado. Caso contrário, valores NAN são atribuídos ao dicionário.

Após o mapeamento de todos os picos de emissão presentes nos arquivos listados, converte-se o dicionário com as informações de comparação em um DATAFRAME que é salvo em uma tabela com formato .TSV. Esta tabela será utilizada na segunda parte desta etapa.

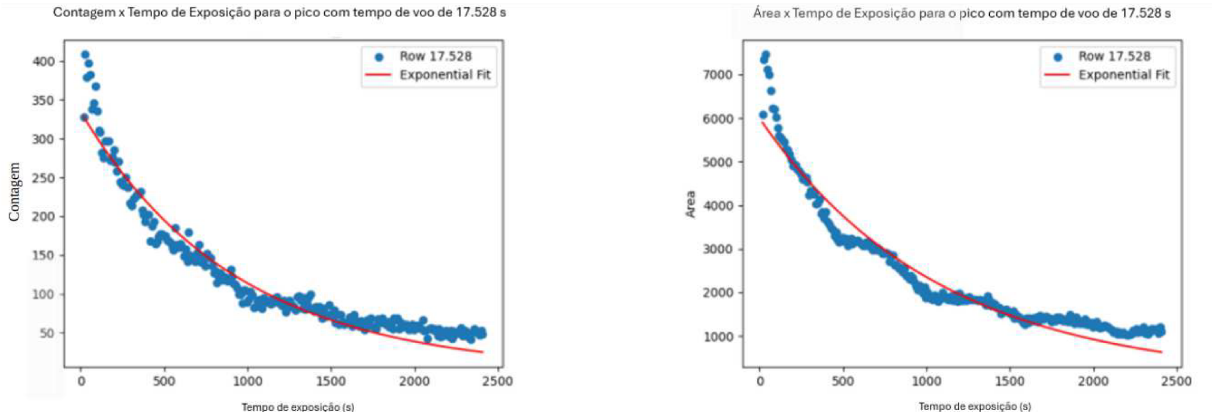
4.2.2 Ajuste

Seguindo o fluxo do processo, na segunda parte desta etapa utiliza-se a tabela gerada na etapa anterior para construir gráficos de tempo de exposição por contagem ou área para os picos mapeados anteriormente. Para isso, utiliza-se o método ADJUSTED_SPECS – presente na classe DATAPROCESSOR3.

Por meio deste método é possível ler o arquivo em formato de tabela gerado anteriormente em forma de DATAFRAME, e para cada linha deste DATAFRAME os valores das colunas são extraídos e processados, formando assim um conjunto de pontos de dados (x, y) para o ajuste da função. O valor de x é calculado a partir do valor do tempo-de-voo somado a um deslocamento linear baseado no índice da coluna – o incremento temporal entre cada espectro obtido sequencialmente. Os pontos de dados são então usados para ajustar duas curvas diferentes: (i) exponencial, Equação (14); e (ii) exponencial modificada, Equação (15).

$$g_{\text{exp}}(x) = a \times \exp(bx) \tag{14}$$

Figura 23 – Exemplo de gráficos do número máximo de contagens e da área para o pico de emissão observado em 17,5280 μs em função do tempo de exposição ao feixe contínuo de íons primários. Os dados foram obtidos a partir do mapeamento de picos e ajuste de uma curva exponencial para espectros do experimento **PMMA_6MeV_Cu**.



$$g_{\text{mod}}(x) = a \times \exp\left[-k_1 \frac{1-\exp(-k_2 x)}{k_2 x}\right] \tag{15}$$

onde a, b, k_1 e k_2 são parâmetros ajustáveis.

Para cada linha de dados, ocorre a plotagem de um gráfico de dispersão dos pontos de dados junto com as curvas ajustadas. Estas se ajustam tanto para o número máximo de contagens por tempo de exposição quanto para área por tempo de exposição. Os gráficos resultantes – um exemplo típico é dado na Figura 23 – são salvos em arquivos no formato .PNG no diretório especificado.

Nem sempre foi possível realizar um ajuste saisfatório para a exponencial modificada, sendo necessário um estudo mais aprofundado acerca dos parâmetros utilizados no ajuste. Por isso, decidiu-se realizar o gráfico e salvar os resultados somente para o ajuste da exponencial.

Os resultados do ajuste são salvos em um arquivo no formato .CSV no diretório especificado, contendo três colunas: (i) valor do tempo-de-voo máximo do pico de emissão; (ii) tipo de função ajustada; e (iii) uma lista com os parâmetros ajustados.

4.3 ÍONS SECUNDÁRIOS

A partir dos valores de tempo-de-voo é possível encontrar qual o íon responsável por cada pico de emissão encontrado no espectro. Para isso, utiliza-se a Equação (16), característica de calibrações em espectrômetros de massa.

$$M = \frac{(ToF - t_0)^2}{(Cb)^2} \tag{16}$$

onde *M* é a massa do íon secundário em unidades de massa atômica, *ToF* o tempo-de-voo com máximo número de contagens para cada pico de emissão em microssegundos, t_0 e *Cb* parâmetros ajustados a uma curva de calibração (respectivamente 2,5904 e 0,1445). Para tal próposito, foi utilizado um espectro obtido através da irradiação de PS, onde a emissão de hidrogênio e da molécula C_7H_7 são bem características para fazer uma calibração inicial. Dessa calibração inicial é feita a identificação de outras moléculas – no intervalo entre 1 e 300 unidades de massa atômica – para o ajuste da função descrita na Equação (16).

Para realizar este cálculo de forma automatizada para cada conjunto de picos encontrados em cada experimento, utiliza-se o método FIND_MASS –encontrado na classe DATAPROCESSOR4. Este método recebe o caminho onde o arquivo .CSV com os valores de tempo-de-voo e parâmetros ajustados – gerados na etapa anterior – estão salvos. Recebe também um parâmetro booleano, que indica se o cálculo para encontro das massas será feito a partir do ajuste de área – caso verdadeiro – ou por ajuste de contagem – caso falso.

A partir do caminho passado como argumento, é possível ler o arquivo .CSV em um DATAFRAME, utilizando o método READ_FILES – presente na classe FILES. Para cada linha deste DATAFRAME, ou seja, para cada tempo-de-voo relacionado ao máximo de contagens num pico, realiza-se a operação da Equação (16), e cria-se uma nova coluna neste DATAFRAME com o valor desta massa calculada para cada linha, a partir do método CAL_EJECTED_PART_MASS – pertencente a classe CALC.

A seguir utilizando-se das tabelas de massas e tempo-de-voo para hidrocarbonetos, óxidos e cloretos, afere-se quais partículas podem ter sido ejetadas, comparando suas massas com as massas calculadas anteriormente. Para isto, lê-se estas tabelas em um outro DATAFRAME, utilizando o método READ_CSV_MULTIPLE_SHEETS. Entretanto, a tabela lida dependerá da amostra presente em cada experimento.

Para experimentos com amostras de PS, foram utilizadas tabelas com massas de hidrocarbonetos. Já para as amostras de PMMA foram utilizadas tabelas com massas de hidrocarbonetos e de óxidos. Por fim, para os experimentos com amostras de PVC foram usadas tabelas com massas de hidrocarbonetos e cloretos.

A partir das colunas de massa presentes nestes dois DATAFRAMES é calculada a diferença entre elas através do método COMPARE_MASS – presente na classe CALC. Caso a diferença entre as massas seja de 0.01, 0.05 ou 0.10 retorna-se uma lista de DATAFRAMES, um para cada tipo de familia de elementos. Estes DataFrames são formados pela quantidade de H e C, para todos os experimentos, H, C e O para experimentos com amostras de PMMA e H, C e Cl-35 e H, C e Cl-37 – onde os valores numéricos indicam o isótopo do Cloro sendo considerado – para experimentos com amostras de PVC, além do tempo-de-voo tabelado e calculado, o intervalo de diferença entre as massas e as massas. Estes DATAFRAMES são salvos em arquivos de texto

através do método SAVE_DATAFRAMES_AS_TXT – presente na classe FILES.

4.4 SEÇÃO DE CHOQUE PARA DANOS POR IRRADIAÇÃO

Esta etapa do *script* é responsável por calcular a seção de choque e o raio dessa seção. A seção de choque, σ, é calculada a partir da Equação (17), onde b é o parametro ajustado da exponencial, Equação (15), e φ é a fluxo médio de íons primários.

$$\sigma = \frac{b}{\varphi} \tag{17}$$

O fluxo médio foi obtido através da carga integrada total medida para cada experimento. As condições dos experimentos e seus valores de fluxos estão apresentados na Tabela 2. Através do método READ_FLUENCE_TABLE a tabela é lida como um DATAFRAME e, a partir dos parâmetros do próprio DATAFRAME, cria-se uma nova coluna com o nome do experimento.

Assim, um cruzamento utilizando o método SELECTING_RIGHT_EXPERIMENT_ROW é feito entre esta nova coluna com o nome do experimento e o caminho do experimento analisado, retornando o valor do fluxo para o experimento em questão. O arquivo .CSV contendo os valores da função exponencial ajustada é também carregado como DATAFRAME, através do método READ_CSV. E para cada linha é calculada a seção de choque, utilizado o método CAL_CROSS_SECTION.

Para finalizar a etapa de análise dos espectros de degradação, calcula-se o raio da seção de choque a partir da Equação (18), onde supõe-se uma seção de choque circular de raio r.

$$r = \sqrt{\frac{\sigma}{\pi}} \tag{18}$$

Tabela 2 – Informações dos íons primários para o cálculo da seção de choque.

Alvo	**Íon**	**Carga**	**Energia**	**Duração (s)**	**Fluxo (10^{10} íons cm^{-2} s^{-1})**
PMMA	Cu	4	6,0	2532	1,52
PS	Cl	5	12,0	1987	1,58
PMMA	Cl	5	12,0	1995	1,63
PS	Cl	5	9,0	1992	2,33
PMMA	Cl	5	9,0	1992	2,32
PS	Cl	5	6,0	1997	2,55
PS	Cu	4	6,0	934	3,40
PVC	Cu	4	6,0	2790	2,29
PVC	Cu	4	6,0	930	1,89
PVC	Cu	4	6,0	interrompido por queda de luz	

Já os erros para a seção de choque e raio foram calculados a partir da propragação de erro, sendo o erro de *b* propagado para calcular o erro da seçção de choque através da Equação (19) e o erro do raio foi calculado propagando o erro da seção de choque a partir da Equação (20).

$$error_\sigma = \sqrt{\left(\frac{\partial\sigma}{\partial b}\right)^2 \times (erro_b)^2 + \left(\frac{\partial\sigma}{\partial\varphi}\right)^2 \times (erro_\varphi)^2} \tag{19}$$

$$erro_r = \sqrt{\left(\frac{\partial r}{\partial\sigma}\right)^2 \times (erro_\sigma)^2} \tag{20}$$

Apesar do erro do fluxo média ser de cerca de 10% de seu valor, escolheu-se por não propagar este erro para o cálculo de erro da seção de choque, pois este é um tipo de erro sistemático. Dessa forma, apenas o erro em *b* foi utilizado para propagação.

4.5 IDENTIFICAÇÃO DE POSSÍVEIS MOLÉCULAS

A última etapa do software identifica possivéis moléculas ejetadas. Para isto, é realizada uma comparação entre os tempos de voos calculados para cada molécula e os tempos de voo de cada pico. Este processo é feito pelo método COMPARE_FINAL_RESULTS, recebendo como paramêtros o diretório base onde estão salvos os arquivos .CSV e uma FLAG indicando se os picos são referentes a área ou não.

O arquivo .CSV criado na etapa anterior composto pelas seção de choque e raio é lido em forma de DATAFRAME. Este DATAFRAME é então filtrado para excluir valores nulos em qualquer coluna e valores em que o erro da seção de choque seja maior que 20% do valor de sua seção de choque.

A seguir, DATAFRAMES compostos pela massas e tempo-de-voos para hidrocarbonetos, óxidos e cloretos são carregados separadamente, de acordo com o tipo de experimento realizado. O cálculo do tempo-de-voo é feito através da Equação (16). No Quadro 6 é possível observar as famílias de elementos carregadas para cada experimento.

Para cada DATAFRAME composto pelas famílias de elementos compara-se os tempo-de-voo calculados com os tempos-de-voo observados presentes no DATAFRAME com as seções de choque. Caso haja alguma correspondência os valores de seção de choque, erro da seção de choque, raio, erro do raio e a fórmula para aquele elemento correspodente é acrescentado ao DATAFRAME composto pelas famílias. Caso não haja correspondência para alguma linha, esta linha é excluida. Por fim, este DATAFRAME é salvo em um arquivo .CSV, resultando em um arquivo por família.

Quadro 6 – Número de espectros e famílias analisadas em cada experimento.

Experimento	Número Espectros	Familia de Elementos
PMMA_6_MeV_Cu	239	Hidrocarbonetos e Óxidos
PS_12_MeV_Cl	359	Hidrocarbonetos
PMMA_12_MeV_Cl	359	Hidrocarbonetos e Óxidos
PS_9_MeV_Cl	359	Hidrocarbonetos
PMMA_9_MeV_Cl	359	Hidrocarbonetos e Óxidos
PS_6_MeV_Cl	359	Hidrocarbonetos
PS_6_MeV_Cu	600	Hidrocarbonetos
1_PVC_6_MeV_Cu	1799	Hidrocarbonetos e Cloretos
2_PVC_6_MeV_Cu	599	Hidrocarbonetos e Cloretos
3_PVC_6_MeV_Cu	2910	Hidrocarbonetos e Cloretos

4.6 RESULTADOS

Para cada experimento apresentado no Quadro 1 foram analisados diferentes quantidades de espectros. Para cada um deles as seguintes análises foram feitas:

1. Encontrou-se os picos em cada espectro;
2. A partir dos tempos-de-voo para os valores máximos de contagem encontrados em cada pico, ajustou-se uma função exponencial, Equação (15), relacionando os tempos de exposição com a contagem e com a área de cada pico em todos os espectros analisados. Utilizou-se os picos do décimo espectro como referência, sendo então importante ressaltar que alguns ruídos podem ter sido classificados erroneamente como picos, gerando gráficos e ajustes equivocados;
3. Para cada pico do espectro de referência, encontrou-se a massa referente àquele pico utilizando seu tempo-de-voo através da Equação (16). E através do ajuste exponencial, sua seção de choque e o raio dessa seção de choque, a partir das Equação (18) e Equação (19), respectivamente;
4. Utilizando tabelas com massas e tempo-de-voo de hidrocarbonetos, óxidos e cloretos pode-se comparar quais elementos foram responsáveis pela emissão de cada pico identificado.

O Quadro 6 apresenta a quantidade de espectros analisados para cada experimento e a família de elementos em que suas massas foram comparadas.

5 DISCUSSÃO

O programa desenvolvido nesse trabalho foi capaz de realizar a análise de degradação por feixe de íons em amostras contendo polímeros através do estudo de espectros gerados por espectrometria de massa de íons secundários utilizando um analisador do tipo tempo-de-voo.

Foram analisados dez experimentos – um interrompido por falta de luz – com oito condições diferentes de feixe entre 2017 e 2018 na linha de espectrometria de massa do LII-UFRGS. Através do desenvolvimento de um programa em linguagem PYTHON, foi possível analisar todos os espectros gerados por cada um desses experimentos de forma automatizada. A seguir, encontram-se os resultados obtidos pelo programa para tais experimentos.

5.1 EMISSÃO DE ÍONS SECUNDÁRIOS

Utilizando a etapa de identificação de moléculas presente no programa foi possível identificar os trinta principais hidrocarbonetos emitidos após a incidência do feixe de íons primários nas amostras, conforme observa-se pela Figura 24. Esta análise foi feita levando em consideração o conjunto de dados para contagem máxima e todos os experimentos executados. Em contraste, a Figura 25 representa, dentre os hidrocarbonetos identificados, aqueles menos emitidos. A Figura 26 mostra os principais óxidos emitidos após a incidência do feixe de íons nas amostras. Para esta análise também foi levado em consideração o conjunto de dados para contagem máxima e apenas os experimentos com amostras de PMMA. Em contraste, a Figura 27 representa, dentre os óxidos identificados, aqueles menos emitidos. Por fim, as Figura 28 e Figura 29 mostram os principais cloretos emitidos contendo os isótopos 35 e 37, respectivamente, do Cloro emitidos após a incidência do feixe de íons primários nas amostras. Para esta análise também foi levado em consideração o conjunto de dados para contagem máxima e apenas os experimentos com amostras de PVC. Em contraste, as Figura 30 e Figura 31 representam, dentre os cloretos identificados para os isótopos 35 e 37 do Cloro, aqueles menos emitidos.

De uma maneira geral, observa-se que hidrocarbonetos contendo mais átomos de Carbono do que de Hidrgênio são preferencialmente emitidos, enquanto o caso inverso representa os menos observados. Para os óxidos, aqueles contendo apenas um átomo de Oxigênio parecem ser preferencialmente emitidos. As correlações são menos claras no caso dos cloretos.

Figura 24 – Principais hidrocarbonetos emitidos como íons secundários (contagem).

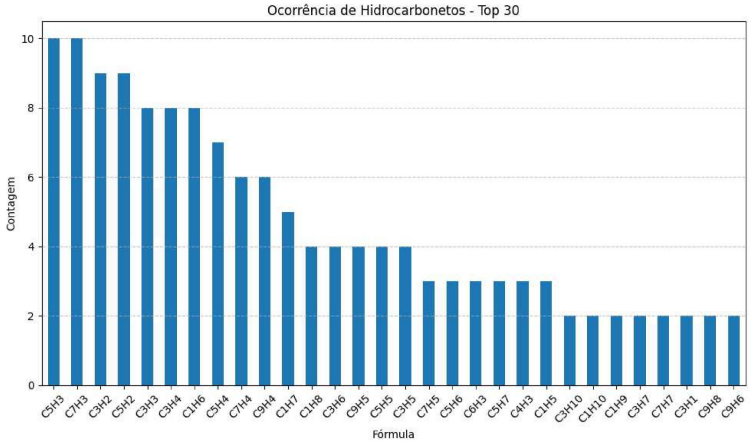


Figura 25 – Hidrocarbonetos menos emitidos como íons secundários (contagem).

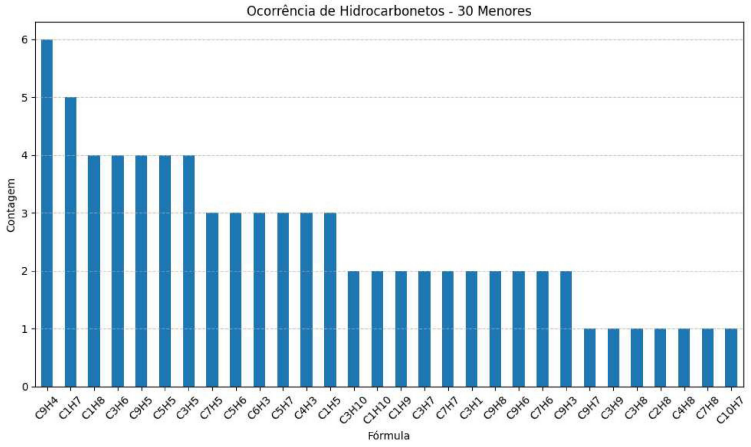


Figura 26 – Principais óxidos emitidos como íons secundários (contagem).

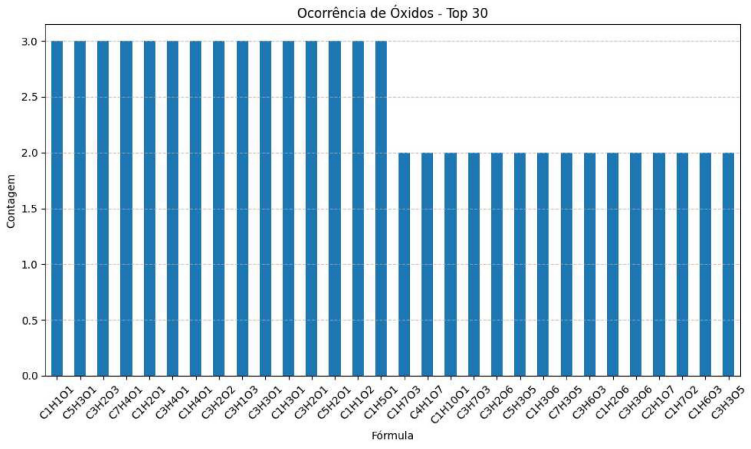


Figura 27 – Óxidos menos emitidos como íons secundários (contagem).

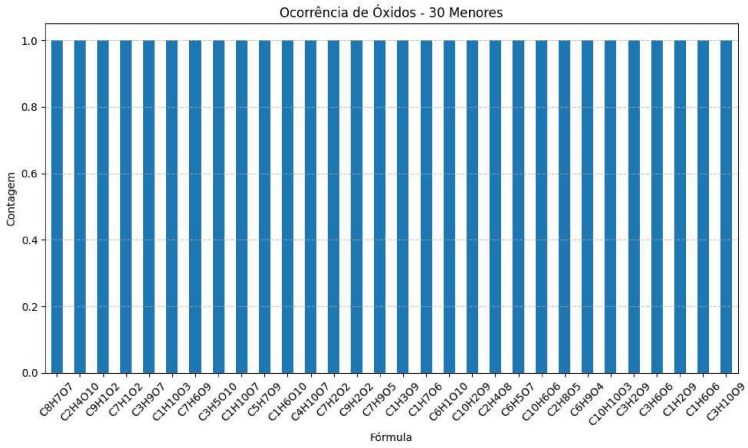


Figura 28 – Principais cloretos (Cl-35) emitidos como íons secundários (contagem).

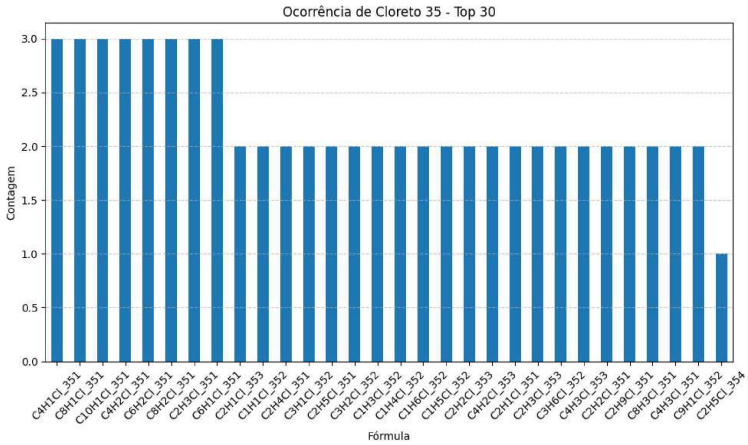


Figura 29 – Principais cloretos (Cl-37) emitidos como íons secundários (contagem).

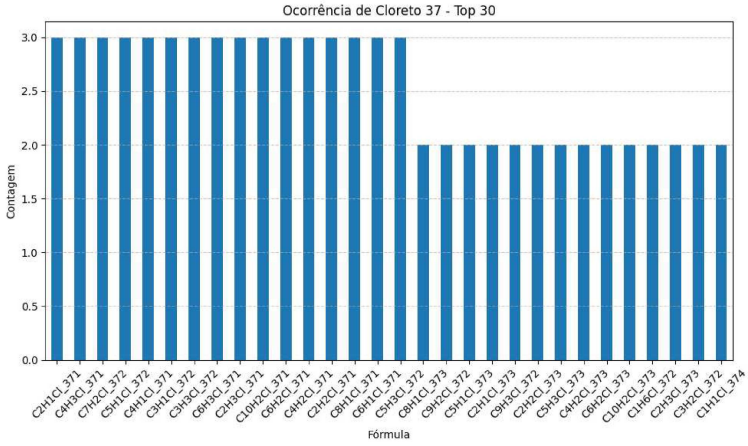


Figura 30 – Cloretos (Cl-35) menos emitidos como íons secundários (contagem).

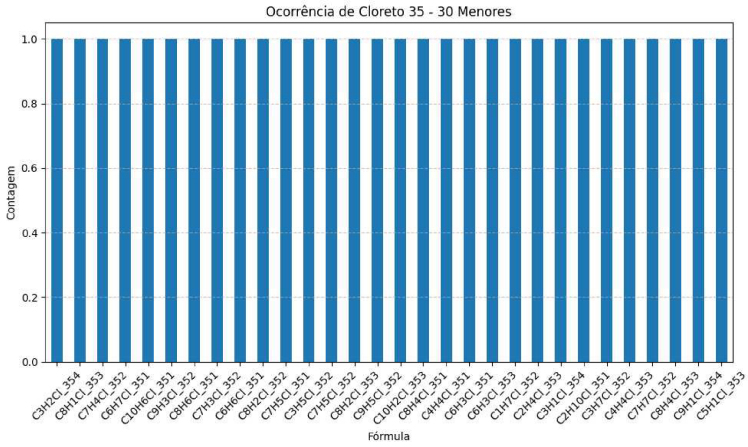
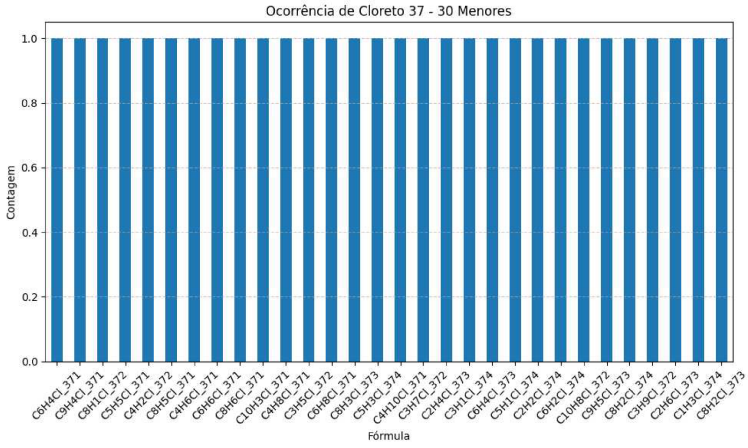


Figura 31 – Cloretos (Cl-37) menos emitidos como íons secundários (contagem).



Já os principais hidrocarbonetos, óxidos e cloretos (isótopos 35 e 37 do Cloro) levando em consideração a análise da área no lugar da contagem podem ser observados nas Figura 32, Figura 33, Figura 34 e Figura 35, respectivamente. Enquanto as Figura 36, Figura 37, Figura 38 e Figura 39, dentre os grupos moleculares identificados, aqueles menos emitidos.

A Tabela 3 mostra os hidrocabonetos, oxidos, cloretos (isótopos 35 e 37 do Cloro) de maior e menor massa idenficados nos espectros analisados pelo programa.

A identificação das moléculas secundárias emitidas durante o desbastamento eletrônico de filmes poliméricos é muito importante para investigar os mecanismos de cissão das cadeias. Assim, esta análise é apenas preliminar, pois fez-se necessário olhar a correlação de quebra com a molécula principal da amostra. Este será um tema para investigações futuras que poderão ser implementadas no programa.

Figura 32 – Principais hidrocarbonetos emitidos como íons secundários (área).

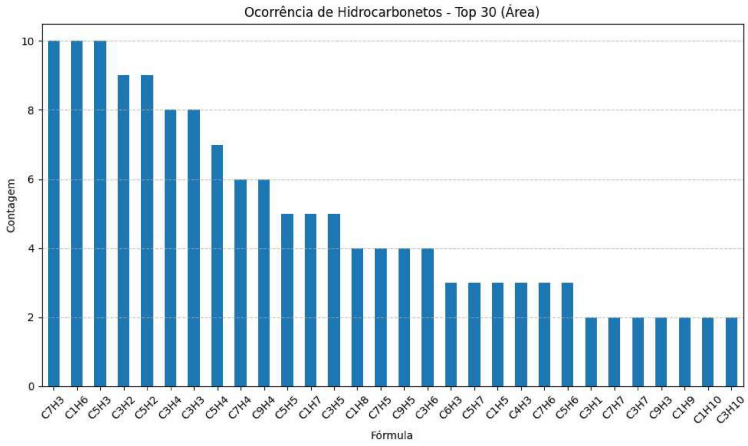


Figura 33 – Principais óxidos emitidos como íons secundários (área).

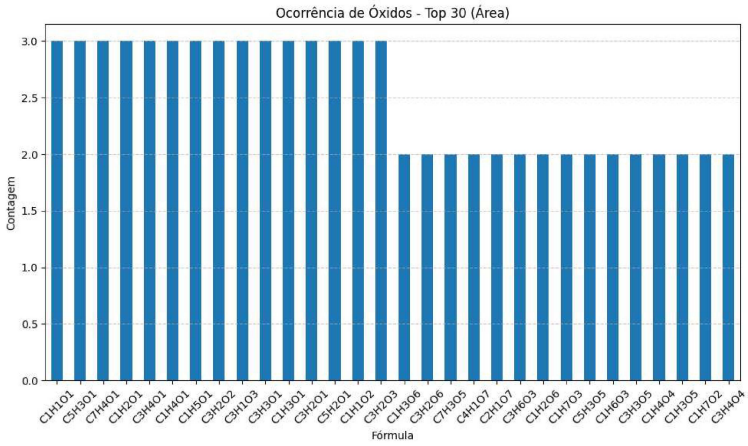


Figura 34 – Principais cloretos (Cl-35) emitidos como íons secundários (área).

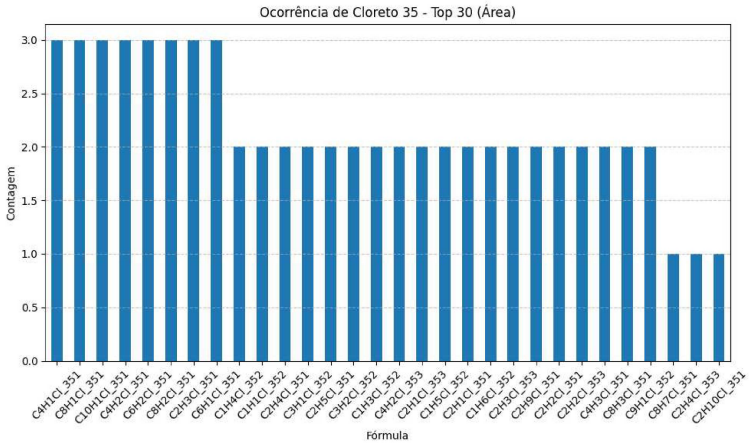


Figura 35 – Principais cloretos (Cl-37) emitidos como íons secundários (área).

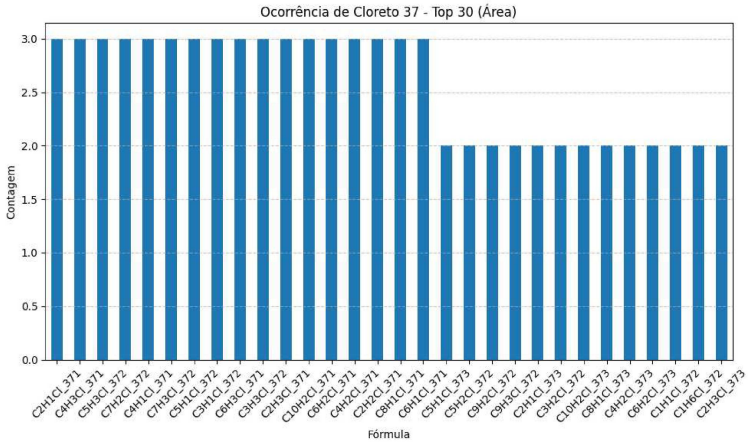


Figura 36 – Hidrocarbonetos menos emitidos como íons secundários (área).

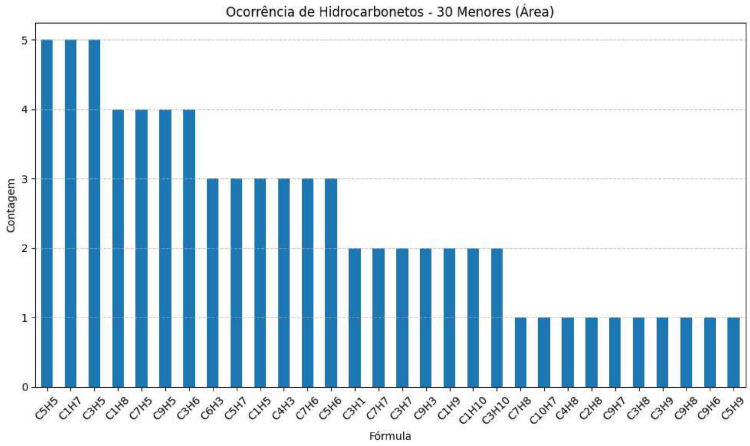


Figura 37 – Óxidos menos emitidos como íons secundários (área).

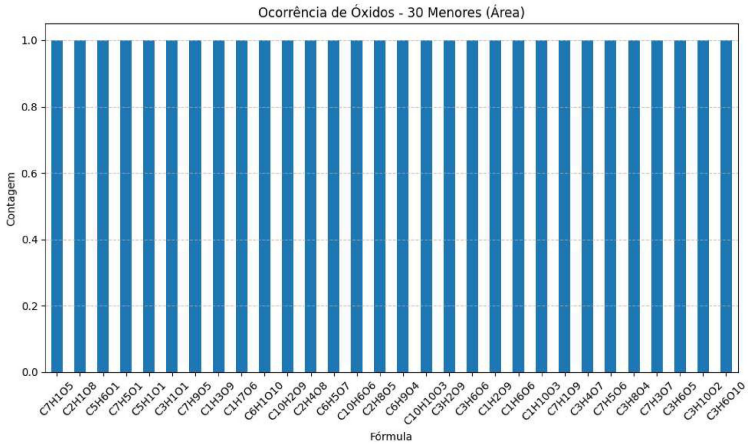


Figura 38 – Cloretos (Cl-35) menos emitidos como íons secundários (área).

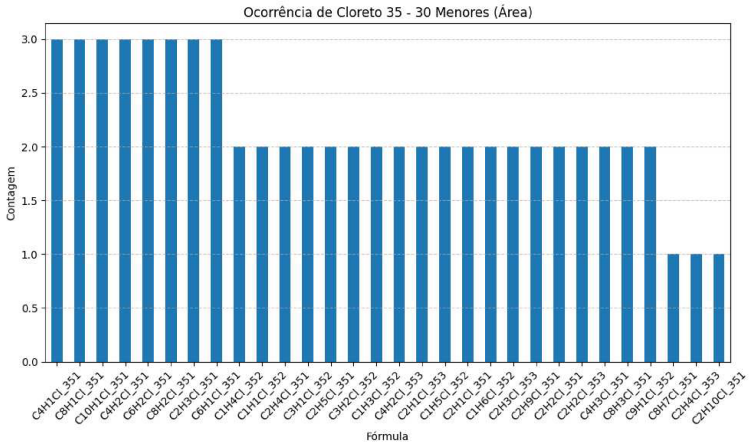


Figura 39 – Cloretos (Cl-37) menos emitidos como íons secundários (área).

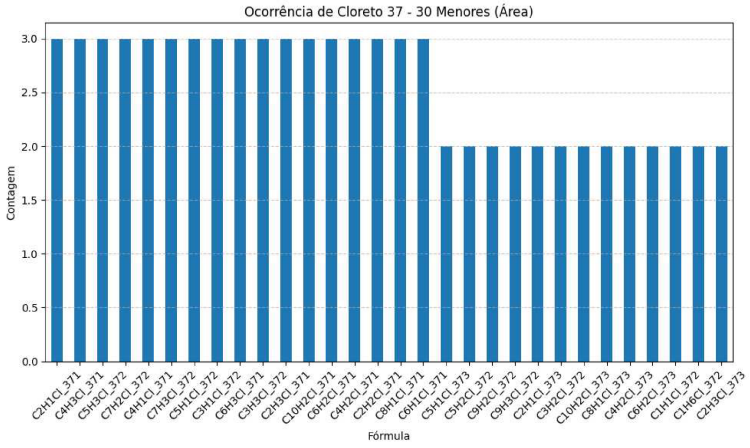


Tabela 3 – Tabela com hidrocarbonetos, óxidos, cloretos (isótopos 35 e 37) de maior e menor massa emitidos durante os experimentos.

Grupo	**Fórmula**	**Massa (u)**	**Tempo de voo (μs)**
Hidrocarbonetos	C_3H_1	14,9774	10,1695
	C_3H_{10}	122,958	28,8685
Óxidos	$C_1H_1O_1$	28,9699	14,0870
	$C_3H_{10}O_9$	266,931	42,4665
Cloretos (Cl-35)	$C_2H_1Cl_1$	48,8850	18,2560
	$C_2H_2Cl_6$	235,870	39,9280
Cloretos (Cl-37)	$C_2H_1Cl_1$	50,9165	18,6285
	$C_4H_7Cl_4$	235,8698	39,9280

5.2 HIDROCARBONETOS

Para realizar a análise dos hidrocarbonetos identificados nos espectros de emissão encontrados pelo programa foram utilizados todos os experimentos. A partir destes dados, pode-se verificar a relação da seção de choque com a massa do hidrocarboneto emitido, levando em conta os valores de contagem máxima e de área do pico responsável por esta emissão, respectivamente, conforme observa-se nas Figura 40 e Figura 41.

Figura 40 – Relação entre seção de choque e massa para hidrocarbonetos emitidos (contagem).

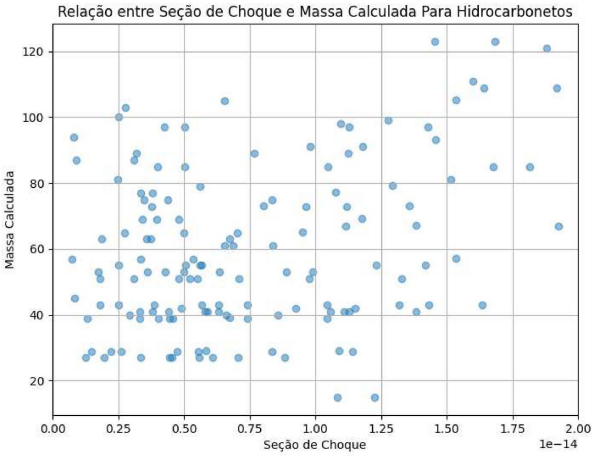
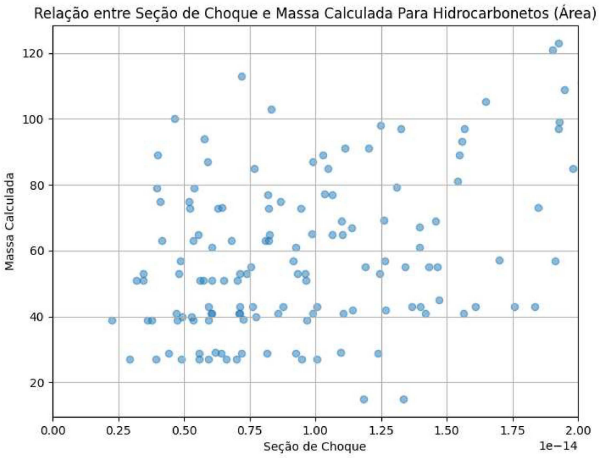


Figura 41 – Relação entre seção de choque e massa para hidrocarbonetos emitidos (área).



Nota-se, a partir das Figura 40 e Figura 41, que não existe uma correlação clara entre seção de choque e massa para os hidrocarbonetos. Os pontos encontram-se consideravelmente dispersos.

Outra análise capaz de ser realizada através dos dados obtidos pelo programa é a frequência da seção de choque para danos obtidas pela análise dos hidrocarbonetos, conforme observa-se pelas Figura 42 e Figura 43. A Figura 42 analisa a frequência de seção de choque a partir dos valores de contagem máxima do pico, e a Figura 43 a partir da área.

Figura 42 – Histograma de frequência de seção de choque para hidrocarbonetos (contagem).

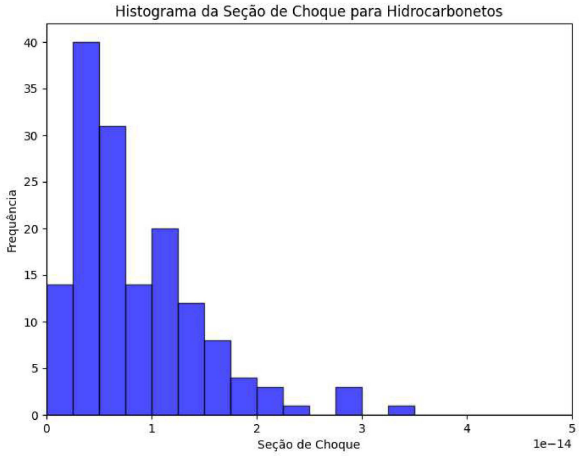


Figura 43 – Histograma de frequência de seção de choque para hidrocarbonetos (área).

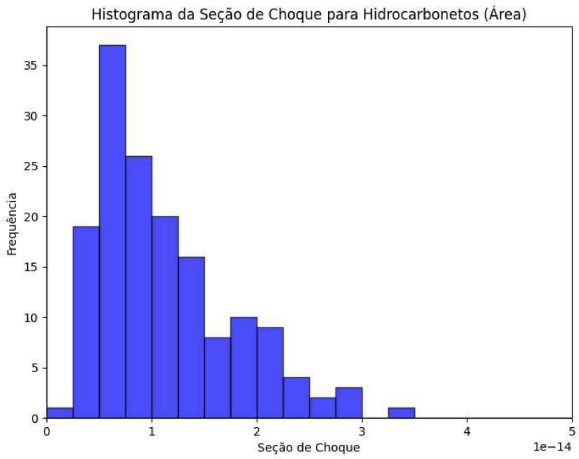


Tabela 4 – Tabela estatística para a seção de choque dos hidrocarbonetos.

	σ (10^{-14} **cm^2**)	**Raio (nm)**
Contagem		
Média	0,85 $\pm$ 0,03	0,49 $\pm$ 0,09
Mediana	0,65 $\pm$ 0,03	0,46 $\pm$ 0,08
90 % Percentil	1,63 $\pm$ 0,05	0,72 $\pm$ 0,13
Área		
Média	1,13 $\pm$ 0,05	0,60 $\pm$ 0,13
Mediana	0,96 $\pm$ 0,03	0,55 $\pm$ 0,07
90 % Percentil	2,09 $\pm$ 0,09	0,81 $\pm$ 0,17

As distribuições de seção de choque são bastante assimétricas. Observa-se uma tendência às seções de choque calculadas a partir da contagem máxima dos picos de emissão a se concentrarem na região próxima a 0,8 $\times$ 10^{-14} cm^2. Já as seções de choque calculadas a partir da área dos picos de emissão possuem uma tendência de se concentrarem em valores para próximos à 1,0 $\times$ 10^{-14} cm^2.

A partir dos cálculos de seção de choque e raio realizados pelo programa, construiu-se a Tabela 4, que mostra os valores de média, mediana e percentil de 90 % para os valores de seção de choque para hidrocarbonetos, tanto para as calculadas a partir da contagem máxima quanto a partir da área. Uma diferença existe entre os valores de seção de choque e de raio quando comparam-se as duas formas de calculá-los, sendo a média dos valores de seção de choque aproximadamente 33% maior quando utiliza-se a área do pico para calculá-la ao invés da contagem máxima para os picos. Já para a média do raio este valor é de cerca de 17%. Para as demais estatísticas, esta diferença é de 48% para a mediana da seção de choque e de 21% para a mediana do raio. Por fim, para o percentil é de 28% e 13% para seção de choque e raio, respectivamente.

5.3 ÓXIDOS

A análise dos óxidos identificados como íons secundários foi feita para os experimentos que continham amostras de PMMA. Novamente, procurou-se uma relação entre a seção de choque com as massas de óxidos emitidos, a partir dos cálculos de seção de choque e raio realizados pelo programa. Assim como no caso dos hidrocarbonetos, não foi possível encontrar uma correlação entre ambas. As Figura 44 e Figura 45 apresentam os respectivos gráficos, levando em conta a análise dos valores de contagem máxima e da área do pico responsável pela emissão, respectivamente.

Figura 44 – Relação entre seção de choque e massa para óxidos emitidos (contagem).

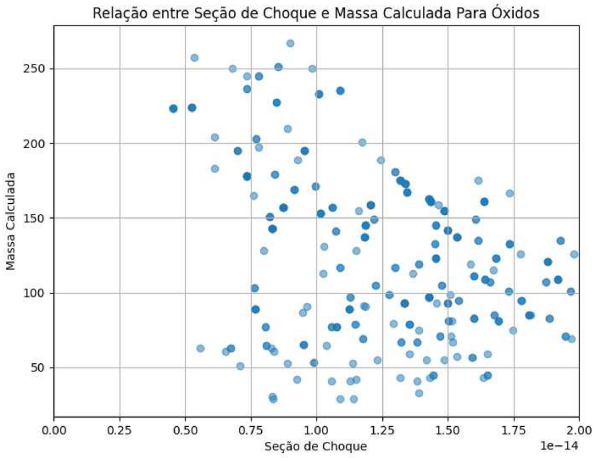
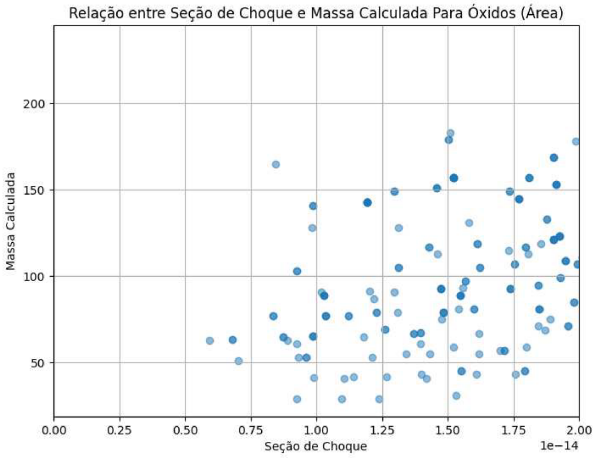


Figura 45 – Relação entre seção de choque e massa para óxidos emitidos (área).



Os histogramas de frequência da seção de choque estão apresentados nas Figura 46 e Figura 47, onde é possível observar um deslocamento na posição mais provável da distribuição para valores maiores em relação aos hidrocarbonetos. Já a Tabela 5 representa as estatísticas de seção de choque e raio obtidos para os óxidos.

Figura 46 – Histograma de frequência de seção de choque para óxidos (contagem).

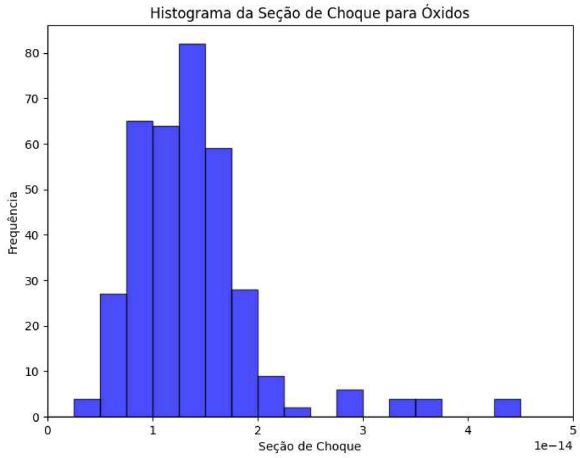


Figura 47 – Histograma de frequência de seção de choque para óxidos (área).

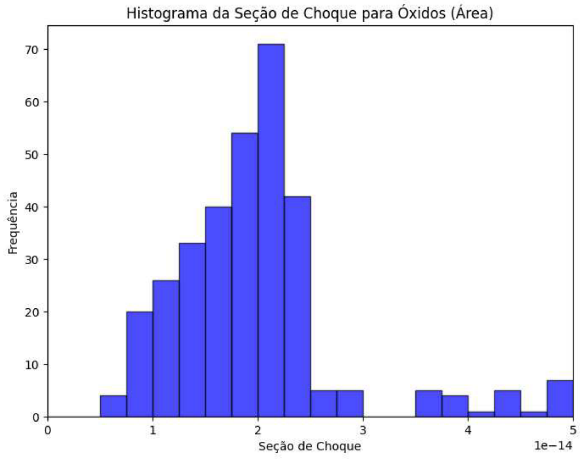


Tabela 5 – Tabela estatística para seção de choque dos óxidos.

	σ (10^{-14} **cm^2**)	**Raio (nm)**
Contagem		
Média	1,39 $\pm$ 0,05	0,67 $\pm$ 0,13
Mediana	1,34 $\pm$ 0,04	0,65 $\pm$ 0,11
90% Percentil	1,93 $\pm$ 0,08	0,78 $\pm$ 0,16
Área		
Média	2,10 $\pm$ 0,10	0,82 $\pm$ 0,25
Mediana	1,93 $\pm$ 0,05	0,78 $\pm$ 0,13
90% Percentil	2,80 $\pm$ 0,10	0,94 $\pm$ 0,18

Nota-se que existe, novamente, uma diferença entre os valores de seção de choque e de raio quando comparam-se as duas formas de calculá-los. A média dos valores de seção de choque é aproximadamente 53 % maior quando utiliza-se a área do pico para calculá-la ao invés da contagem máxima para os picos, enquanto a média para os valores de raios é aproximadamente 23 % maior para cálculos com a área do que a contagem máxima. Este comportamento também se repete para as demais estatísticas, sendo a mediana para seção de choque aproximadamente 23 % maior e o raio cerca de 20 % maior quando se utiliza os parâmetros obtidos através do ajuste da curva exponencial à área do pico em comparação à contagem máxima. Para o percentil de 90 %, a seção de choque é 44 % maior e o raio é 20 %.

5.4 CLORETOS

Por fim, a analise dos cloretos identificados como íons secundários foi feita para os experimentos que continham amostras de PVC. Esta análise foi realizada para dois isótopos diferentes de cloro, Cl-35 e Cl-37. Mais uma vez, procurou-se alguma relação entre a seção de choque com as massas de cloretos emitidos. Assim como nos casos anteriores, não foi possivel encontrar uma correlação entre ambas. Figura 48, Figura 49, Figura 50 e Figura 51 apresentam os respectivos gráficos para os isótopos 35 e 37 do Cloro, levando em conta a analise dos valores de contagem máxima e da área do pico responsável pela emissão.

Figura 48 – Relação entre seção de choque e massa para cloretos (Cl-35) emitidos (contagem).

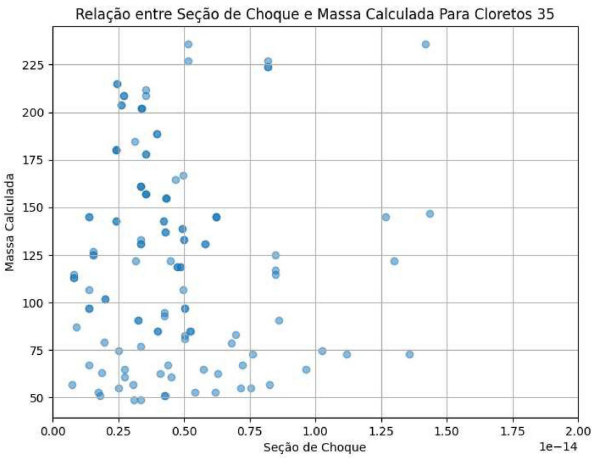


Figura 49 – Relação entre seção de choque e massa para cloretos (Cl-35) emitidos (área).

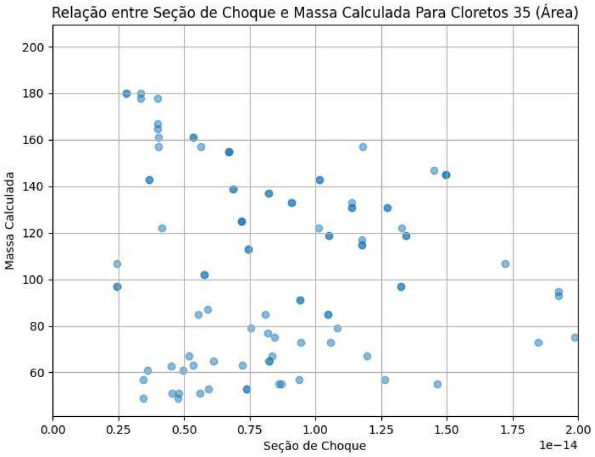


Figura 50 – Relação entre seção de choque e massa para cloretos (Cl-37) emitidos (contagem).

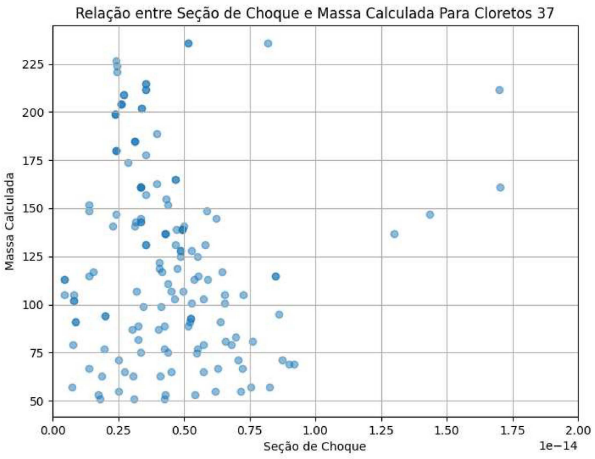
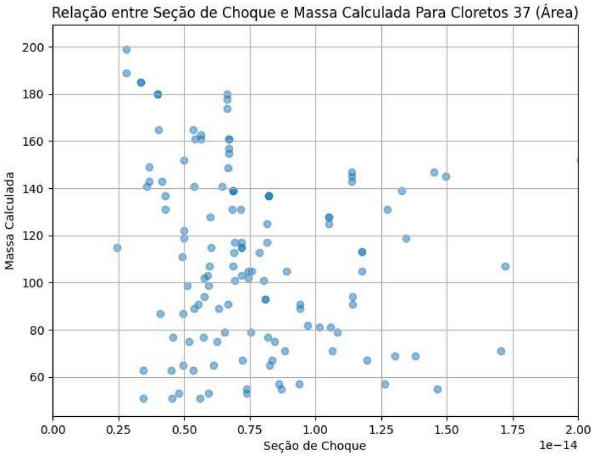


Figura 51 – Relação entre seção de choque e massa para cloretos (Cl-37) emitidos (área).



Os histogramas de frequência da seção de choque podem ser observados nas Figura 52 e Figura 53 para o isótopo 35 do Cloro e nas Figura 54 e Figura 55 para o isótopo 37 do Cloro.

Figura 52 – Histograma de frequência de seção de choque para cloretos (Cl-35, contagem).

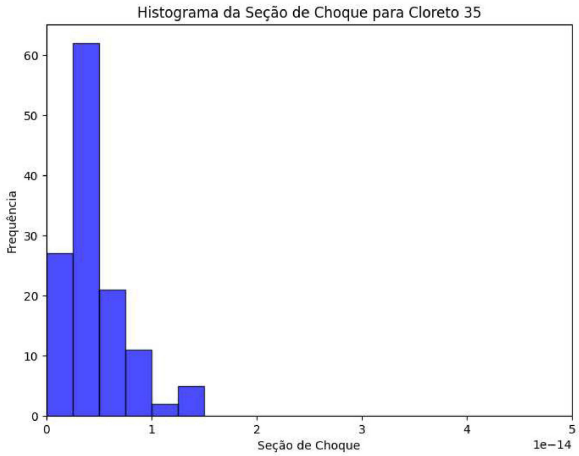


Figura 53 – Histograma de frequência de seção de choque para cloretos (Cl-35, área).

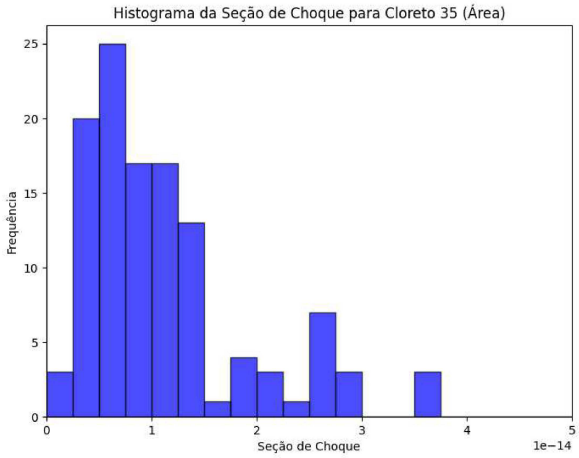


Figura 54 – Histograma de frequência de seção de choque para cloretos (Cl-37, contagem).

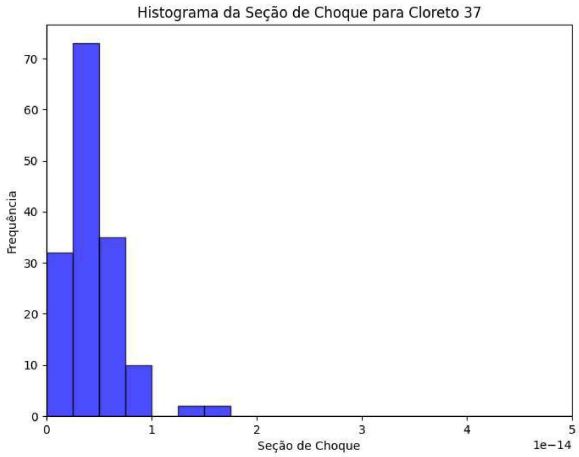
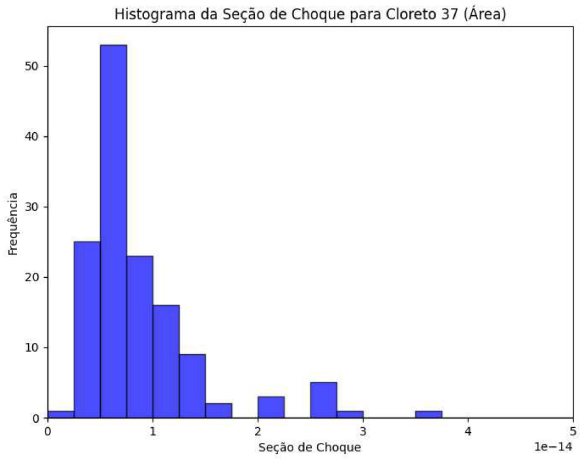


Figura 55 – Histograma de frequência de seção de choque para cloretos (Cl-37, área).



Os valores estatísticos de média, mediana e percentil 90 % para os isótopos 35 e 37 do Cloro podem ser encontrados nas Tabela 6 e Tabela 7, respectivamente. Comparando as duas abordagens de calcular os parâmetros estatísticos, a diferença da média da seção de choque e do raio para o isótopo 35 do Cloro é cerca de 145 % e 55 %, respectivamente. Já para o isótopo 37, tem-se uma diferença para seção de choque e para o raio de 100 % e 28 %, respectivamente.

Tabela 6 – Tabela de estatística para seção de choque dos cloretos (Cl-35).

	σ (10^{-14} **cm^2**)	**Raio (nm)**
Contagem		
Média	0,46 $\pm$ 0,03	0,38 $\pm$ 0,10
Mediana	0,42$\pm$ 0,02	0,37 $\pm$ 0,08
90% Percentil	0,82 $\pm$ 0,06	0,51 $\pm$ 0,14
Área		
Média	1,12 $\pm$ 0,07	0,60 $\pm$ 0,15
Mediana	0.87 $\pm$ 0,04	0,53 $\pm$ 0,13
90% Percentil	2,50 $\pm$ 0,20	0,89 $\pm$ 0,25

Tabela 7 – Tabela de estatística para seção de choque dos cloretos (Cl-37).

	σ (10^{-14} **cm^2**)	**Raio (nm)**
Contagem		
Média	0,44 $\pm$ 0,02	0,38 $\pm$ 0,08
Mediana	0,41 $\pm$ 0,02	0,36 $\pm$ 0,08
90% Percentil	0,74 $\pm$ 0,05	0,49 $\pm$ 0,13
Área		
Média	0.88 $\pm$ 0,04	0,53 $\pm$ 0,11
Mediana	0,72 $\pm$ 0,02	0,48 $\pm$ 0,08
90% Percentil	1,50 $\pm$ 0,10	0,69 $\pm$ 0,18

5.5 OUTRAS COMPARAÇÕES

Duas comparações das diferenças entre os tamanhos dos raios referentes à seção de choque nos hidrocarbonetos, óxidos e cloretos – considerando os dois isótopos de Cloro – são apresentadas nas Tabela 8 e Tabela 9 para as análises baseadas na contagem máxima e na área dos picos de emissão, repectivamente. A maior diferença encontrada para análise da contagem máxima é quando comparam-se óxidos e cloretos, enquanto a maior diferença para a área é quando compara-se hidrocarbonetos e óxidos. Já a menor diferença na comparação utilizando a contagem ocorre entre os isótopos de cloreto e, para a área, ocorre entre hidrocarbonetos e óxidos.

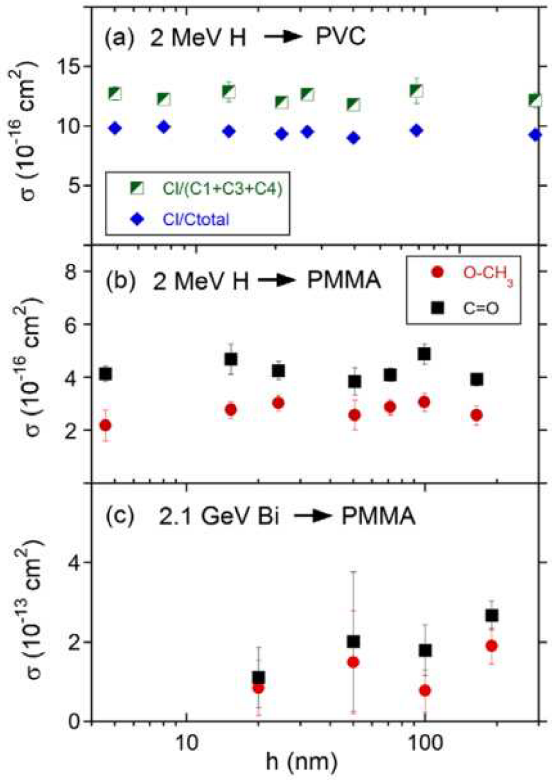
Tabela 8 – Diferença percentual entre os raios obtidos (contagem).

Substâncias Comparadas	**Diferença Percentual**
Hidrocarbonetos vs Óxidos	+33 %
Hidrocarbonetos vs Cloretos (Cl-35)	–25 %
Hidrocarbonetos vs Cloretos (Cl-37)	–16 %
Óxidos vs Cloretos (Cl-35)	–44 %
Óxidos vs Cloretos (Cl-37)	–37 %
Cloretos (Cl-35) vs Cloretos (Cl-37)	–13%

Tabela 9 – Diferença percentual entre os raios obtidos (área).

Substâncias Comparadas	**Diferença Percentual**
Hidrocarbonetos vs Óxidos	+39 %
Hidrocarbonetos vs Cloretos (Cl-35)	–1.7 %
Hidrocarbonetos vs Cloretos (Cl-37)	–11 %
Óxidos vs Cloretos (Cl-35)	–29 %
Óxidos vs Cloretos (Cl-37)	–36 %
Cloretos (Cl-35) vs Cloretos (Cl-37)	–10 %

Figura 56 – Seções transversais de danos químicos em filmes de PVC e PMMA de diferentes espessuras *h*.



Fonte: Extraído de THOMAZ *et al.* (2018)

Além disso, é possível comparar os resultados obtidos pelo programa desenvolvido neste trabalho, com os resultados reportados na literatura (THOMAZ *et al.*, 2018), no qual a seção de choque para amostras de PMMA e PVC foram calculadas a partir de dados obtidos com a aplicação da técnica XPS. Esta técnica determina a composição química da superfície do material, bem como informações sobre o estado de oxidação dos elementos presentes através da análise da energia cinética dos fotoelétrons. A Figura 56 reproduz os resultados obtidos para a seção de choque.

É interessante notar que, no caso das amostras de PMMA – irradiadas com feixes de Hidrogênio com energia de 2 MeV – a seção de choque obtida varia entre 2×10^{-16} cm^2 e 6×10^{-16} cm^2 (THOMAZ *et al.*, 2018). O aumento da deposição de energia na superfície através do aumento da massa e energia do íon incidente ocasiona um aumento significativo na seção de choque observada. Entretanto, estas seções de choque ainda estão uma ordem de magnitude abaixo dos valores encontrados no presente trabalho. Isso indica que o polímero PMMA, dentro dos limites de detecção da técnica, permanece inerte quimicamente e a ejeção de moléculas está ligada mais provavelmente a um processo físico.

Em contraste, as amostras de PVC – também irradiadas com feixes de Hidrogênio com energia de 2 MeV – a seção de choque obtida varia entre 10×10^{-16} cm^2 e 15×10^{-16} cm^2 (THOMAZ *et al.*, 2018). Neste caso, o presente trabalho obteve uma média na mesma ordem de grandeza desse valor. Isso é indicativo de que processos químicos podem ter um papel relevante na ejeção de material.

As comparações dos raios obtidos no presente trabalho com as dimensões das crateras observadas na superfície de polímeros irradiados com íons pesados no regime eletrônico não relativístico com técnicas SFM (PAPALÉO *et al.*, 2006; THOMAZ *et al.*, 2023) também não estão de acordo. Neste caso, entretanto, os raios obtidos no presente trabalho são uma ordem de grandeza inferiores aos raios reportados na literatura. Isso sugere que a eficiência de ejeção de íon secundários é baixa ($\sim$1 %) em relação aos secundários neutros.

É claro que uma análise mais aprofundada levando em consideração as diferentes deposições de energia de cada feixe precisa ser realizada para averiguar estas hipóteses.

5.6 PERSPECTIVAS FUTURAS

A função utilizada para o ajuste dos dados de contagem e área por tempo de exposição implementada no programa é uma função exponencial simples, descrita conforme a Equação (14). Apesar de utilizada como função base para o ajuste, nota-se que ela não oferece o melhor ajuste para os dados obtidos, conforme pode-se observar pelos gráficos em escala logarítmica, apresentados nas Figura 57 e Figura 58. Nesse contexto, é nítido a necessidade de ajustar outras funções.

Figura 57 – Gráfico em escala logarítmica para o pico de emissão referente ao tempo-de-voo de 31,5755 μs em função do tempo de exposição para o experimento **PMMA_6_MeV_Cu** (contagem).

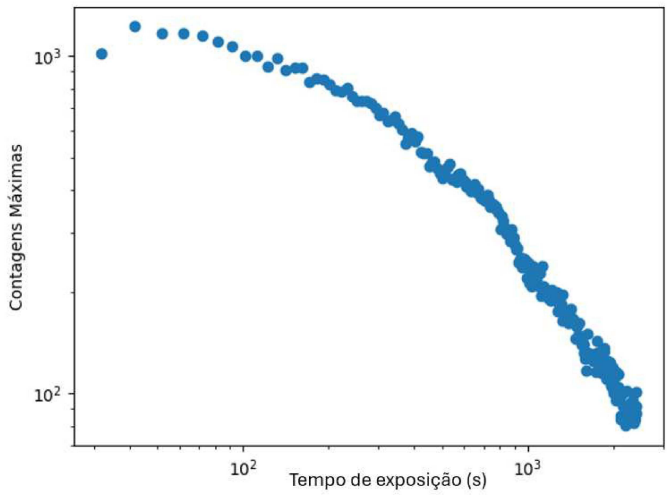
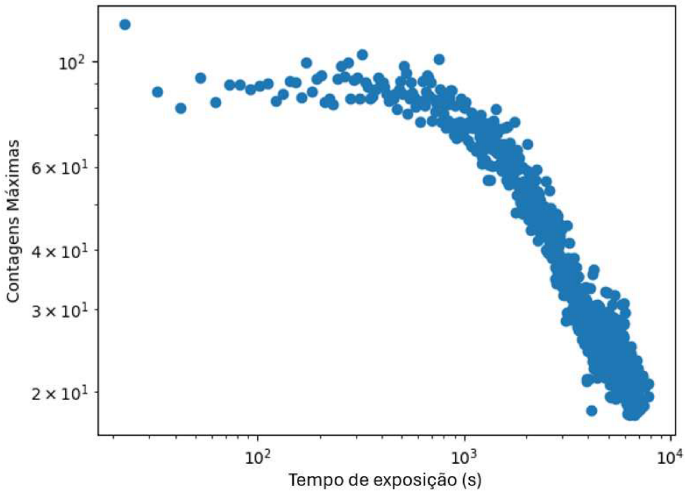


Figura 58 – Gráfico em escala logarítmica para o pico de emissão referente ao tempo-de-voo de 22,5670 μs em função do tempo de exposição para o experimento **PMMA_6_MeV_Cu** (área).



O programa é capaz de implementar facilmente novas funções para o ajuste, escrevendo-as na classe FUNCTION. Para um próximo desenvolvimento do programa, as funções descritas por Tee *et al.* (2021) serão implementadas.

6 CONCLUSÕES

Este trabalho desenvolveu um programa capaz de avaliar a degradação causada por feixes de íons no regime eletrônico de deposição como consequência do processo de desbastamento para três tipos diferentes de polímeros – PS, PMMA e PVC – através da análise de espectros gerados em experimentos de espectrometria de massa por íons secundários em medidas do tempo-de-voo.

Foram análisados um total de 7942 espectros divididos em dez experimentos compostos por diferentes amostras irradiadas por feixes continuos de íons primários de Cloro e Cobre com energias entre 6,0 e 12,0 MeV.

Para a realização desta análise, o programa desenvolvido em linguagem computacional PYTHON é responsável por: (i) identificar picos de emissão de íons secundários em todos os espectros; (ii) comparar estes picos com picos de referência; (iii) ajustar uma função exponencial à curva gerada por esta comparação; (iv) calcular a seção de choque para danos por irradiação e seus respectivos raios; e (v) por fim, identificar possíveis moléculas geradoras destes picos nos diferentes grupos moleculares – hidrocarbonetos, óxidos e cloretos.

Através dos dados de seção de choque e raios obtidos pelo programa observou-se para o grupo dos hidrocarbonetos uma média de $1{,}13 \times 10^{-14}$ cm^2 para sua seção de choque e 0,60 nm para o raio, obtidos a partir da área dos picos de emissão. Já utilizando os valores obtidos a partir da contagem máxima de cada pico estes valores reduzem em 33% para a seção de choque e 17% para o raio.

Também foi possível observar pelos valores de média para a seção de choque – $2{,}10 \times 10^{-14}$ cm^2 – que o grupo dos óxidos aparenta ser o menos resistente à incidência de radiação, sendo o que mais sofre o processo degradação. Seu valor médio para o tamanho do raio da seção de choque é de 0,82 nm. Essas médias reduzem 53% e 23% quando observadas para seção de choque e raio obtidos com os cálculos para a contagem máxima dos picos de emissão.

Já os cloretos apresentaram a menor seção de choque quando calculadas a partir da área dos picos de emissão, sendo seu valor médio de $1{,}12 \times 10^{-14}$ cm^2 para o isótopo 35 do Cloro e $0{,}88 \times 10^{-14}$ cm^2 para o isótopo 37. Entretanto, também foram o grupo com maior diferença entre as seções de choque obtidos pela área e pela contagem máxima dos picos de emissão, reduzindo em 145% para o isótopo 35 do Cloro e 100% para o isótopo 37.

Ao comparar os resultados de seção de choque encontrados pelo programa para este estudo de caso com trabalhos que aplicam as técnicas de XPS e SFM, pode-se notar que as seções de choque calculadas neste trabalho são diferentes. Na comparação com a técnica XPS, o polímero PMMA apresenta seções de choque uma ordem de magnitude acima dos valores reportados na literatura, enquanto o polímero

PVC apresenta valores da mesma ordem de grandeza. É possível que efeitos químicos sejam importantes nesse caso. Para a técnica SFM, o presente trabalho obteve raios uma ordem de grandeza inferiores, indicando que a maior parte do material ejetado encontra-se num estado de carga eletricamente neutro. Essas hipóteses necessitam ser confirmadas através de uma análise aprofundada das diferentes deposições de energia em cada feixe.

Como perspectiva futura espera-se a integração cada vez maior da automação em uma variedade de experimentos – buscando avaliar uma quantidade maior de espectos afim de viabilizar a análise de experimentos com nível de complexidade crescente –, bem como uma análise mais aprofundada da fragmentação de moléculas – buscando investigar os padrões e mecanismos de fragmentação em maior detalhe. Para tal, será necessário implementar funções para realização de ajustes mais refinados.

REFERÊNCIAS

ABU HASSAN SHAARI, H.; RAMLI, M.M.; MOHTAR, M.N.; RAHMAN, N. Abdul; AHMAD, A. Synthesis and conductivity studies of poly (methyl methacrylate) (PMMA) by co-polymerization and blending with polyaniline (PANi). **Polymers**, v. 13, p. 1939, 2021.

ALENCAR, I.; ONZI, G. S.; BATTÚ, L.; D'AVILA, B. N.; BOUFLEUR, L. A.; PAPALÉO, R. M.; GRANDE, P. L.; AMARAL, L.; DIAS, J. F. Operating a dual-stage mirror ToF-MeV-SIMS instrument with continuous sources of primary ions: Optimization and possibilities. [*S.l.*], 2024.

ALI, U.; ABD KARIM, K. J. Bt.; BUANG, N. A. A Review of the properties and applications of poly (methyl methacrylate) (PMMA). **Polymer Reviews**, v. 55, p. 678–705, 2015.

BARKUSKY, F.; BAYER, A.; PETH, C.; MANN, K. Direct structuring of solids by EUV radiation from a table-top laser produced plasma source. **Proceedings of SPIE - The International Society for Optical Engineering**, v. 7361, 2009.

BATTÚ, L. **Online ToF-SIMS: Novas implementações para análise de degradação de materiais**. 2019. Trabalho de Conclusão de Curso para Bacharelado em Engenharia Física – Universidade Federal do Rio Grande do Sul.

BEHRISCH, R.; ECKSTEIN, W. Introduction and Overview. *In*: SPUTTERING by particle bombardment: Experiments and computer calculations from threshold to MeV energies. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007. p. 1–20. ISBN 978-3-540-44502-9.

CHANDA, M.; ROY, S. **Plastics Fundamentals, Properties, and Testing**. [*S.l.*: *s.n.*], jul. 2008. p. 1–253. ISBN 9781420080612.

DHRUV, A. J.; PATEL, R.; DOSHI, N. Python: The most advanced programming language for computer science applications. **Proceedings of the International Conference on Culture Heritage, Education, Sustainable Tourism, and Innovation Technologies**, p. 292–299, 2020.

FINDLAY, C. **Development of FTIR tomography for thermal-source imaging of 3D biochemical distributions in micro-samples of cells and fibers**. Mai. 2018. Tese (Doutorado).

EL-GHARBAWY, A. Poly vinyl chloride additives and applications – A review. **Journal of Risk Analysis and Crisis Response**, v. 12, p. 143–151, 2022.

GREGOR-SVETEC, D. Chapter 14 - Polymers in printing filaments. *In*: IZDEBSKA-PODSIADLY, J. (Ed.). **Polymers for 3D Printing**. [*S.l.*]: William Andrew Publishing, 2022. (Plastics Design Library). p. 155–269.

HASHIM, A.; ABBAS, B. Recent Review on Poly-methyl methacrylate (PMMA)-Polystyrene (PS) Blend Doped with Nanoparticles For Modern Applications. **Research Journal of Agriculture and Biological Sciences**, v. 14, p. 6–12, 2019.

LÓPEZ FERNÁNDEZ, F. Secondary Ion Mass Spectrometry (SIMS): principles and applications. *In*: JOSÉ RAMÓN SEOANE, Xavier Llovet (Ed.). **Handbook of instrumental techniques for materials, chemical and biosciences research, Centres Científics i Tecnològics.** [*S.l.*]: Universitat de Barcelona, 2012. cap. 1.10, p. 1–12.

MACFARLANE, R. D.; TORGERSON, D. F. Californium-252 Plasma Desorption Mass Spectroscopy. **Science**, v. 191, p. 920–925, 1976.

MAHER, S.; JJUNJU, F. P. M.; TAYLOR, S. Colloquium: 100 years of mass spectrometry: Perspectives and future trends. **Reviews of Modern Physics**, v. 87, p. 113–135, 2015.

MCKINNEY, Wes. **Pandas Documentation**. [*S.l.*: *s.n.*]. `https://pandas.pydata.org/docs/`. Online; accessado em 10 de março 2024.

MONTENEGRO, R. S. P.; SERFATY, M. E. Aspectos gerais do poliestireno. **BNDES Setorial**, v. 16, p. 123–136, 2002.

NORDLUND, K. Historical review of computer simulation of radiation effects in materials. **Journal of Nuclear Materials**, v. 520, p. 273–295, 2019.

NZERUE-KENNETH, P.E.; ONU, F.U.; DENIS, A.U.; IGWE, J.S.; OGBU, N.H. Detailed Study of the Object-Oriented Programming (OOP) Features in Python. **British Journal of Computer, Networking and Information Technology**, v. 6, p. 83–93, 2023.

PAPALÉO, R.M.; FARENZENA, L.S.; DE ARAÚJO, M.A.; LIVI, R.P. Surface tracks in polymers induced by MeV heavy-ion impacts. **Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms**, v. 151, p. 135–139, 1999.

PAPALÉO, R.M.; HASENKAMP, W.; BARBOSA, L.G.; LEAL, R. Relaxation of craters produced by ion bombardment on PMMA as a function of temperature. **Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms**, v. 242, p. 190–193, 2006.

PYTHON SOFTWARE FOUNDATION. **Python 3 Documentation - Data Structures**. [*S.l.*: *s.n.*]. https://docs.python.org/3/tutorial/datastructures.html. Online; accessado em 10 de março de 2024.

RANJAN, M.; BAROT, K.; KHAIRNAR, V.; RAWAL, V.; PIMPALGAONKAR, A.; SAXENA, S.; SATTAR, A. Python: Empowering data science applications and research. **Journal of Operating Systems Development & Trends**, v. 10, p. 27–33, 2023.

RODRIGUES, T. T. **Polímeros nas indústria de embalagens**. 2018. Monografia para Trabalho de Conclusão de Curso em Engenharia Química – Universidade Federal de Uberlândia.

SCIPY CONTRIBUTORS. **SciPy – find_peaks documentation**. [*S.l.*: *s.n.*]. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.find_peaks.html`. Online; accessado em 10 de março de 2024.

SPEIGHT, J. Monomers, Polymers, and Plastics. *In*: [*S.l.*: *s.n.*], 2011. p. 499–537. ISBN 9780750686327.

SRINATH, K.R. Python—The Fastest Growing Programming Language. **International Research Journal of Engineering and Technology**, v. 4, p. 354–357, 2017.

TEE, B.P.E.; VOS, M.; TROMBINI, H.; SELAU, F.F.; GRANDE, P.L.; THOMAZ, R. The influence of radiation damage on electrons and ion scattering measurements from PVC films. **Radiation Physics and Chemistry**, v. 179, p. 109173, 2021.

THOMAZ, R.; LIMA, N. W.; TEIXEIRA, D.; GUTIERRES, L. I.; ALENCAR, I.; TRAUTMANN, C.; GRANDE, P. L.; PAPALÉO, R. M. Ion tracks in ultrathin polymer films: The role of the substrate. **Current Applied Physics**, v. 32, p. 91–97, 2021.

THOMAZ, R.; LOUETTE, P.; HOFF, G.; MÜLLER, S.; PIREAUX, J.J.; TRAUTMANN, C.; PAPALÉO, R. M. Bond-breaking efficiency of high-energy ions in ultrathin polymer films. **Physical Review Letters**, v. 121, p. 066101, 2018.

THOMAZ, R.; NGONO-RAVACHE, Y.; SEVERIN, D.; TRAUTMANN, C.; PAPALÉO, R. M. Thinning of poly (methyl methacrylate) and poly (vinyl chloride) thin films induced by high-energy ions of different stopping powers. **Polymers**, v. 15, p. 4471, 2023.

URBASSEK, H. M. Molecular-dynamics simulation of sputtering. **Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms**, v. 122, p. 427–441, 1997.

APÊNDICE A – CLASSE CALC CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```
1  import numpy as np
2  from statistics import mean, stdev
3  import matplotlib.pyplot as plt
4  import scipy.signal
5  import os
6  import sys
7  import pandas as pd
8  import math
9
10 sys.path.append("../")
11
12 from graphics.graphics import Graphics
13
14
15 class Calc:
16     def __init__(self):
17         self.graphic = Graphics()
18
19     def find_data_multiple_peaks(self, tofs, count, peak_path, plot_path, spec_number,
       ↪  spec_name, save_files):
20         h1 = np.mean(count) + np.std(count)
21         h1 = 0.025 * np.max(count)
22
23         #(1)(2)(3)(4)(6)(9)(10)peaks, properties = scipy.signal.find_peaks(count,
           ↪  height = h1, prominence = 10, width=0.01, distance = 400)
24         peaks, properties = scipy.signal.find_peaks(count, height = h1, prominence =
           ↪  20, width=0.01, distance = 400)
25         #(7)peaks, properties = scipy.signal.find_peaks(count, height = h1, prominence
           ↪  = 8, width=0.01, distance = 400)
26         #(8)peaks, properties = scipy.signal.find_peaks(count, height = h1, prominence
           ↪  = 2.5, width=0.01, distance = 400)
27         x1_max = tofs[peaks] #Lista da posição dos picos
28         y1_max = properties['peak_heights'] #Lista das alturas dos picos
29         picos = zip(x1_max, y1_max)
30
31         peaks_file_path = os.path.join(peak_path,
           ↪  f'{spec_number}_{spec_name}_peaks.txt')
32
33         if save_files:
34             with open(peaks_file_path, 'w') as my_file:
35                 my_file.write("Peak_ToF\tPeak_Count\n")
36                 for x, y, in picos:
37                     my_file.write("{0}\t{1}\n".format(x, y))
38             print('File created in:', peaks_file_path)
```

```
39
40              ######## Plot ##########
41              self.graphic.plot_peak(tofs, count, x1_max, y1_max, h1, plot_path,
                ↪  spec_number, spec_name)
42
43          return peaks_file_path
44
45      def analyze_statistics_bkgd(self, data_tof, data_count, peak_tof):
46          # Região anterior ao pico para análise estatistica
47          #x_tof_before_peak = []
48          y_count_before_peak = []
49          for i in range(len(peak_tof)):
50              #x1 = data_tof[np.where((data_tof > (peak_tof[i] - 0.1)) & (data_tof <
                ↪  (peak_tof[i] - 0.05)))]
51              y1 = data_count[np.where((data_tof > (peak_tof[i] - 0.1)) & (data_tof <
                ↪  (peak_tof[i] - 0.05)))]
52
53              #x_tof_before_peak.append(x1)
54              y_count_before_peak.append(y1)
55
56          # Análise estatistica
57          mean_bkgd = []
58          stdev_bkgd = []
59          for i in range(len(peak_tof)):
60              if y_count_before_peak[i].size == 0:
61                  media = 0
62                  desvio = 0
63              else:
64                  media = mean(y_count_before_peak[i])
65                  desvio = stdev(y_count_before_peak[i])
66
67
68              mean_bkgd.append(media)
69              stdev_bkgd.append(desvio)
70
71          return mean_bkgd, stdev_bkgd
72
73      # Valor fixo para limite inferior e superior do pico usando media diferente
74      def count_inside_region_of_interest_minus_mean_bkgd(self, data_tof, data_count,
        ↪  peak_tof, superior_limit, inferior_limit, mean_bkgd):
75          peak_tof_region = []
76          peak_count_region = []
77          for i in range (0, len(peak_tof), 1):
78              regiao_interesse_tof = data_tof[np.where((data_tof > (peak_tof[i] -
                ↪  inferior_limit)) & (data_tof < (peak_tof[i] + superior_limit)))]
79              regiao_interesse_contagem = data_count[np.where((data_tof > (peak_tof[i] -
                ↪  inferior_limit)) & (data_tof < (peak_tof[i] + superior_limit)))]
```

```
80
81              peak_tof_region.append(regiao_interesse_tof)
82              peak_count_region.append(regiao_interesse_contagem)
83
84          new_peak_count_region= []
85          for i in range (0, len(peak_tof), 1):
86              nova_regiao_interesse_contagem = peak_count_region[i] - mean_bkgd[i]
87              new_peak_count_region.append(nova_regiao_interesse_contagem)
88
89          return peak_tof_region, new_peak_count_region
90
91      def new_max_tof_count_inside_peak_region(self, peak_tof, new_peak_count, peak_path,
        ↪  spec_number, spec_name):
92          maximo_contagem = []
93          for i in range (0, len(peak_tof), 1):
94              max_contagem = new_peak_count[i].max()
95              maximo_contagem.append(max_contagem)
96
97          index_tof = []
98          for i in range (0, len(peak_tof), 1):
99              index= np.where(new_peak_count[i].round(9) == maximo_contagem[i].round(9))
100             index_tof.append(index[0][0])
101
102         maximo_tof = []
103         for i in range (0, len(peak_tof), 1):
104             max_tof = peak_tof[i][index_tof[i]]
105             maximo_tof.append(max_tof)
106
107         picos = zip(maximo_tof, maximo_contagem)
108
109         peaks_file_path = os.path.join(peak_path,
            ↪  f'{spec_number}_{spec_name}_peaks_new.txt')
110         with open(peaks_file_path, 'w') as my_file:
111             my_file.write("Peak_ToF\tPeak_Count\n")
112             for x, y in picos:
113                 my_file.write("{0}\t{1}\n".format(x, y))
114         print('File created in:', peaks_file_path)
115
116         return maximo_tof, maximo_contagem
117
118     def calc_area_for_peaks(self, peaks_tof_region, new_peak_count_region):
119         area = []
120         for i in range (0, len(peaks_tof_region), 1):
121             contagem_soma = sum(new_peak_count_region[i])
122             area.append(contagem_soma)
123
124         return area
```

```
125
126     def is_within_range(base_x, x):
127         return abs(base_x - x) <= 0.05
128
129     def cal_ejected_part_mass(self, row):
130         t = row['row_index']
131         params = eval(row['params'])
132         b = params[1]
133         tof = 0.1445
134         C = 2.5904
135         m = ((tof - t) / (C))**2
136         return m
137
138     def compare_mass(self, df1, df2, base_dir, area):
139         if area:
140             extracted_files = os.listdir(base_dir)
141             csv_files = [file for file in extracted_files if
                ↪  file.endswith('area.csv')]
142             filepath = os.path.join(base_dir, csv_files[0])
143         else:
144             extracted_files = os.listdir(base_dir)
145             csv_files = [file for file in extracted_files if
                ↪  file.endswith('results.csv')]
146             filepath = os.path.join(base_dir, csv_files[0])
147
148         dfs = []
149         ldelta_M = [0.01, 0.05, 0.10]
150
151         if 'PVC' in filepath:
152             for k in ldelta_M:
153                 rows_hidro = []
154                 rows_cl_35  = []
155                 rows_cl_37 = []
156                 for i in range(len(df1['M'])):
157                     massa_tof = df1['M'][i]
158                     for j in range(len(df2['mass'])):
159                         massa_tabela = df2['mass'][j]
160                         if abs(massa_tof - massa_tabela) <= k:
161                             # Append row to the list
162                             rows_hidro.append({'C': df2['C'][j],
163                                                'H': df2['H'][j],
164                                                'diferenca': k,
165                                                'massa_calculada': massa_tof,
166                                                'massa_tabelada': massa_tabela,
167
                                                ↪  'tempo-de-voo_calculado':df1['row_index'][i],
168                                                'tempo-de-voo_tabelado':df2['tof'][j]})
```

```
169
170                 for i in range(len(df1['M'])):
171                     massa_tof = df1['M'][i]
172                     for j in range(len(df2['mass_35'])):
173                         massa_tabela = df2['mass_35'][j]
174                         if abs(massa_tof - massa_tabela) <= k:
175                             # Append row to the list
176                             rows_cl_35.append({'C': df2['C'][j],
177                                                'H': df2['H'][j],
178                                                'Cl_35':df2['Cl_35'][j],
179                                                'diferenca': k,
180                                                'massa_calculada': massa_tof,
181                                                'massa_tabelada': massa_tabela,
182
                                                   ↪  'tempo-de-voo_calculado':df1['row_index'][i],
183                                                'tempo-de-voo_tabelado':df2['tof_35'][j]})
184                 for i in range(len(df1['M'])):
185                     massa_tof = df1['M'][i]
186                     for j in range(len(df2['mass_37'])):
187                         massa_tabela = df2['mass_37'][j]
188                         if abs(massa_tof - massa_tabela) <= k:
189                             # Append row to the list
190                             rows_cl_37.append({'C': df2['C'][j],
191                                                'H': df2['H'][j],
192                                                'Cl_37':df2['Cl_37'][j],
193                                                'diferenca': k,
194                                                'massa_calculada': massa_tof,
195                                                'massa_tabelada': massa_tabela,
196
                                                   ↪  'tempo-de-voo_calculado':df1['row_index'][i],
197                                                'tempo-de-voo_tabelado':df2['tof_35'][j]})
198
199             if rows_hidro:
200                 df_hidro = pd.DataFrame(rows_hidro)
201                 dfs.append(df_hidro)
202             if rows_cl_35:
203                 df_35 = pd.DataFrame(rows_cl_35)
204                 dfs.append(df_35)
205             if rows_cl_37:
206                 df_37 = pd.DataFrame(rows_cl_37)
207                 dfs.append(df_37)
208
209         elif 'PMMA' in filepath:
210             for k in ldelta_M:
211                 rows_hidro = []
212                 rows_o  = []
213                 for i in range(len(df1['M'])):
```

```
214                     massa_tof = df1['M'][i]
215                     for j in range(len(df2['mass'])):
216                         massa_tabela = df2['mass'][j]
217                         if abs(massa_tof - massa_tabela) <= k:
218                             # Append row to the list
219                             rows_hidro.append({'C': df2['C'][j],
220                                                'H': df2['H'][j],
221                                                'diferenca': k,
222                                                'massa_calculada': massa_tof,
223                                                'massa_tabelada': massa_tabela,
224 
                                                ↪  'tempo-de-voo_calculado':df1['row_index'][i],
225                                                'tempo-de-voo_tabelado':df2['tof'][j]})
226 
227                 for i in range(len(df1['M'])):
228                     massa_tof = df1['M'][i]
229                     for j in range(len(df2['mass_o'])):
230                         massa_tabela = df2['mass_o'][j]
231                         if abs(massa_tof - massa_tabela) <= k:
232                             # Append row to the list
233                             rows_o.append({'C': df2['C'][j],
234                                            'H': df2['H'][j],
235                                            'O':df2['O'][j],
236                                            'diferenca': k,
237                                            'massa_calculada': massa_tof,
238                                            'massa_tabelada': massa_tabela,
239 
                                            ↪  'tempo-de-voo_calculado':df1['row_index'][i],
240                                            'tempo-de-voo_tabelado':df2['tof_o'][j]})
241                 # Create a DataFrame from the list of rows
242             if rows_hidro:
243                 df_hidro = pd.DataFrame(rows_hidro)
244                 dfs.append(df_hidro)
245             if rows_o:
246                 df_o = pd.DataFrame(rows_o)
247                 dfs.append(df_o)
248 
249         elif "PS" in filepath:
250             for k in ldelta_M:
251                 rows_hidro = []
252                 rows_o  = []
253                 for i in range(len(df1['M'])):
254                     massa_tof = df1['M'][i]
255                     for j in range(len(df2['mass'])):
256                         massa_tabela = df2['mass'][j]
257                         if abs(massa_tof - massa_tabela) <= k:
258                             # Append row to the list
```

```
259                             rows_hidro.append({'C': df2['C'][j],
260                                                'H': df2['H'][j],
261                                                'diferenca': k,
262                                                'massa_calculada': massa_tof,
263                                                'massa_tabelada': massa_tabela,
264
                                                ↪  'tempo-de-voo_calculado':df1['row_index'][i],
265                                                'tempo-de-voo_tabelado':df2['tof'][j]})
266             if rows_hidro:
267                 df_hidro = pd.DataFrame(rows_hidro)
268                 dfs.append(df_hidro)
269
270
271         return dfs
272
273         # Save the DataFrame to a CSV file
274         #result_df.to_csv(output_file, index=False)
275
276     def cal_cross_section(self, row, fluence):
277         params = eval(row['params'])
278         b = params[1]
279         cross_section = - (b / fluence)
280         return cross_section
281
282     def cal_radius(self, df):
283         df['radius'] = np.sqrt(df['cross_section'] / np.pi)
284         return dfsection'] / np.pi)
285         return df
```

APÊNDICE B – CLASSE FILES CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import numpy as np
import glob
import os
import pandas as pd
import matplotlib.pyplot as plt
import scipy.signal

class Files:
    def __init__(self):
        pass

    def list_txt_files(self, directory_path):
        all_files = os.listdir(directory_path)
        txt_files = sorted([f for f in all_files if f.endswith('.txt')])
        return txt_files

    def construct_path_files(self, directory_path, file):
        filepath = os.path.join(directory_path, file)
        return filepath

    def read_files(self, filepath, column):
        data = np.genfromtxt(filepath)
        return data[1:, column]

    def read_files_2(self, filepath, column):
        try:
            data = np.genfromtxt(filepath)  # Assuming CSV, you can change delimiter if needed
            if data.size == 0:  # Check if the data is empty
                print("File is empty!")
                return None
            return data[1:, column]
        except Exception as e:
            print(f"An error occurred: {e}")
            return None

    def save_to_txt_multiples(self, file_path, peak_tof, max_tof_per_peak, max_count_per_peak, area_under_peaks, error_area, error_count):
        with open(file_path, 'w') as file:

            # Writing column names
            file.write("Max_ToF\tMax_Count\tArea\tError_Area\tError_Count\n")

            # Writing data
```

```
43              for i in range(len(peak_tof)):
44                  file.write(f"{max_tof_per_peak[i]}\t")
45                  file.write(f"{max_count_per_peak[i]}\t")
46                  file.write(f"{area_under_peaks[i]}\t")
47                  file.write(f"{error_area[i]}\t")
48                  file.write(f"{error_count[i]}\n")
49                  #file.write(f"{error_tof[i]}\n")
50
51      def csv_to_df(self, path):
52          df = pd.read_csv(path, delimiter = '\t')
53          df.set_index('Unnamed: 0', inplace=True)
54          return df
55
56      def create_paths(self, base_path):
57          if not os.path.exists(base_path):
58              os.makedirs(base_path)
59
60          peaks_path = os.path.join(base_path, "peaks")
61          if not os.path.exists(peaks_path):
62              os.makedirs(peaks_path)
63
64          plot_path = os.path.join(base_path, 'plots')
65          if not os.path.exists(plot_path):
66              os.makedirs(plot_path)
67
68          adjusted_path = os.path.join(base_path, "adjusted_functions")
69          if not os.path.exists(adjusted_path):
70              os.makedirs(adjusted_path)
71          return peaks_path, plot_path, adjusted_path
72
73
74      def read_csv(self, base_dir, area):
75          if area:
76              extracted_files = os.listdir(base_dir)
77              csv_files = [file for file in extracted_files if
                ↪  file.endswith('results_area.csv')]
78              filepath = os.path.join(base_dir, csv_files[0])
79          else:
80              extracted_files = os.listdir(base_dir)
81              csv_files = [file for file in extracted_files if
                ↪  file.endswith('results.csv')]
82              filepath = os.path.join(base_dir, csv_files[0])
83
84          dataframe = pd.read_csv(filepath, header=0 ,delimiter = ',')
85          return dataframe
86
87
```

```
88      def read_csv_multiple_sheets(self, base_dir, area):
89          if area:
90              extracted_files = os.listdir(base_dir)
91              csv_files = [file for file in extracted_files if
                ↪  file.endswith('area.csv')]
92              filepath = os.path.join(base_dir, csv_files[0])
93          else:
94              extracted_files = os.listdir(base_dir)
95              csv_files = [file for file in extracted_files if
                ↪  file.endswith('results.csv')]
96              filepath = os.path.join(base_dir, csv_files[0])
97
98          table_path =
            ↪  '/home/ubuntu/Documents/Dissertation/Polimeros/specs/mass_compare.xlsx'
99          if 'PS' in filepath:
100             # Read the CSV file with multiple tabs
101             xls = pd.ExcelFile(table_path, engine='openpyxl')
102             # Get the name of the second tab
103             df1 = pd.read_excel(table_path, sheet_name=0, header = 0)
104             df_total = df1
105         elif 'PMMA' in filepath:
106             # Read the CSV file with multiple tabs
107             xls = pd.ExcelFile(table_path)
108             # Read the second tab into a DataFrame
109             df1 = pd.read_excel(table_path, sheet_name=0, header = 0)
110             df2 = pd.read_excel(table_path, sheet_name=1, header = 0)
111             df_total = pd.concat([df1, df2], ignore_index=True)
112             df_total.fillna(0, inplace=True)
113         elif 'PVC' in filepath:
114             # Read the CSV file with multiple tabs
115             xls = pd.ExcelFile(table_path)
116             # Read the second tab into a DataFrame
117             df1 = pd.read_excel(table_path, sheet_name=0, header = 0)
118             df2 = pd.read_excel(table_path, sheet_name=2, header = 0)
119             df_total = pd.concat([df1, df2], ignore_index=True)
120             df_total.fillna(0, inplace=True)
121         else:
122             print("Não encontrado nenhuma aba para comparação")
123
124         return df_total, area
125
126     def df_to_csv(self, dataframe, output_file):
127         dataframe.to_csv(output_file, index=False)
128
129     def read_fluence_table(self):
130         table_path =
            ↪  "/home/ubuntu/Documents/Dissertation/Polimeros/specs/sims-flux.txt"
```

```
131         df = pd.read_csv(table_path , delimiter = '\t', encoding='utf-8')
132         return df
133 
134     def save_dataframes_as_txt(self, base_dir, area, dfs):
135         if area:
136             filepath = os.path.join(base_dir, "area_mass_compared")
137         else:
138             filepath = os.path.join(base_dir, "mass_compared")
139 
140         if 'PVC' in filepath:
141             list_name = ["hidrocarboneto", "cloreto_35", "cloreto_37"]
142             for i, df in enumerate(dfs):
143                 file_name = filepath + "_" + list_name[i] + ".txt"
144                 # Save the DataFrame as a .txt file
145                 df.to_csv(file_name, sep='\t', index=False)
146         if 'PMMA' in filepath:
147             list_name = ["hidrocarboneto", "oxido"]
148             for i, df in enumerate(dfs):
149                 file_name = filepath + "_" + list_name[i] + ".txt"
150                 # Save the DataFrame as a .txt file
151                 df.to_csv(file_name, sep='\t', index=False)
152         if "PS" in filepath:
153             list_name = ['hidrocarboneto']
154             for i, df in enumerate(dfs):
155                 file_name = filepath + "_" + list_name[i] + ".txt"
156                 # Save the DataFrame as a .txt file
157                 df.to_csv(file_name, sep='\t', index=False)
158 
159 
160 
161     def read_csv_2(self, base_dir, area):
162         if area:
163             extracted_files = os.listdir(base_dir)
164             csv_files = [file for file in extracted_files if
                ↪  file.endswith('Cross_and_radius_area.csv')]
165             filepath = os.path.join(base_dir, csv_files[0])
166         else:
167             extracted_files = os.listdir(base_dir)
168             csv_files = [file for file in extracted_files if
                ↪  file.endswith('Cross_and_radius.csv')]
169             filepath = os.path.join(base_dir, csv_files[0])
170 
171         dataframe = pd.read_csv(filepath, header=0 ,delimiter = ',')
172         return dataframe
173 
174     def read_masses_txt(self, base_dir, area):
175         df_total = []
```

```
176         if 'PS' in base_dir:
177             if area:
178                 filepath = os.path.join(base_dir,
                    ↪ "area_mass_compared_hidrocarboneto.txt")
179             else:
180                 filepath = os.path.join(base_dir, "mass_compared_hidrocarboneto.txt")
181             df1 = pd.read_csv(filepath, header = 0, delimiter= '\t')
182             df_total.append(df1)
183 
184         elif 'PMMA' in base_dir:
185             if area:
186                 filepath_hidro = os.path.join(base_dir,
                    ↪ "area_mass_compared_hidrocarboneto.txt")
187                 filepath_oxi = os.path.join(base_dir, "area_mass_compared_oxido.txt")
188             else:
189                 filepath_hidro = os.path.join(base_dir,
                    ↪ "mass_compared_hidrocarboneto.txt")
190                 filepath_oxi = os.path.join(base_dir, "mass_compared_oxido.txt")
191             df_hidro = pd.read_csv(filepath_hidro, header = 0, delimiter= '\t')
192             df_oxido = pd.read_csv(filepath_oxi, header = 0, delimiter= '\t')
193             # Read the CSV file with multiple tabs
194             df_total.append(df_hidro)
195             df_total.append(df_oxido)
196         elif 'PVC' in base_dir:
197             if area:
198                 filepath_hidro = os.path.join(base_dir,
                    ↪ "area_mass_compared_hidrocarboneto.txt")
199                 filepath_cl35 = os.path.join(base_dir,
                    ↪ "area_mass_compared_cloreto_35.txt")
200                 filepath_cl37 = os.path.join(base_dir,
                    ↪ "area_mass_compared_cloreto_37.txt")
201 
202             else:
203                 filepath_hidro = os.path.join(base_dir,
                    ↪ "mass_compared_hidrocarboneto.txt")
204                 filepath_cl35 = os.path.join(base_dir, "mass_compared_cloreto_35.txt")
205                 filepath_cl37 = os.path.join(base_dir, "mass_compared_cloreto_37.txt")
206 
207             df_hidro = pd.read_csv(filepath_hidro, header = 0, delimiter= '\t')
208             df_cl35 = pd.read_csv(filepath_cl35, header = 0, delimiter= '\t')
209             df_cl37 = pd.read_csv(filepath_cl37, header = 0, delimiter= '\t')
210 
211             # Read the CSV file with multiple tabs
212             df_total.append(df_hidro)
213             df_total.append(df_cl35)
214             df_total.append(df_cl37)
215         else:
```

```
216             print("Não encontrado nenhuma aba para comparação")
217
218         return df_total
219
220
221     def save_dataframes_as_txt_2(self, base_dir, area, dfs):
222         if area:
223             filepath = os.path.join(base_dir, "final_result_area")
224         else:
225             filepath = os.path.join(base_dir, "final_result")
226
227         if 'PVC' in filepath:
228             list_name = ["hidrocarboneto", "cloreto_35", "cloreto_37"]
229             for i, df in enumerate(dfs):
230                 file_name = filepath + "_" + list_name[i] + ".txt"
231                 # Save the DataFrame as a .txt file
232                 df.to_csv(file_name, sep='\t', index=False)
233         if 'PMMA' in filepath:
234             list_name = ["hidrocarboneto", "oxido"]
235             for i, df in enumerate(dfs):
236                 file_name = filepath + "_" + list_name[i] + ".txt"
237                 # Save the DataFrame as a .txt file
238                 df.to_csv(file_name, sep='\t', index=False)
239         if "PS" in filepath:
240             list_name = ['hidrocarboneto']
241             for i, df in enumerate(dfs):
242                 file_name = filepath + "_" + list_name[i] + ".txt"
243                 # Save the DataFrame as a .txt file
244                 df.to_csv(file_name, sep='\t', index=False)
```

APÊNDICE C – CLASSE ERRORS CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import numpy as np
from scipy import interpolate

class Errors:
    def __init__(self):
        pass

    def error_area(self, area):
        error_area = []
        for i in area:
            erro = np.sqrt(i)
            error_area.append(erro)

        return error_area

    def error_count(self, maximo_contagem):
        # Maior pico entre os picos
        error_contagem = []
        for i in maximo_contagem:
            erro = np.sqrt(i)
            error_contagem.append(erro)

        return error_contagem

    def error_tof(self, peaks_tof_region, new_peaks_count_region, max_tof_per_peak):
        # Interpolate each region of interest
        interpolated_functions = []
        for i in range(len(peaks_tof_region)):
            x = peaks_tof_region[i]
            y = new_peaks_count_region[i]
            f = interpolate.interp1d(x, y, bounds_error=False, fill_value=0)
            interpolated_functions.append(f)

        # Generate new x-values around each max_tof_per_peak
        x_new = []
        for i, max_tof in enumerate(max_tof_per_peak):
            x_min = max(max_tof - 0.01, min(peaks_tof_region[i]))
            x_max = min(max_tof, max(peaks_tof_region[i]))
            x_novo = np.arange(x_min, x_max, 0.00001)
            x_new.append(x_novo)

        # Compute interpolated y-values for each new x-values set
        interpolated_y_new = []
        for i in range(len(x_new)):
```

```
45              y_interpolated = interpolated_functions[i](x_new[i])
46              interpolated_y_new.append(y_interpolated)
47
48          # Find x-values for half-maximum y-values
49          half_max_x_values = []
50          # for i in range(len(interpolated_y_new)):
51          #     half_max = max(interpolated_y_new[i]) / 2
52          #     index_half_max_y = np.abs(interpolated_y_new[i] - half_max).argmin()
53          #     half_max_x_values.append(x_new[i][index_half_max_y])
54          for i in range(len(interpolated_y_new)):
55              if len(interpolated_y_new[i]) == 0:  # Check if sequence is empty
56                  half_max_x_values.append(None)  # or handle it in another suitable
                    ↪ manner
57              else:
58                  half_max = max(interpolated_y_new[i]) / 2
59                  index_half_max_y = np.abs(interpolated_y_new[i] - half_max).argmin()
60                  half_max_x_values.append(x_new[i][index_half_max_y])
61
62          # Compute difference between half_max_x_values and max_tof_per_peak
63          #difference = [max_t - half_max_x for max_t, half_max_x in
            ↪ zip(max_tof_per_peak, half_max_x_values)]
64          difference = [max_t - half_max_x for max_t, half_max_x in zip(max_tof_per_peak,
            ↪ half_max_x_values) if half_max_x is not None]
65
66
67          return half_max_x_values, difference
68
69
70      def error_fluence(self, fluence):
71          error_fluence = 0.1 * fluence
72          return error_fluence
73
74      def error_cross_section(self, row, fluence, error_fluence):
75          params = eval(row['params'])
76          b = - params[1]
77
78          if row['errors'] == '[inf, inf]':
79              sigma_f = np.NaN
80              return sigma_f
81          else:
82              params_errors = eval(row['errors'])
83              error_b = - params_errors[1]
84
85              # Calcular as derivadas parciais
86              df_db = 1 / fluence
87
88              # Calcular o erro propagado
```

```
89              sigma_f = np.sqrt((df_db**2)*(error_b)**2)
90
91              return sigma_f
92
93
94      def error_radius(self, row):
95          radius = row['radius']
96          error_cross = row['error_cross_section']
97          df_dcross = 1 / (2*np.sqrt(radius)*np.sqrt(np.pi))
98          error_radius = np.sqrt((df_dcross**2)*(error_cross**2))
99          return error_radius
```

APÊNDICE D – CLASSE UTILS CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import numpy as np
import glob
from statistics import mean, stdev
import matplotlib.pyplot as plt
import scipy.signal
import os
import re
import pandas as pd


class Utils:
    def __init__(self):
        pass

    # def extract_number(self, s):
    #     match = re.search(r'', s)
    #     return int(match.group(1)) if match else 1

    def extract_spec_number(self, filepath):
        match = re.search(r'Spec(\d+)', filepath)
        if match:
            return int(match.group(1))
        return None

    def extract_spec_name(self, filepath):
        match = re.search(r'-([\w]+-\d+s)-', filepath)

        if match:
            extracted_part = match.group(1)
            return extracted_part
        else:
            return "Pattern not found in the filename."

    def custom_sort(self, file_name):
        num = int(''.join(filter(str.isdigit, file_name)))
        return num

    def selecting_right_experiment_row(self, dataframe, adjusted_path):
        matches = dataframe['experiment'].apply(lambda exp: exp in adjusted_path)

        # Check if any match is found
        if matches.any():
            matching_fluence = dataframe.loc[matches, 'fluence'].iloc[0]
            matching_fluence = float(matching_fluence)
```

```
45            return matching_fluence
46        else:
47            print("No experiment value is present in the filename.")
48
49    def add_cross_section_and_radius(self, df_list, reference_df):
50        final_df = []
51        for df in df_list:
52
53            # Merge the current DataFrame with the reference DataFrame based on the
              ↪  common column
54            merged_df = pd.merge(df, reference_df, left_on='tempo-de-voo_calculado',
              ↪  right_on='row_index', how='inner')
55
56            # Add the cross_section and radius columns to the current DataFrame
57            df['cross_section'] = merged_df['cross_section']
58            df['error_cross_section'] = merged_df['error_cross_section']
59            df['radius'] = merged_df['radius']
60            df['error_radius'] = merged_df['error_radius']
61            df = df.dropna()
62            final_df.append(df)
63
64        return final_df
65
66    def create_formula_column(self, df):
67        formula_list = []
68
69        # Define a mapping of columns to their corresponding elements
70        element_mapping = {
71            'C': 'C',
72            'H': 'H',
73            'O': 'O',
74            'Cl_35': 'Cl_35',
75            'Cl_37': 'Cl_37'
76        }
77
78        # Create formula based on available columns
79        for index, row in df.iterrows():
80            formula = ''
81            for column_name, value in row.iteritems():
82                if column_name in element_mapping:
83                    formula += f"{element_mapping[column_name]}{int(value)}"
84            formula_list.append(formula)
85
86        df['Formula'] = formula_list
87        return df
88
89    def add_formula_column_to_dfs(self, list_of_dfs):
```

```
90          updated_dfs = []
91          for df in list_of_dfs:
92              updated_dfs.append(self.create_formula_column(df))
93          return updated_dfs
```

APÊNDICE E – CLASSE GRAPHICS CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import numpy as np
from statistics import mean, stdev
import matplotlib.pyplot as plt
import scipy.signal
import os
import pandas as pd
import ast
from scipy.optimize import curve_fit
import csv



from functions.functions import Functions

class Graphics:
    def __init__(self):
        self.function = Functions()

    def adjust_spec(self, peak_path, adjusted_path):
        path = os.path.join(peak_path, 'analised_specs.tsv')
        df = pd.read_csv(path, delimiter='\t')
        df.set_index('Unnamed: 0', inplace=True)

        df = df.applymap(lambda x: ast.literal_eval(x) if pd.notna(x) else x)

        for row_index, row in df.iterrows():
            x_values = []
            y_values = []
            for col_name, value in row.items():
                if not pd.isna(value):
                    max_tof_value = value.get('Max_ToF')
                    max_count = value.get('Max_Count')
                    col_index = df.columns.get_loc(col_name)
                    max_tof = max_tof_value + 10 * col_index
                    x_values.append(max_tof)
                    y_values.append(max_count)

            plt.scatter(x_values, y_values, label=f'Row {row_index}')
            plt.xlabel('Max_ToF')
            plt.ylabel('Max_Count')
            plt.title(f'Scatter Plot of Max_ToF vs Max_Count for Row {row_index}')
            plt.legend()

            save_path = f'{adjusted_path}/_scatter_plot_row{row_index}.png'
```

```
45
46              plt.savefig(save_path)
47              plt.close()
48
49      def adjust_spec(self, peak_path, adjusted_path, plot_area, plot_exponential=True,
        ↪  plot_modified_exponential=False):
50          print("------------------------")
51          print("Adjusting functions to peaks..... ")
52          path = os.path.join(peak_path, 'analised_specs.tsv')
53          df = pd.read_csv(path, delimiter='\t')
54          df.set_index('Unnamed: 0', inplace=True)
55
56          df = df.applymap(lambda x: ast.literal_eval(x) if pd.notna(x) else x)
57
58          results = []
59
60          for row_index, row in df.iterrows():
61              x_values = []
62              y_values = []
63
64              for col_name, value in row.items():
65                  if not pd.isna(value):
66                      if plot_area == True:
67                          area = value.get('Area')
68                          y_values.append(area)
69                      else:
70                          max_count = value.get('Max_Count')
71                          y_values.append(max_count)
72
73                      max_tof_value = value.get('Max_ToF')
74                      col_index = df.columns.get_loc(col_name)
75                      max_tof = max_tof_value + 10 * col_index
76                      x_values.append(max_tof)
77
78
79              params_exp = None
80              try:
81                  if plot_exponential:
82                      params_exp, errors_exp =
                        ↪  curve_fit(self.function.exponential_function, x_values,
                        ↪  y_values, p0=[1, 0])
83                      errors_exp = np.array([np.sqrt(errors_exp[0][0]),
                        ↪  np.sqrt(errors_exp[1][1])])
84                      # results.append({'row_index': row_index, 'function_type':
                        ↪  'Exponential', 'params': params_exp.tolist()})
```

```
85                  results.append({'row_index': row_index, 'function_type':
                    ↪  'Exponential', 'params': params_exp.tolist(), 'errors':
                    ↪  errors_exp.tolist()})
86              except Exception as e:
87                  print(f"Error fitting exponential function for row {row_index}: {e}")
88
89              params_mod_exp = None
90              try:
91                  if plot_modified_exponential:
92                      params_mod_exp, errors_mod_exp =
                        ↪  curve_fit(self.function.modified_exponential_function,
                        ↪  x_values, y_values, p0=[1, 0.1, 0.1], bounds=([0, 0, 0],
                        ↪  [np.inf, np.inf, np.inf]), maxfev=5000)
93                      errors_mod_exp = np.sqrt(np.diag(errors_mod_exp))
94                      results.append({'row_index': row_index, 'function_type': 'Modified
                        ↪  Exponential', 'params': params_mod_exp.tolist(), 'errors':
                        ↪  errors_mod_exp.tolist()})
95              except Exception as e:
96                  print(f"Error fitting modified exponential function for row
                    ↪  {row_index}: {e}")
97
98              # Plotting the scatter plot and the fitted curves
99              if plot_area == True:
100                 plt.scatter(x_values, y_values, label=f'Row {row_index}')
101                 plt.xlabel('Time-of-Flight (s)')
102                 plt.ylabel('Area')
103                 plt.title(f'Scatter Plot of Time-of-Flight vs Area for Row
                    ↪  {row_index}')
104             else:
105                 plt.scatter(x_values, y_values, label=f'Row {row_index}')
106                 plt.xlabel('Time-of-Flight (s)')
107                 plt.ylabel('Max_Count')
108                 plt.title(f'Scatter Plot of Time-of-Flight vs Max_Count for Row
                    ↪  {row_index}')
109
110             if plot_exponential and params_exp is not None:
111                 x_curve_exp = np.linspace(min(x_values), max(x_values), 100)
112                 y_curve_exp = self.function.exponential_function(x_curve_exp,
                    ↪  *params_exp)
113                 plt.plot(x_curve_exp, y_curve_exp, label='Exponential Fit',
                    ↪  color='red')
114
115             if plot_modified_exponential and params_mod_exp is not None:
116                 x_curve_mod_exp = np.linspace(min(x_values), max(x_values), 100)
117                 y_curve_mod_exp =
                    ↪  self.function.modified_exponential_function(x_curve_mod_exp,
                    ↪  *params_mod_exp)
```

```
118                 plt.plot(x_curve_mod_exp, y_curve_mod_exp, label='Modified Exponential
                    ↪  Fit', color='blue')
119
120             # Save the plot
121             if plot_area == True:
122                 save_path = f'{adjusted_path}/scatter_plot_row{row_index}_area.png'
123             else:
124                 save_path = f'{adjusted_path}/scatter_plot_row{row_index}.png'
125             plt.legend()
126             plt.savefig(save_path)
127             plt.close()
128
129         # Save the results
130             if plot_area == True:
131                 results_csv_path = os.path.join(adjusted_path,
                    ↪  'fitting_results_area.csv')
132             else:
133                 results_csv_path = os.path.join(adjusted_path, 'fitting_results.csv')
134
135             with open(results_csv_path, 'w', newline='') as results_csv_file:
136                 csv_writer = csv.writer(results_csv_file)
137                 csv_writer.writerow(['row_index', 'function_type', 'params',
                    ↪  'errors'])  # Write header
138                 for result in results:
139                     csv_writer.writerow([result['row_index'], result['function_type'],
                        ↪  result['params'], result['errors']])
140             print("Function adjusted.")
141             print(f"Image saved in: {save_path}.")
142             print(f"File saved in: {results_csv_path}")
143             print("-------------------------")
144
145
146
147     def plot_peak(self, tofs, count, x1_max, y1_max, h1, plot_path, spec_number,
    ↪  spec_name):
148         print("----------------------")
149         print(f"Adjusting spec {spec_name} number {spec_number}")
150         plt.plot(tofs, count, "-", label="data")
151         plt.plot(x1_max, y1_max, ".", label="peaks")  # Use "o" for visualization
        ↪  clarity
152         plt.axhline(y=h1, color="r", label="0.025 * np.max(count)")
153         plt.xlabel("Time of Flight(s)")
154         plt.ylabel("Count")
155         plt.xticks(np.arange(0, 50, 5))
156         plt.xlim(0, 50)
157         plt.legend()
158         plt.grid()
```

```
159
160         image_file_path = os.path.join(plot_path,
            ↪  f'{spec_number}_{spec_name}_Peaks_Image.png')
161         print(f"Saving adjusted spec to {image_file_path}")
162         plt.savefig(image_file_path)
163         plt.clf()
164
```

APÊNDICE F – CLASSE FUNCTIONS CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```
1 import numpy as np
2
3 class Functions:
4     def __init__(self):
5         pass
6
7     def exponential_function(self, x, a, b):
8         return a * np.exp(b * x)
9
10    def modified_exponential_function(self, x, a, k1, k2):
11        return a * np.exp(-k1 * (1 - np.exp(-k2 * x))/(k2 * x))
```

APÊNDICE G – CLASSE DATAPROCESSOR CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```
import os

from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils

class DataProcessor:
    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()
        #self.graphic = Graphics()

    def process_multiple_files(self, directory_path, peak_path, plot_path, save_files):
        all_files = self.files.list_txt_files(directory_path)
        for file in all_files:
          filepath = self.files.construct_path_files(directory_path, file)
          spec_number = self.utils.extract_spec_number(filepath)
          spec_name = self.utils.extract_spec_name(filepath)
          print(f"Analysing Spec Number {spec_number} of {spec_name}")
          data_tof = self.files.read_files(filepath, 0)
          print("Data Time Of Flight:")
          print(data_tof)
          print("----------------------")

          data_count = self.files.read_files(filepath, 1)
          print("Data Count:")
          print(data_count)
          print("----------------------")

          peaks_path = self.calc_instance.find_data_multiple_peaks(data_tof,
          ↪  data_count, peak_path, plot_path, spec_number, spec_name, save_files)

          peak_tof = self.files.read_files_2(peaks_path, 0)
          print("Peak Time Of Flight:")
          print(peak_tof)
          print("----------------------")

          if peak_tof is not None:
              peak_count = self.files.read_files(peaks_path, 1)
              print("Peak Count")
              print(peak_count)
```

```
44                print("----------------------")
45
46                mean_bkgd, stdev_bkgd =
                  ↪  self.calc_instance.analyze_statistics_bkgd(data_tof, data_count,
                  ↪  peak_tof)
47                print("Background mean of regions before each peak:")
48                print(mean_bkgd)
49                print("----------------------")
50                print("Background Stdev of regions before each peak:")
51                print(stdev_bkgd)
52                print("----------------------")
53
54                peaks_tof_region, new_peaks_count_region =
                  ↪  self.calc_instance.count_inside_region_of_interest_minus_mean_bkgd(data_tof,
                  ↪  data_count, peak_tof, 0.02, 0.02, mean_bkgd)
55                print("ToF inside regions of interesse for peaks:")
56                print(peaks_tof_region)
57                print("----------------------")
58                print("Count inside regions of interesse for peaks:")
59                print(new_peaks_count_region)
60                print("----------------------")
61
62                max_tof_per_peak, max_count_per_peak =
                  ↪  self.calc_instance.new_max_tof_count_inside_peak_region(peaks_tof_region,
                  ↪  new_peaks_count_region, peak_path, spec_number, spec_name)
63                print("Max ToF for peaks inside regions of interesse:")
64                print(max_tof_per_peak)
65                print("----------------------")
66                print("Max count for peaks inside regions of interesse:")
67                print(max_count_per_peak)
68                print("----------------------")
69
70                area_under_peaks =
                  ↪  self.calc_instance.calc_area_for_peaks(peaks_tof_region,
                  ↪  new_peaks_count_region)
71                print("Area under peaks")
72                print(area_under_peaks)
73                print("----------------------")
74
75                error_area = self.errors.error_area(area_under_peaks)
76                print("Error area")
77                print(error_area)
78                print("----------------------")
79
80                error_count = self.errors.error_count(max_count_per_peak)
81                print("Error max count per peak")
82                print(error_count)
```

```
83                print("----------------------")
84
85
86                half_tof, error_tof = self.errors.error_tof(peaks_tof_region,
                  ↪  new_peaks_count_region, max_tof_per_peak)
87                print("Half tof per peak")
88                print(half_tof)
89                print("----------------------")
90                print("Error tof per peak")
91                print(error_tof)
92                print("----------------------")
93
94
95                result_path = os.path.join(peak_path,
                  ↪  f'result_{spec_number}_{spec_name}.txt')
96                self.files.save_to_txt_multiples(result_path, new_peaks_count_region,
                  ↪  max_tof_per_peak, max_count_per_peak, area_under_peaks, error_area,
                  ↪  error_count)
97                print(f"Results saved to: {result_path}")
```

APÊNDICE H – CLASSE DATAPROCESSOR2 CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```
import os
import pandas as pd

from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils
#from graphics.graph import Graphics

class DataProcessor2:
    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()

    def comparing_peaks(self, peaks_path):
        extracted_files = os.listdir(peaks_path)
        result_files = [file for file in extracted_files if 'result' in file]
        result_files = sorted(result_files, key=self.utils.custom_sort)

        base_file_path = os.path.join(peaks_path, result_files[10])
        print("----------------------")
        print(f"Comparing peaks found in {base_file_path} to all peaks found in the
        ↪ {len(result_files)} specs analised")
        print("----------------------")



        base_df = pd.read_csv(base_file_path, sep="\t")
        data_dict = {x: {} for x in base_df['Max_ToF']}

        for arquivo in result_files:
            file_path = os.path.join(peaks_path, arquivo)

            try:
                df = pd.read_csv(file_path, sep="\t")

                for base_x in base_df['Max_ToF']:
                    matched_row = df[df['Max_ToF'].apply(lambda x:
                    ↪  Calc.is_within_range(base_x, x))]

                    if not matched_row.empty:
                        max_tof = matched_row['Max_ToF'].iloc[0]
```

```
43                          max_count = matched_row['Max_Count'].iloc[0]
44                          area = matched_row['Area'].iloc[0]
45                          data_dict[base_x][arquivo] = {'Max_ToF': max_tof, 'Max_Count':
                            ↪  max_count, 'Area': area}
46                      else:
47                          data_dict[base_x][arquivo] = float('nan')
48              except Exception as e:
49                  data_dict[base_x][arquivo] = {'Max_ToF': float('nan'), 'Max_Count':
                    ↪  float('nan'), 'Area': float('nan'), 'Erro': str(e)}
50
51          result_df = pd.DataFrame.from_dict(data_dict, orient='index')
52
53          save_path = os.path.join(peaks_path, 'analised_specs.tsv')
54          print(f"Comparison saved in {save_path}")
55          result_df.to_csv(save_path, sep='\t')
```

APÊNDICE I – CLASSE DATAPROCESSOR3 CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils
from graphics.graphics import Graphics

class DataProcessor3:
    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()
        self.graphics = Graphics()

    def adjusted_specs(self, peak_path, adjust_spec, plot_area):
        self.graphics.adjust_spec(peak_path, adjust_spec, plot_area)

```

APÊNDICE J – CLASSE DATAPROCESSOR4 CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils

class DataProcessor4:

    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()

    def find_mass(self, adjusted_path, area= True):
        print("Find mass originated from the peaks.....")
        df = self.files.read_csv(adjusted_path, area)
        df['M'] = df.apply(self.calc_instance.cal_ejected_part_mass, axis=1)

        df_comp, area =  self.files.read_csv_multiple_sheets(adjusted_path, area)
        result_df= self.calc_instance.compare_mass(df, df_comp, adjusted_path, area)
        self.files.save_dataframes_as_txt(adjusted_path, area, result_df)
        print("Table with mass and tof calculated saved")

```

APÊNDICE K – CLASSE DATAPROCESSOR5 CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import os

from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils

class DataProcessor5:

    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()

    def find_cross_section(self, adjusted_path, area):
        print("-------------------------")
        print("Calculating cross section and radius....")
        fluence_table = self.files.read_fluence_table()
        fluence_table["experiment"] = fluence_table.apply(lambda row:
        ↪  f"{row['target']}_{row['energy']}_MeV_{row['ion']}", axis=1)

        adjusted_function = self.files.read_csv(adjusted_path, area)

        fluence = self.utils.selecting_right_experiment_row(fluence_table,
        ↪  adjusted_path)

        adjusted_function['cross_section'] = adjusted_function.apply(lambda row:
        ↪  self.calc_instance.cal_cross_section(row, fluence), axis=1)

        final_dataframe = self.calc_instance.cal_radius(adjusted_function)

        if area:
            output_filepath = os.path.join(adjusted_path, "Cross_and_radius_area.csv")
        else:
            output_filepath = os.path.join(adjusted_path, "Cross_and_radius.csv")
        self.files.df_to_csv(final_dataframe, output_filepath)
        print("Cross section and radius successfully calculated!")

```

APÊNDICE L – CLASSE DATAPROCESSOR6 CONSTRUIDA PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import sys
import os
import glob
import numpy as np
import pandas as pd
sys.path.append("../")

from calculation.calc import Calc
from files.files import Files
from errors.error import Errors
from utils.util import Utils

class DataProcessor6:

    def __init__(self):
        self.calc_instance = Calc()
        self.files = Files()
        self.errors = Errors()
        self.utils = Utils()

    def compare_final_results(self, adjusted_path, area):
        df_cross = self.files.read_csv_2(adjusted_path, area)
        df_cross = df_cross.dropna()
        df_cross = df_cross.query('error_cross_section <= 0.2 * cross_section')
        dfs_mass = self.files.read_masses_txt(adjusted_path, area)
        df_match = self.utils.add_cross_section_and_radius(dfs_mass, df_cross)
        updated_list_of_dfs = self.utils.add_formula_column_to_dfs(df_match)
        self.files.save_dataframes_as_txt_2(adjusted_path, area, updated_list_of_dfs)

```

APÊNDICE M – MÉTODO FLOW_FUNCTION CONSTRUIDO PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```python
import sys
import os

from implementation.implement import DataProcessor
from implementation.implement_2 import DataProcessor2
from implementation.implement_3 import DataProcessor3
from implementation.implement_4 import DataProcessor4
from implementation.implement_5 import DataProcessor5
from implementation.implement_6 import DataProcessor6

from files.files import Files

def flow_function(base_path, plot_area):

    data_processor = DataProcessor()
    data_processor2 = DataProcessor2()
    data_processor3 = DataProcessor3()
    data_processor4 = DataProcessor4()
    data_processor5 = DataProcessor5()
    data_processor6 = DataProcessor6()

    files = Files()

    # Creating Paths
    peak_path, plot_path,adjusted_path = files.create_paths(base_path)


    # Analising Peaks
    data_processor.process_multiple_files(base_path, peak_path, plot_path,
    ↪  "save_files")
    data_processor2.comparing_peaks(peak_path)

    # Adjusting Functions
    data_processor3.adjusted_specs(peak_path, adjusted_path, plot_area)

    # Finding mass of ejected particles
    data_processor4.find_mass(adjusted_path, plot_area)


    # Finding cross section
    data_processor5.find_cross_section(adjusted_path, plot_area)

    # Comparing Final Results
    data_processor6.compare_final_results(adjusted_path, plot_area)
```

```
44
45
46
47 if __name__ == "__main__":
48
       ↪ flow_function("/home/ubuntu/Documents/Dissertation/Polimeros/specs/1_PMMA_6_MeV_Cu",
       ↪ True)
```

APÊNDICE N – MÉTODO PARSE_ARGUMENTS CONSTRUIDO PARA O ALGORITMO DE ANALISE DE DEGRADAÇÃO DE ESPECTROS

```
1 import argparse
2
3 from flow.flow import flow_function
4
5 def parse_arguments():
6     parser = argparse.ArgumentParser(description='Description of your program')
7     parser.add_argument('base_path', type=str, help='Base path for processing')
8     parser.add_argument('--plot_area', action='store_true', help='Include to plot the
      ↪  area')
9     return parser.parse_args()
10
11 if __name__ == "__main__":
12     args = parse_arguments()
13     flow_function(args.base_path, args.plot_area)
```