Universidade Estadual de Campinas

Instituto de Física Gleb Wataghin

Instrumentação I - F530:

Programação em *Python* para aquisição de espectro de nanopartículas

Relatório Pré-Final

Aluna:

Maria Helena Gonçalves - RA:156584

Curso: Licenciatura em Física

mariahelenags10XarrobaXgmail.com

Orientador: Prof. Dr. Varlei Rodrigues

Campinas, 1 de Novembro de 2017

Conteúdo

1 Introdução

Clusters são agregados de átomos ou moléculas. Essas partículas podem ser compostas desde dois, a vários milhares de átomos, podendo ser formadas por um ou mais elementos e encontram-se na fronteira entre os átomos e o *bulk*. Seu estudo possibilita uma melhor compreensão de como as propriedades macroscópicas surgem do comportamento quântico da matéria [6, 5].

A partir do fim da década de 70 as estruturas nanométricas começaram a ser estudadas academicamente por sua relevância na área da biomedicina. Duas décadas depois surgiram os estudos na área da física, e desde de então o interesse nesta área cresce em ritmo acelerado, pois essas estruturas apresentam propriedades novas e interessantes. Essas propriedades as diferem enormemente dos sistemas macroscópicos e fornecem às nanoestruturas um grande potencial tecnológico em áreas como: química [4], eletrônica[9] e biomedicina [11, 14].

Neste contexto, uma das vertentes dos estudos realizados pelo Grupo de Física de Nanossistemas e Materiais Nanoestruturados (GFNMN) do Departamento de Física Aplicada (DFA), são nanopartículas metálicas produzidas por síntese física a partir de uma Fonte de *Clusters* e Agregados (FoCA). Este instrumento permite produzir partículas, controlando seu tamanho, sua dispersão e sua composição. A FoCA foi desenvolvida por Artur Domingues Tavares de Sá e Giulia Di Domenicantonio, ex-membros do grupo.

Para obter a distribuição de massas dessas partículas, e assim melhor estudá-las, é utilizado a técnica de espectrometria de massa por tempo de voo, em que um osciloscópio adquire os dados experimentais e, atualmente, um software baseado no linguagem *LabView* controla o sistema e registra os dados em forma digital. O uso do *LabView* tem imposto limites à flexibilidade experimental desejada pelo GFNMN, assim este projeto consiste em desenvolver um novo sistema de controle e aquisição do tempo de voo dos agregados, baseado em *Python 3.6* em substituição ao *LabView*.

2 Descrição

2.1 Fonte de *Clusters* e Agregados

A Fonte de *Clusters* e Agregados, esquematizada na Figura 1, funciona basicamente em quatro etapas: primeiramente uma nuvem de átomos é produzida; em seguida, ocorre a agregação dos átomos em nanopartículas; na sequência, o feixe de agregados, carregados eletricamente, é guiado por um conjunto de lentes eletrostásticas até chegar no espectrômetro de massa por tempo de voo, onde é possível identificar o espectro de distribuição de massa das partículas produzidas, e que serão depositadas no porta amostras. A obtenção da distribuição de tamanhos produzida possibilita o ajuste dos parâmetros da máquina para otimizar a produção do tamanho desejado [7]. Na Figura 2 podemos ver uma foto real da máquina.

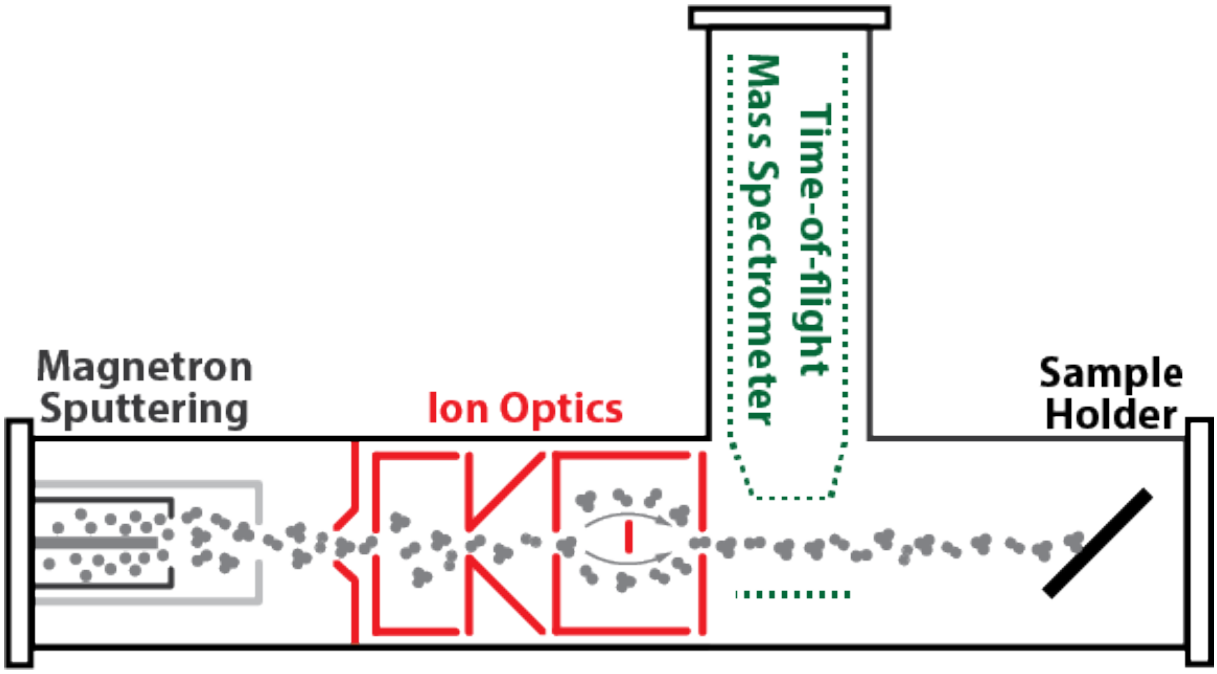

Figura 1: Diagrama da fonte de Fonte de *Clusters* e Agregados. "*Magnetron Sputtering*" é a fonte de átomos que se encontra dentro da câmara de agregação. Depois de produzido e agregado, o feixe passa por um conjunto de lentes eletrostásticas "*Ion Optics*". Uma parte do feixe é desviado para o "*Time-of-flight Mass Spectrometers*" (ToF), onde é realizada a aquisição do espectro de voo, outra parte é depositada na amostra "*Sample Holder*"[10].

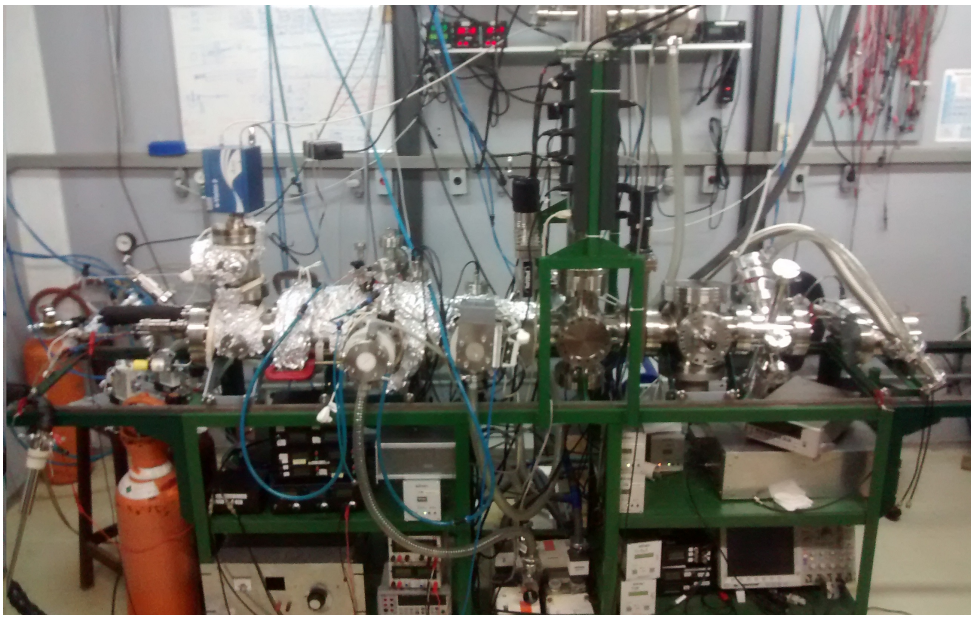

Figura 2: Imagem real da Fonte de *Clusters* e Agregados.

Para gerar a nuvem de átomos, essa máquina utiliza um *magnetron cilíndrico*, onde é gerado um plasma, e pela presença de um campo elétrico, os íons são acelerados em direção ao alvo do metal de interesse e o corrói. Na Figura 3 é possível observar um alvo de prata novo e ao lado um já erodido.

Figura 3: Foto de um alvo de prata de fio único. À esquerda temos um alvo novo e à direita temos um alvo já corroído.

2.2 Espectrômetro de massa por tempo de voo

O *time of flight mass spectrometers* (TOFMSs) funciona baseado no fato de que, ao receber a mesma quantidade de energia fornecida por um campo elétrico pulsado, partículas com massas diferentes adquirem velocidades distintas [12]. Assim, as partículas mais leves atingirão o detector antes das partículas mais pesadas.

Na Figura 4, pode-se observar o esquema do espectrômetro, onde o pulso elétrico fornece velocidade perpendicular ao feixe de partículas carregadas, que viajam dentro de um tubo de voo - livre de variação de potencial elétrico - até encontrarem um detetor de corrente. Por sua vez, o detetor faz a contagem de íons em função dos tempos de chegada.

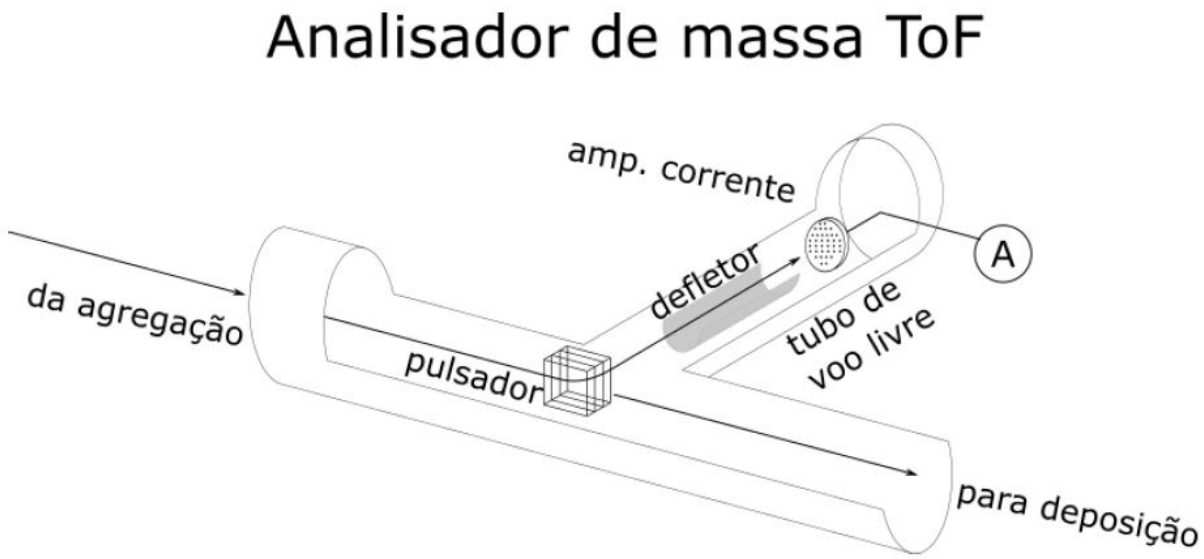

Figura 4: O pulsador é um conjunto de placas com campo pulsado, que confere uma velocidade perpendicular ao feixe de partículas eletricamente carregadas. Os defletores impedem que as partículas colidam com as paredes do tubo de voo livre. "A" representa o detector de corrente, cuja função é realizar a contagem dos íons em função do tempo de chegada. [12].

Para calcular o tempo de voo, considere um *cluster* de massa m e carga q. O campo elétrico vai fornecer energia cinética ($K = qV$) para a partícula, em que V é a voltagem fornecida pelo pulsador. Assim, é possível expressar a velocidade da partícula como:

$$v = \sqrt{\frac{2qV}{m}} \tag{1}$$

O detector de corrente está situado a uma distância L da região onde a partícula adquire velocidade. Note que essa partícula leverá um tempo t_{voo} para atingir o detector, e esse tempo pode ser calculado por:

$$t_{voo} = L \cdot \sqrt{\frac{1}{2V}\frac{m}{q}} \tag{2}$$

A corrente que chega ao detector é convertida em tensão por um amplificador IV, e o sinal é adquirido em um osciloscópio. Atualmente, utiliza-se um programa em *LabView* para controle e aquisição dos dados do osciloscópio.

A variável L possui o valor de um metro, o potencial aplicado é de $V = 7$ kV, e o tempo de voo das partículas é fornecido pelo osciloscópio, possibilitando calcular a massa das partículas.

Note que utilizando a Equação 2 é possível também calcular o t_{voo} de uma partícula se soubermos a massa. Vamos fazer uma análise para o caso de um átomo de prata. Segundo a tabela periódica um átomo de prata possui uma massa de 107.87 unidades de massa atômica, convertendo sua massa para quilos temos 1.79×10^{-25} kg. A carga de partícula é a carga elementar de um elétron 1.60×10^{-19} C, e então seu tempo de voo vai ser aproximadamente 17.6 μs.

Podemos também escrever a massa de um *cluster* em função da massa de uma outra partícula da qual conhecemos a massa e o tempo de voo.

$$M = \left(\frac{t}{t'}\right)^2 \cdot M' \tag{3}$$

Podemos estabelecer uma relação que futuramente vai permitir diferenciar outros picos de prata, e então calibrar o espectro de partículas produzido, confirmando o que foi depositado na amostra.

3 Desenvolvimento do projeto

O presente projeto procurou desenvolver um programa de controle e aquisição de dados em *Python 3.6*, utilizando as bibliotecas *PyVisa* [3] , *NumPy*[2] e *Matplotlib*[1]. O osciloscópio usado será um TEKTRONIX modelo DPO4054. Com as ferramentas citadas pretendemos realizar conjuntamente o controle do osciloscópio, a aquisição do espectro de massa das partículas e o processamento dos dados, tornando esse processo mais flexível e ágil para que os usuários aproveitem ao máximo o potencial da Fonte de *Clusters* e Agregados.

3.1 Novo sistema de aquisição de dados

Para dar início a um novo sistema de aquisição, foi estudado o manual do osciloscópio TEKTRONIX modelo DPO4054 [15], e a biblioteca *PyVisa* [3] onde foi possível encontrar comandos fundamentais para o controle do osciloscópio, os quais serão tratados a seguir.

Primeiramente era necessário estabelecer uma conexão entre o computador e o osciloscópio. Com o estudo da biblioteca *PyVisa*, foram aprendidos comando que permitem controlar todos os tipos de dispositivos de medição e vamos utilizá-la para controlar o osciloscópio em questão.

O primeiro bloco de comandos do programa de aquisição verifica as conexão existentes nas portas USB, e então estabelece uma comunicação com o osciloscópio. Para verificar a comunicação foi utilizado um comando padrão (<*IDN?>) pertencente a biblioteca *PyVisa*.

O segundo bloco do programa é referente a algumas configurações de aquisição de dados do equipamento, para tanto foi desenvolvido um programa que adquire os dados em linguagem binária. Seguem os principais comandos utilizados para realizar a aquisição:

- $< ENCDg\ ribinary >$ os dados adquiridos em linguagem binária são representados por valores inteiro negativo ou inteiro positivo. Quando a largura do byte é um, como no nosso caso, os intervalos de dados inteiros assinados de -128 a 127 e os valores inteiros positivos variam de 0 a 255. O comando utilizado aqui especifica a representação de ponto pertencente aos dados, inteiro assinada com o máximo

significativo byte transferido primeiro.

- *<HORizontal:RECOrdlength 10000>* especifica que 10000 pontos de dados serão adquiridos para cada registro realizado.

Ainda nesse bloco são adquiridas algumas configurações de escala de tempo que serão utilizadas futuramente.

O próximo bloco do programa são adquiridas 100 curvas, cada uma contendo 10000 pontos, para que seja obtido uma boa estatística de dados. Dentro desse realizamos a montagem histograma, a partir da definição da linha de ruído.

Após os dados adquiridos do osciloscópio, é necessário montar o eixo do tempo, uma vez que o osciloscópio fornece apenas os dados verticais. Com o comando *<HORizontal:SCAle?>* foi obtido como resposta a indicação que a escala principal está atualmente configurada para $100\mu s$ por divisão e então foi montada a escala do tempo.

Os próximos dois blocos do programas plota o gráfico e salva em um arquivo os dados adquiridos. O código para aquisição de dado pode ser conferido na Secção 5

Para testar o programa foi realizado um experimento com prata para a calibração da máquina, o gráfico com os dados brutos pode ser visto na Figura 5.

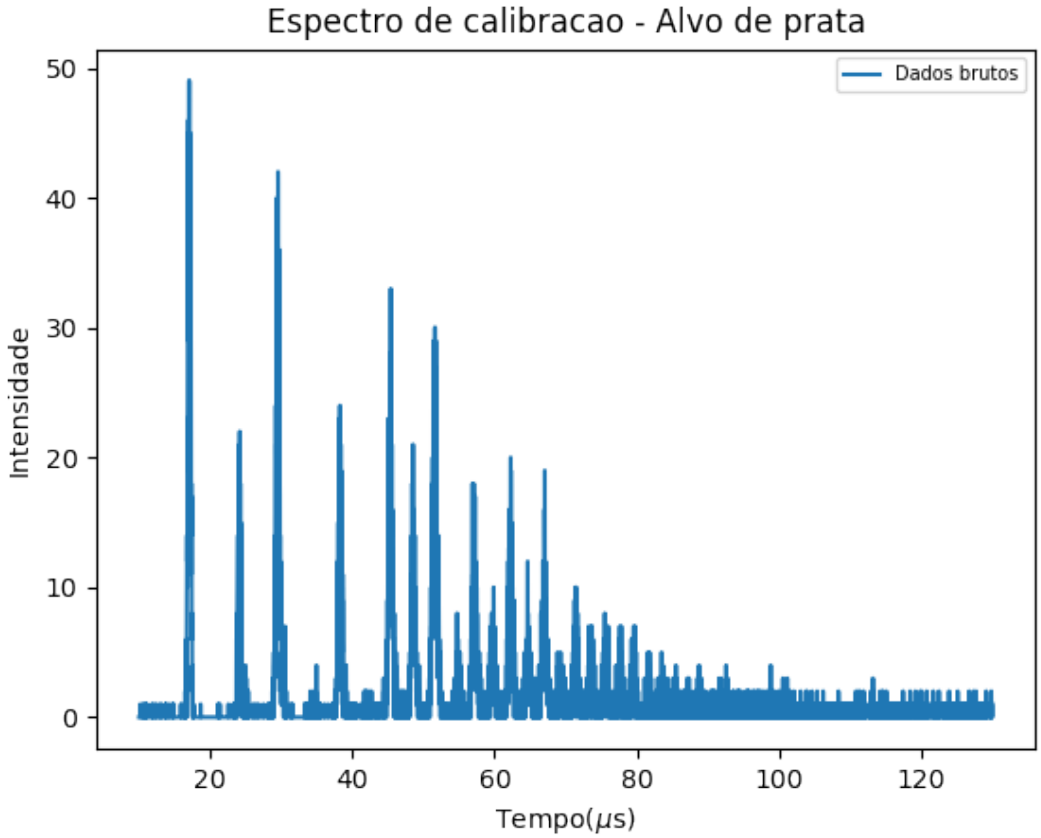

Figura 5: Espectro de calibração sem tratamento de dados.

3.2 Processamento dos dados adquiridos

Foi elaborado um código, em *Python 3.6*, que efetuasse o processamento de dados de um experimento realizado em novembro pela aluna com partículas de prata. Esse código pode ser conferido na Secção 5.

Para aquisição desse espectro foi utilizado o novo sistema de aquisição apresentado na secção 3.1, em *Python 3.6*, que fornece o tempo de voo das partículas e as intensidade dos picos em formato digital, adquiridos do osciloscópio.

O objetivo desse experimento era obter, com o novo sistema de aquisição em *Python 3.6*, um espectro de massa que contivesse picos bem definidos de *clusters* de prata, principalmente os *clusters* com números de átomos menores como por exemplo $1, 2, 3$ ou 4 átomos, para que pudesse ser feita a calibração da máquina a partir do código apresentado nessa secção.

O gráfico da Figura 6, mostra uma curvas com: os dados brutos, redução de ruído, com linha de base em zero e renormalização dos dados pelo tempo.

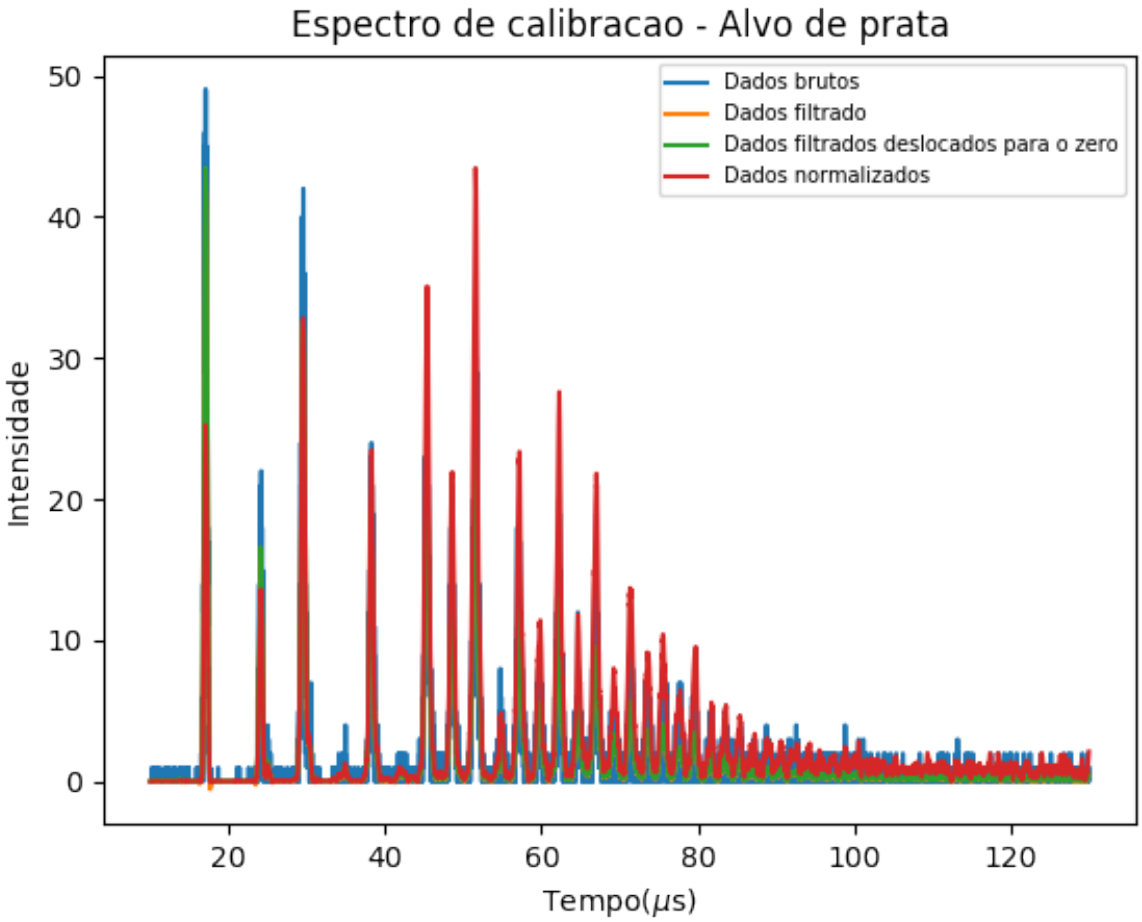

Figura 6: Espectro de calibração com processamento de dados.

A curva azul da Figura 6, foi obtida plotando diretamente os dados adquiridos pelo novo sistema de aquisição de dados. A curva laranja foi obtida por meio de uma suavização *Savitzky-Golay filter* pertencente a biblioteca *Scipy*, essa suavização foi necessária para diminuir os ruídos dos dados adquiridos. A partir dos dados suavizados foi criado uma rotina que descolasse os dados para que todos eles ficassem positivos, isso está representado pela curva verde. Na curva vermelha é feita uma normalização pela eficiência do detector de corrente, uma vez que as partículas com menor massa são melhor detectadas do que as partículas de maiores massas.

Após o tratamento de dados, era desejado agora que o código encontrasse os picos do espectro. Com o auxilio da biblioteca *Scipy* foram identificados muitos picos na curva

apresentada, os quais foram sinalizados no gráfico da Figura 7, para facilitar a visualização e indentificação, os pico foram plotados em relação aos índices e não em fução do tempo.

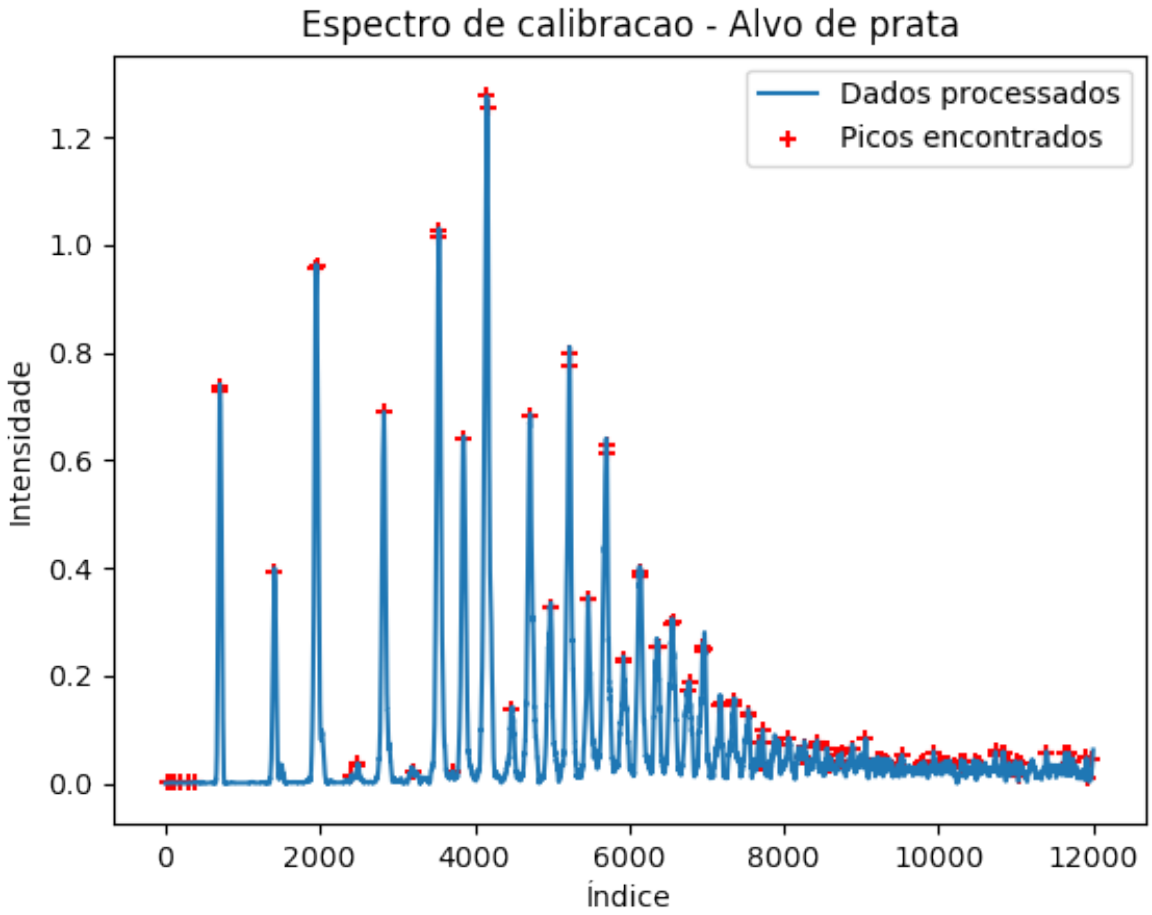

Figura 7: Espectro de calibração com picos marcados.

Note que foram identificados vários picos que não são desejados. Para resolver esta dificuldade foram criadas algumas rotinas para serem usadas com intervenção do operador, após a inspeção dos picos indesejados. Uma das rotinas é usada para definir uma altura mínima para os picos e também uma distância mínima entre picos imediatamente vizinhos. Também foram criados rotinas para inserir picos que não foram contabilizados e retirar picos que ainda não são desejados e foram selecionados. O resultado final pode ser visto no gráfico da Figura 8.

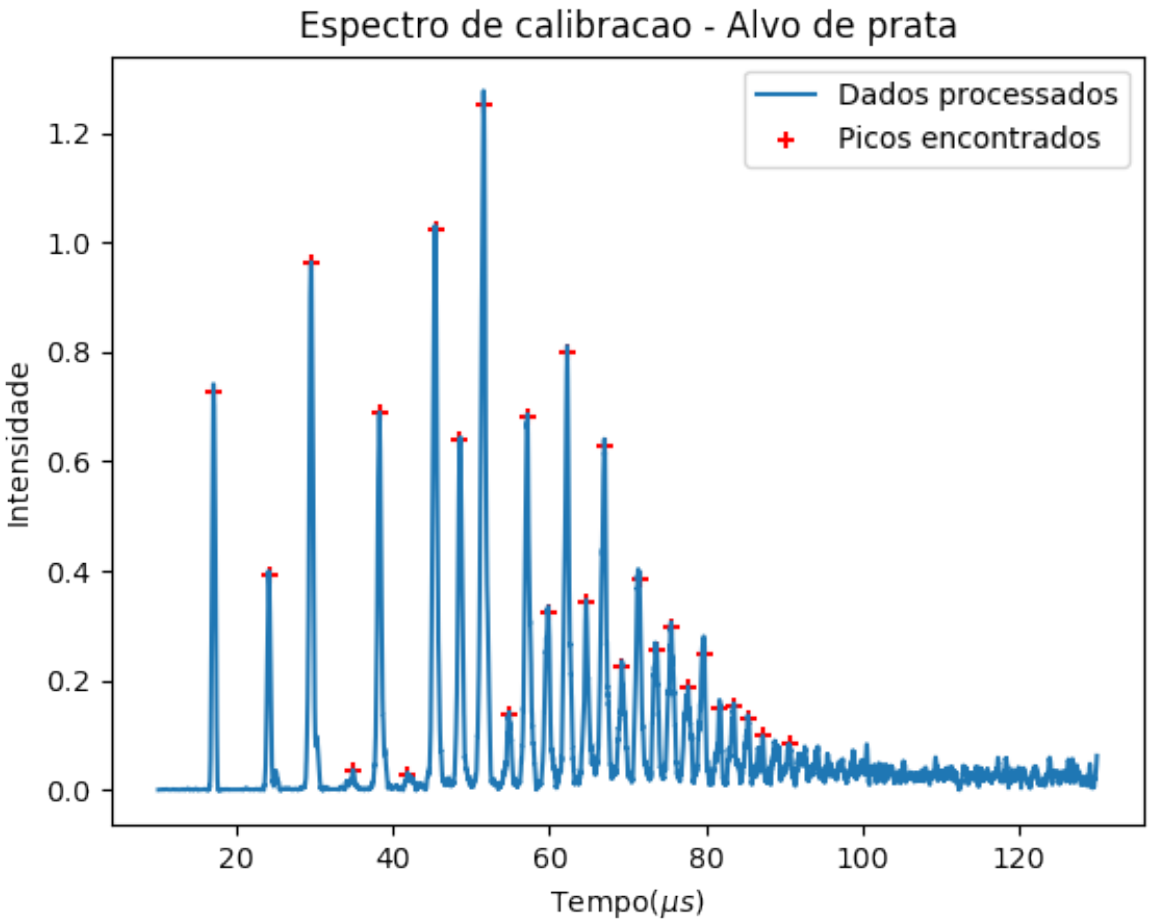

Figura 8: Espectro de calibração com picos selecionados após inspeção visual do operador.

Com o objetivo de obter uma maior precisão na determinação dos tempos de voo das partículas de prata, foi feita uma função que traçasse uma curva gaussiana em cima de cada pico do espectro. É possível ver o resultado dessa função na Figura 9.

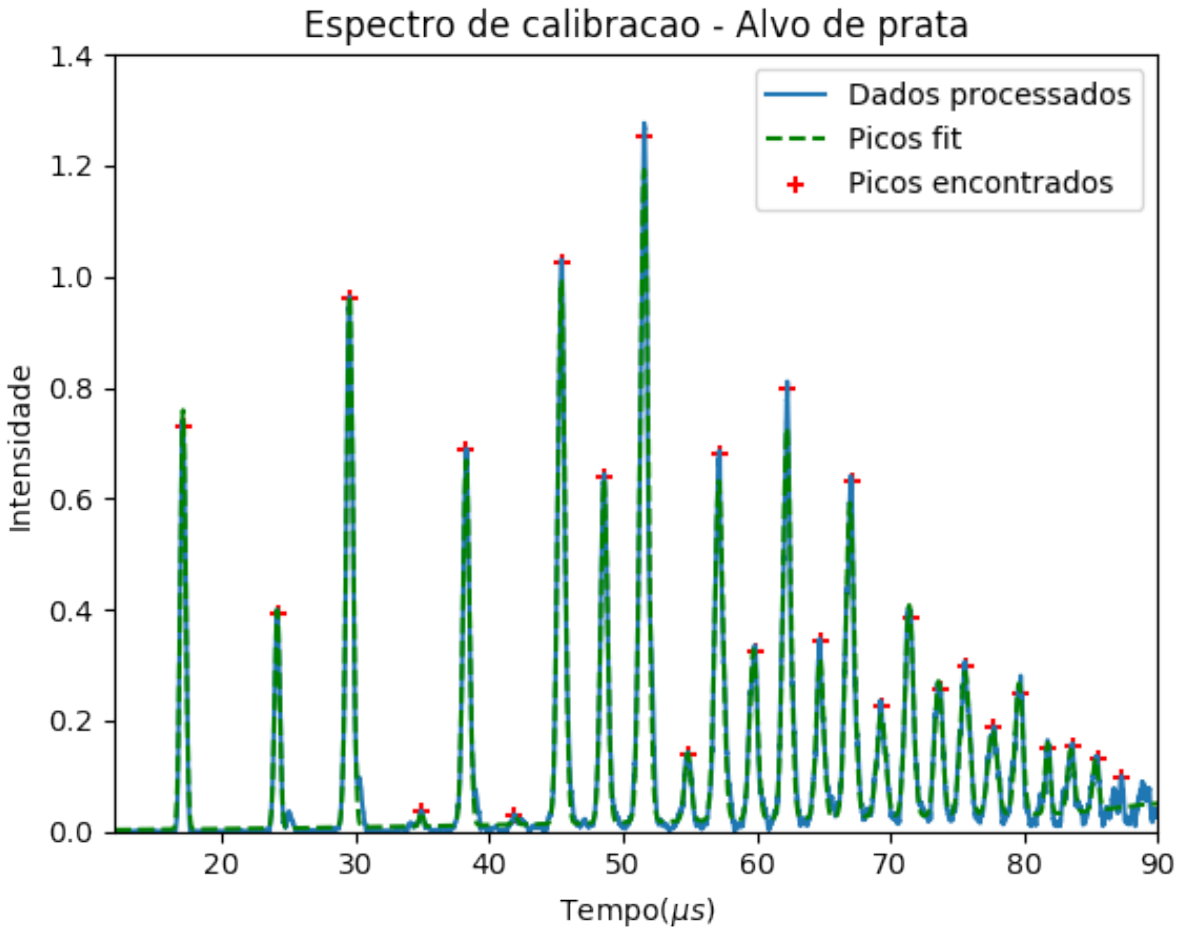

Figura 9: Curva de calibração com regressão linear.

A Tabela 1 mostra os valores dos picos encontrados, em comparação com os valores dos picos calculados teoricamente pela Equação 3. Dentro de uma certa tolerância é possível perceber que os picos encontrados correspondem com os que eram esperados.

Sabendo o número de átomos de prata que corresponde a cada tempo de voo, é possível montar uma curva de calibração Massa $\times$ tempo, e assim realizar uma regressão quadrática para termos coeficientes que vão permitir identificar a massa de todo o espectro adquirido.

Para realizar a regressão quadrática do gráfico da Figura 11 foi utilizado a biblioteca *NumPy* que fornece a curva e seus respectivos parâmetros.

Tabela 1: Tempo de voo dos átomos de Prata

Prata (n)	Tempo de voo teórico (s)	Tempo de voo encontrado (s)
1	1.73E-05	1.71E-05
2	2.45E-05	2.41E-05
3	3.00E-05	2.96E-05
4	3.46E-05	3.49E-05
5	3.87E-05	3.83E-05
6	4.24E-05	4.19E-05
7	4.58E-05	4.54E-05
8	4.89E-05	4.86E-05
9	5.19E-05	5.16E-05
10	5.47E-05	5.48E-05
11	5.74E-05	5.72E-05
12	5.99E-05	5.99E-05
13	6.24E-05	6.23E-05
14	6.47E-05	6.47E-05
15	6.70E-05	6.71E-05
16	6.92E-05	6.93E-05
17	7.13E-05	7.15E-05
18	7.34E-05	7.37E-05
19	7.54E-05	7.56E-05
20	7.74E-05	7.77E-05
21	7.93E-05	7.98E-05
22	8.11E-05	8.18E-05
23	8.30E-05	8.36E-05
24	8.48E-05	8.55E-05

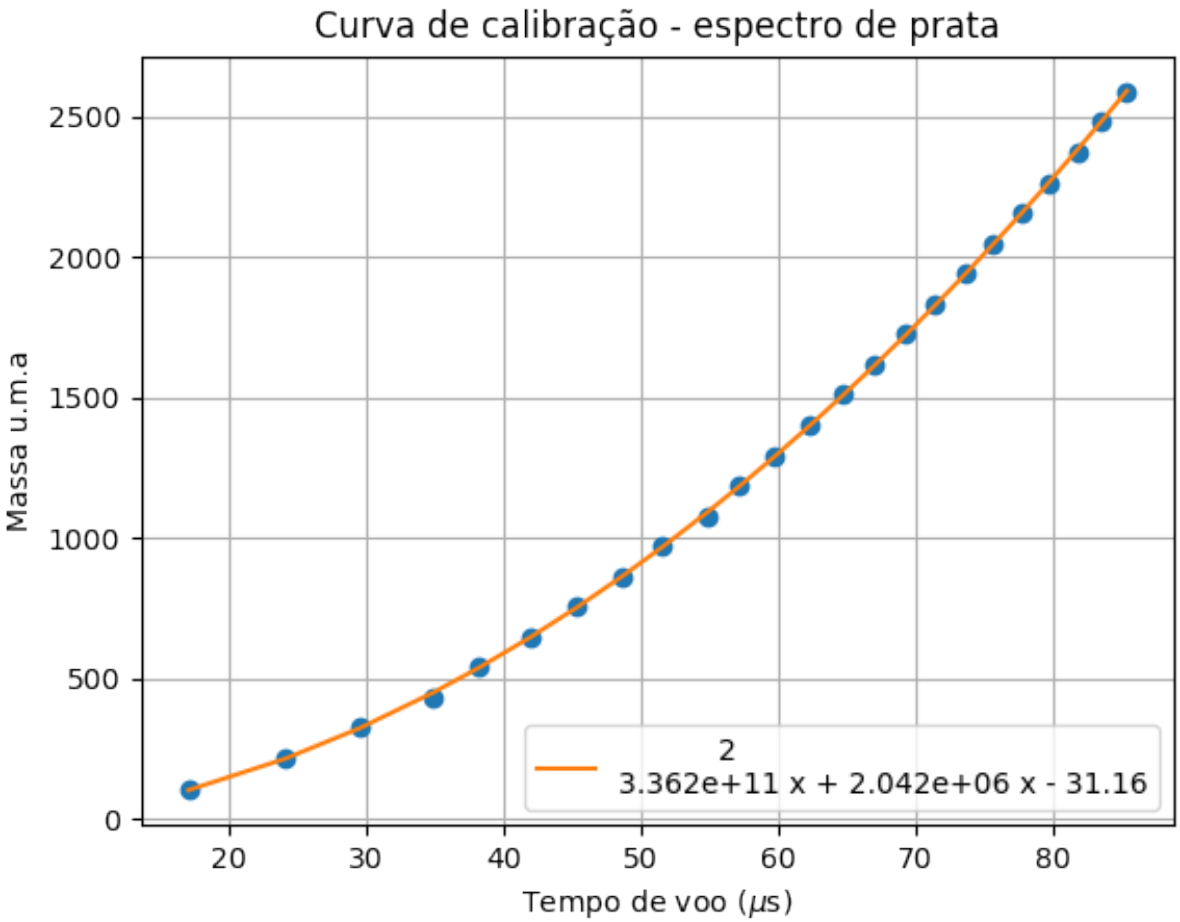

Figura 10: Curva de calibração com regressão quadrática.

Dessa forma temos que a massa (M) das partículas de prata podem ser calculadas apartir da Equação 4:

$$M = 3.362e^{11}t^2 + 2.042e^{6}t - 31.16 \tag{4}$$

Utilizando a Equação 4 é possível mudar o eixo de tempo de voo, da aquisição de dados, para o seu valor corresponde em massa, conforme do gráfico da Figura 11.

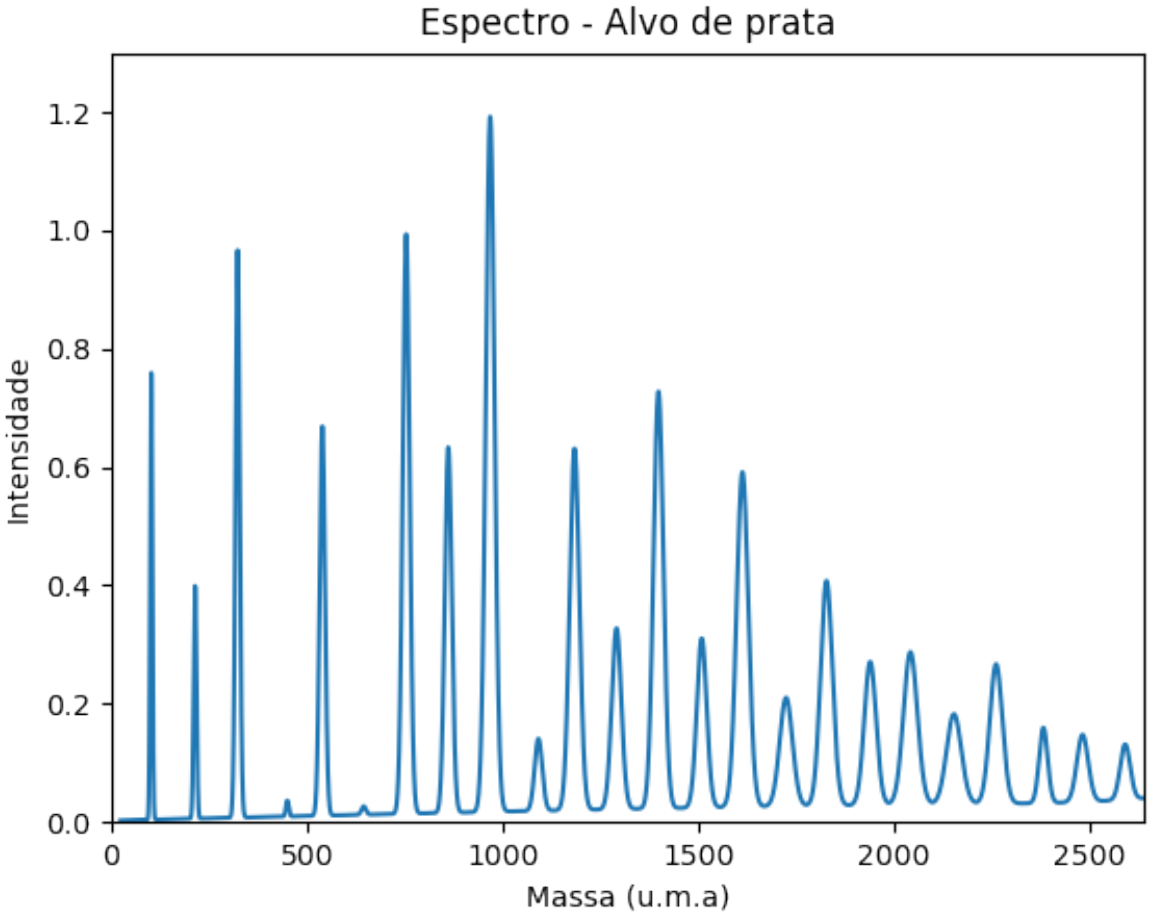

Figura 11: Espectro de prata plotado na forma de intensidade por massa em unidade de massa atômica.

4 Considerações pré-finais

4.1 Considerações pré-finais da aluna

A linguagem *Python*, por sua grande versatilidade, vem sendo muito utilizada dentro e fora do meio acadêmico, por isso sua aprendizagem é de suma importância. Podemos citar alguns projetos comerciais de grande relevância que utilizam a linguagem *Python*, como por exemplo: o sítio *YouTube*; a provedora global de filmes e séries *Netflix*; o serviço de música digital *Spotify*. Outras grandes organizações que usam a linguagem incluem *Google*, *Yahoo!* e *NASA*.

Dentro do meio acadêmico podemos citar como aplicações mais atuais a utilização da *HyperSpy*, uma biblioteca *Python* que fornece ferramentas para facilitar a análise de dados interativos de multidimensionais com o uso da tecnologia *machine learning*, como por exemplo realizar o *PCA* (*"Principal component analysis"*)[13]. Há muitas outras bibliotecas, que são recursos dentro da linguagem *Python*, como *NumPy*, *SciPy*, *StatsModels*, *scikit-learn* e *pandas*, que oferecem aos acadêmicos ferramentas necessárias, acessíveis e rápidas, principalmente se comparadas com os outros pacotes para a ciência de dados utilizados (como *MatLab*, *Stata* e *R*).

Meu contato com *Python* era reduzido, e com esse projeto tive a oportunidade de ser introduzida nos modelos atuais de tratamento de dados realizados dentro da ciência, além de ampliar os horizontes para as aplicações dessa linguagem, como por exemplo o que foi desenvolvido nesse projeto, que possibilita o controle de portas USB através de comandos gerais aprendidos, e também o controle específico do equipamento laboratorial de ponta, o osciloscópio TEKTRONIX modelo DPO4054.

Os conhecimentos adquiridos por mim sobre essa nova linguagem contribuem para uma formação atualizada e interada com os processos atuais que ocorrem no âmbito acadêmico.

4.2 Considerações pré-finais do orientador

Meu orientador concorda com o expressado neste relatório parcial e deu a seguinte opinião: O relatório está muito bem escrito, conciso e objetivo. A aluna trabalhou com dedicação, disciplina e bastante independência. O projeto foi executado dentro do cronograma prevista e foi concluído com êxito.

5 Apêndice

5.1 Código em Python: aquisição de dados

Código utilizado para realizar aquisição de dados.

```
import visa
import numpy as np
import matplotlib.pyplot as plt
import time

### Parametros de entrada ###

nome_arq = 'ponto_a_1.dat'
canal = 'CH2'

#### Conectar instrumento ####

rm = visa.ResourceManager()
instruments = (rm.list_resources())

for test in instruments:
    inst = rm.open_resource(test)
    idn = inst.query("*IDN?")
    if idn.find("TEK") != -1:
        inst.close()
        tek = rm.open_resource(test)

    else:
        inst.close()

print(tek.query("*IDN?"))

TEKTRONIX,DPO4054,C021939,CF:91.1CT FV:v2.48

#### Adquirindo em Bin ####

tek.write('data:encdg ribinary')
tek.write('data:width 1')
```

```
34 Y=tek.query_binary_values('CURVe?', datatype='b', is_big_endian=True,
       container = np.array)
35 tek.write('HEADER OFF')
36 tek.write("DATa:SOUrce "+canal)
37 tek.query("*OPC?")
38 tek.write("DATa:ENCdg ASCIi")
39 tek.write("HORizontal:RECOrdlength 10000")
40 tek.write('Data:Width 1')    # set the data width to 1 byte
41 tek.query("*OPC?")
42
43 '1\n'
44
45
46
47 YOFF_in_dl = float(tek.query("WFMPre:YOFf?")[:-1])
48 tek.query("*OPC?")
49 YMUlt = float(tek.query("WFMPre:YMUlt?")[:-1])
50 tek.query("*OPC?")
51 YZERO_in_YUNits = float(tek.query("WFMPre:YZEro?")[:-1])
52 tek.query("*OPC?")
53 curve_in_dl = tek.query("CURVe?")
54 tek.query("*OPC?")
55 curve_in_dl = [float(x) for x in curve_in_dl.split(',')]
56 contagem = np.zeros(len(curve_in_dl))
57
58 10000
59 10000
60
61 tek.write('Data:ENCDg ribinary')  # set the instrument to
62 tek.query("*OPC?")
63
64
65 inicio = time.time()
66 for i in range(100):
67     curve_in_dl = tek.query_binary_values('CURVe?', datatype='b',
       is_big_endian=True)
68     tek.query("*OPC?")
69     curve_in_dl = [float(x) for x in curve_in_dl]
70     Y = np.array ([(value - YOFF_in_dl) for value in curve_in_dl])
71     posicao =np.where(Y>20.0)
```

```
72      for i in posicao:
73          contagem[i] = contagem[i]+1
74  fim = time.time()
75
76  print(fim - inicio)
77
78  #### Montar eixo tempo ####
79
80  time_scale = float(tek.query("HORizontal:SCAle?")[:-1])
81  time_step = time_scale/1000
82  time = [(i+1)*time_step for i in range(len(Y))]
83
84  tek.close()
85
86  ### salvar dados ###
87  arq = open(nome_arq, 'w')
88  arq.write('time (s) \t' + canal + ' (V)\n' )
89  for i in range(len(contagem)):
90      arq.write(str(time[i]) + '\t')
91      arq.write(str(contagem[i]) + '\n')
92  arq.close()
93
94  print (len(contagem))
95
96  ### grafico ###
97  plt.plot(time,contagem)
98  plt.grid(True)
99  plt.xlabel('time [s]', fontsize=18)
100 plt.ylabel(canal+'(V)', fontsize=18)
101
102 plt.show()
```

5.2 Código em Python: análise de dados

Código utilizado para realizar análise de dados.

```
1 %matplotlib nbagg
2 import numpy as np
3 import matplotlib.pyplot as plt
```

```
4  from scipy import signal
5  from scipy.optimize import curve_fit
6
7
8  # Leitura dos dados brutos
9
10 arq_dados = '
       segundo_histograma_dados_calib_bin_integral_10k_100curves_teste.txt'
11 i = 12000
12 f = 24000
13 t = np.loadtxt(arq_dados,skiprows=(2), usecols=(0), delimiter='\t')[i:f]/10
14 y_bruto = np.loadtxt(arq_dados,skiprows=(2), usecols=(1), delimiter='\t')[i
       :f]
15
16 plt.plot(t*1000000,y_bruto, label='Dados brutos')
17 plt.title('Espectro de calibracao - Alvo de prata')
18 plt.xlabel('Tempo($\mu$s)')
19 plt.ylabel('Intensidade')
20 plt.legend(loc=0, prop={'size': 7})
21 #plt.axis([1,10, -0.5, 50])
22 plt.savefig('dados_brutos.png')
23 plt.show()
24
25
26 # Reducao de ruido
27
28 y_processado = signal.savgol_filter(y_bruto,51,2)
29 plt.plot(t*1000000,y_processado, label='Dados filtrado')
30
31 plt.legend(loc=0, prop={'size': 7})
32 #plt.axis([0.00001, 0.00010, -0.5, 50])
33 plt.show()
34
35
36 # Colocar a linha de base em zero
37
38 linha_base = np.amin(y_processado[:500])
39 #print(linha_base)
40 y_processado = y_processado-linha_base
41
```

```
42 y_processado[y_processado<=0] = 0
43 plt.plot(t*1000000,y_processado, label = 'Dados filtrados deslocados para o
       zero')
44 plt.legend(loc=0, prop={'size': 7})
45 #plt.axis([0.00001, 0.00010, -0.5, 50])
46
47 plt.show()
48
49
50 # Renormalizados dos dados por t - eficiencia das MCPs
51
52 max_i = np.amax(y_processado)
53 y_processado = np.multiply(y_processado,t)
54 max_f = np.amax(y_processado)
55
56 plt.plot(t*1000000, (max_i/max_f) * y_processado, label = 'Dados
      normalizados')
57 plt.legend(loc=0, prop={'size': 7})
58 #plt.axis([0.00001, 0.00010, -0.5, 50])
59 plt.show()
60 plt.savefig('dados_pre_processados.png')
61
62 plt.close()
63
64
65
66
67 # procura picos no histrograma
68
69
70 peakind = signal.find_peaks_cwt(y_processado, [10,50])
71 y_picos = y_processado[peakind]
72 i = range(len(y_processado))
73
74
75 plt.close()
76 plt.title('Espectro de calibracao - Alvo de prata')
77 plt.xlabel('indice')
78 plt.ylabel('Intensidade')
79 plt.plot(i, y_processado, label = 'Dados processados')
```

```
80 plt.scatter(peakind, y_picos,color='red', marker='+', label = 'Picos
       encontrados')
81 plt.legend()
82 plt.savefig('espectro_calib_peaks_todos.png')
83 plt.show()
84
85 print(peakind)
86
87
88 # Eliminar picos erroneamente detectados:
89
90 print(len(peakind))
91 print(peakind)
92
93 max_ampl = 0.08
94 min_dis = 50
95 picos_add = np.array([3185,2489])
96 picos_del = np.array([11993])
97
98 peakind = peakind[y_processado[peakind]>max_ampl]
99
100 print(len(peakind))
101 print(peakind)
102
103 j = np.empty([0])
104 for i in range(len(peakind))[:-1]:
105     if (peakind[i+1] - peakind[i]) > min_dis:
106         j = np.append(j,peakind[i])
107
108 peakind = np.copy(j.astype(int))
109
110 print(len(peakind))
111 print(peakind)
112
113 #adicina picos
114 peakind = np.append(peakind,picos_add)
115
116 print(len(peakind))
117 print(peakind)
118
```

```
119 #deleta picos
120 peakind = np.setdiff1d(peakind,picos_del)
121
122
123 print(len(peakind))
124 print(peakind)
125
126 y_picos = y_processado[peakind]
127 plt.scatter(peakind, y_picos,color='green', marker='*', label = 'Picos
        encontrados')
128 plt.legend()
129 plt.savefig('espectro_calib_peaks_selecionados.png')
130 plt.show()
131
132 y_picos = y_processado[peakind]
133 t_picos = t[peakind]
134
135 plt.close()
136 plt.title('Espectro de calibracao - Alvo de prata')
137 plt.xlabel('Tempo($\mu s$)')
138 plt.ylabel('Intensidade')
139 plt.plot(t*1000000, y_processado, label = 'Dados processados')
140 plt.scatter(t_picos*1000000, y_picos, color='red', marker='+', label = '
        Picos encontrados')
141 plt.legend()
142 plt.savefig('espectro_calib_peaks_tempo.png')
143 plt.show()
144
145 arq = open("picos_times.txt", 'w')
146 for i in range(len(t_picos)):
147     arq.write(str(t_picos[i]) + '\n')
148 arq.close()
149
150 def func(x, *params):
151     y = np.zeros_like(x)
152     for i in range(0, len(params), 3):
153         ctr = params[i]
154         amp = params[i+1]
155         wid = params[i+2]
156         y = y + amp * np.exp( -((x - ctr)/wid)**2)
```

```
157     return y
158
159 largura = 2e-7
160
161
162 guess = np.column_stack((t_picos,y_picos,np.full(len(peakind),largura)))
163 #guess = [705,3.15,100,1409,0.86,100, 1949,3.1,100]
164
165 popt, pcov = curve_fit(func, t, y_processado, p0=guess)
166 print (popt)
167 fit = func(t, *popt)
168
169
170
171 plt.plot(t*1000000, fit , '--', color='green', label = 'Picos fit')
172 plt.legend()
173 plt.axis([12, 90, 0, 1.4])
174 plt.savefig("picos.pdf", bbox_inches='tight')
175 plt.show()
176
177 plt.savefig("picos_gaussiana.png", bbox_inches='tight')
178
179 len(fit)
180
181 12000
182
183 tempo = np.reshape(popt, (3,int(len(popt)/3)), order='F')[0]
184 amplitude = np.reshape(popt, (3,int(len(popt)/3)), order='F')[1]
185 largura = np.reshape(popt, (3,int(len(popt)/3)), order='F')[2]
186 print(tempo)
187 print(amplitude)
188 print(largura)
189
190
191 #Calcula a massa a partir dos picos
192 m = []
193
194 for i in range (len(tempo)):
195    m.append(107.86 * (i+1))
196
```

```
197 print (m)
198 print(tempo)
199
200
201 coefficients = np.polyfit(tempo,m, 2)
202 polynomial = np.poly1d(coefficients)
203 ys = polynomial(tempo)
204 plt.close()
205
206 plt.plot(tempo*1000000, m, 'o')
207 plt.plot(tempo*1000000,ys,label=(polynomial))
208 plt.legend(loc='lower right')
209 plt.grid(True)
210
211 plt.title('Curva de calibracao - espectro de prata')
212 plt.xlabel('Tempo de voo ($\mu$s)')
213 plt.ylabel('Massa u.m.a')
214 plt.savefig('curv_calib_cali1_1000.png')
215 plt.show()
216
217 print (coefficients)
218 print (polynomial)
219
220
221 # Com os coeficientes calcula da massa do espectro
222 novo_vetor=[]
223 print(coefficients[0], coefficients[1], coefficients[2])
224
225
226 plt.close()
227 novo_vetor= ((coefficients[0]*t*t)+ coefficients[1]*t + coefficients[2])
228 #pyplot.figure(figsize=(10,6))
229 plt.axis([0, 2800, 0, 1.3])
230 plt.plot(novo_vetor, fit)
231 plt.title('Espectro - Alvo de prata')
232 plt.xlabel('Massa (u.m.a)')
233 plt.ylabel('Intensidade')
234 plt.savefig('espec_calib_ag_massa_cali1_1000.png')
235 plt.show()
236 plt.close()
```

5.3 Horário para evento de consulta à comunidade - CàC

Evento de Consulta à Comunidade - CàC:

- **Data:** 07 de Novembro de 2017
- **Horário:** 16 às 19 h
- **Turma escolhida:** segunda turma (17-19 h)

Referências

[1] Matplotlib. https://matplotlib.org/, acesso 25/08/2017.

[2] Numpy. http://www.numpy.org/#, acesso 25/08/2017.

[3] Pyvisa: Control your instruments with python. https://pyvisa.readthedocs.io/en/stable/, acesso 25/08/2017.

[4] Bell, A. T. The impact of nanoscience on heterogeneous catalysis.

[5] Brack, M. *Rev. Mod. Phys. 65* (1993), 677.

[6] de Heer, W. A. *Rev. Mod. Phys. 65* (1993), 611.

[7] de Sá, A. D. T. *Desenvolvimento de uma fonte de Nano-agregados metálicos*. Universidade Estadual de Campinas, Campinas, SP, 2009.

[8] de Sá, A. D. T. *Nano-agregados metálicos: Produção e propriedades magnéticas*. Universidade Estadual de Campinas, Campinas, SP, 2013.

[9] Dietl, T. *Nature Materials 2* (2003), p. 646–648.

[10] Huttel, Y. *Gas-Phase Synthesis of Nanoparticles*. Wiley, 2017.

[11] Kim, J. S., Kuk, E., Kyeong Nam Yu and, J.-H. K., Park, S. J., Lee, H. J., Kim, S. H., Park, Y. K., Park, Y. H., Hwang, C.-Y., Kim, Y.-K., Lee, Y.-S., and Dae Hong Jeong and, M.-H. C. Antimicrobial effects of silver nanoparticles. *Nanomedicine: Nanotechnology, Biology, and Medicine* (2007).

[12] Rodrigues, K. L. *Espectroscopia de capacitância de nanoagregados selecionados em massa*. Universidade Estadual de Campinas, Campinas, SP, 2015.

[13] Shlens, J. A tutorial on principal component analysis.

[14] Tarek M.Fahmy, Peter M.Fong, A. G. W. M. S. Targeted for drug delivery.

[15] Tektronix. *MSO4000 and DPO4000 Series Programmer Manual*. Beaverton, USA.