Produção de material didático: Análise de dados com Python em disciplinas experimentais do IFUSP

Alunos: Laura Oliveira Dias e Gabriel Meneghel de Sousa Luiz
Orientador: Felix Guillermo Gonzalez Hernandez

Fevereiro 2025

1 Resumo

O projeto visa ensinar e orientar os alunos para que incorporem a programação (Python) em suas análises de dados, as quais são exigidas, pelo menos, durante metade da graduação do Bacharelado de Física. Assim, revisamos os principais softwares que são utilizados em sala de aula, além dos experimentos oferecidos durante as disciplinas de Física Experimental. Com tais revisões, criamos materiais didáticos que permitem que o aluno possa usar o Python para fazer todas as atividades exigidas durante essas disciplinas, mesmo sem ter um conhecimento prévio de programação. Os materiais disponibilizados tem aplicabilidade direta para 6 semestres, o que corresponde a 75% do curso. São beneficiadas 6 turmas simultâneamente, contando os turnos noturno e integral.

2 Introdução

O python, ao invés de outras linguagens, é utilizado justamente por sua praticidade e popularidade, sendo uma língua de alto nível, que não requer tantas linhas de código e possui fácil visualização. Além disso, consideramos que é a linguagem que os alunos têm contato com a disciplina obrigatória (para o bacharelado) de Introdução à Computação para Ciências Exatas e Tecnologia.

Para começar a desenvolver os materiais e as videoaulas, revisamos os tipos de análises de dados que são exigidas aos alunos nas disciplinas de Física Experimental I a V. Assim, pode-se realizar o material didático que será utilizado como base das videoaulas e é usado para o programa disponibilizado e oferecido aos alunos que cursam/cursarão tais disciplinas.

Para a primeira parte do projeto, revisamos experimentos específicos oferecidos nas disciplinas de Física Experimental I e III, os quais foram: Fractais, Queda Livre, Massa-mola, Lei de Ohm, Circuito RC e Resistor/pilha.

O python, ao invés de outras linguagens, é utilizado justamente por sua praticidade e popularidade, além disso, consideramos que é a linguagem que os alunos têm contato com a disciplina obrigatória (para o bacharelado) de Introdução à Computação para Ciências Exatas e Tecnologia.

2.1 Objetivos

O principal objetivo do projeto é apresentar aos alunos que vão cursar as disciplinas experimentais do IFUSP uma outra forma mais poderosa de realizar as análises, considerando os softwares que são geralmente utilizados durante Física Experimental I a V. Dessa maneira, o aluno pode, além de ter maior liberdade em suas análises, obter uma base mais forte em programação, a qual é introduzida aos alunos (de bacharelado) no segundo semestre.

3 Método: Python

Python é uma linguagem de programação de alto nível, conhecida por sua sintaxe simples e legível, o que a torna uma das linguagens mais populares do mundo. Criada por Guido van Rossum nos anos 90, ela é amplamente utilizada em diversas áreas, como desenvolvimento web, ciência de dados, inteligência artificial, automação, análise de dados e até mesmo em aplicações embarcadas. Sua versatilidade e grande número de bibliotecas fazem com que Python seja uma escolha ideal tanto para iniciantes quanto para programadores experientes. Além disso, a linguagem conta com uma comunidade ativa e uma vasta documentação, facilitando o aprendizado e a resolução de problemas.

Aprender Python é importante porque abre portas para diversas oportunidades no mercado de trabalho e possibilita o desenvolvimento de soluções eficientes para problemas complexos. Seja para análise de grandes volumes de dados, criação de algoritmos de aprendizado de máquina ou desenvolvimento de aplicações modernas, Python é uma ferramenta essencial para quem deseja atuar no mundo da ciência e da tecnologia.

3.1 Funções básicas

O projeto visa ensinar alunos novatos em Python, sem muito contato prévio com a linguagem. Dessa forma é necessário abordar funções bem básicas, como imprimir informações na tela, contabilizar elementos, usar loopings para repetição de tarefas e outras operações simples. Essas funções iniciais são essenciais para que os alunos adquiram uma base sólida para avançar para conceitos mais complexos. Para isso, foram produzidas videoaulas que serão disponibilizadas posteriormente.

3.2 Bibliotecas

A salvo de poucas funções mais específicas explicadas ao longo do código, a análise dos experimentos tem base primordial em 3 bibliotecas do python, são elas NumPy, Pandas e Matplotlib.

O NumPy é essencial para qualquer aluno que deseja trabalhar com computação científica e análise de dados. Ele oferece suporte para arrays multidimensionais e funções matemáticas eficientes, permitindo operações vetorizadas muito mais rápidas do que loops convencionais em Python. É fundamental para manipular grandes quantidades de dados numéricos de forma eficiente.

O Pandas é uma biblioteca voltada para a manipulação e análise de dados estruturados, como tabelas e séries temporais. Com seus DataFrames, facilita a leitura, organização, filtragem e agregação de dados, tornando a análise mais intuitiva. Aprender Pandas permite transformar as informações brutas obtidas no laboratório em insights úteis.

O Matplotlib é a principal biblioteca para visualização de dados em Python, permitindo criar gráficos personalizáveis de diversos tipos. Ele é crucial para representar padrões e tendências nos dados de forma clara e compreensível.

As funções destas bibliotecas utilizadas em algum ponto da análise estão listadas e esclarecidas no arquivo do colab a seguir. **Glossário de funções**

4 Leitura de dados

Para facilitar a análise sem a necessidade da instalação de quaisquer ferramenta ou ambiente de trabalho, todos os dados utilizados nos exemplos são copiados diretamente do google sheets. Dessa forma também não há necessidade de conectar o ambiente do colab ao drive. Copiando e colando uma tabela simples diretamente na célula, os dados aparecem em CSV no seguinte formato:

```
import io                    #exp1 - fractais
import pandas as pd
pd.read_csv(io.StringIO('''
"129,6",256
"86,94",128
"64,58",64
"46,84",32
"34,52",16
"24,4",8
"17,38",4
"13,3",2
"9,64",1
'''), header=None)
```

	0	1
0	129,6	256
1	86,94	128
2	64,58	64
3	46,84	32
4	34,52	16
5	24,4	8
6	17,38	4
7	13,3	2
8	9,64	1

4.1 Tabelas

Nota-se que os dados já aparecem com um código adicional, de forma que ao executa-lo é gerada uma tabela como a mostrada acima. Somente são necessários os dados, a parte entre aspas ("'..."'), definidos no código abaixo. Pelo formato CSV, para ler o arquivo são necessárias as bibliotecas pandas e io, de forma que os dados são traduzidos em arrays, separados por coluna.

```
import io                                    #exp1- fractais
import pandas as pd

```

```
dados = '''
"129,6",256
"86,94",128
...
"9,64",1
'''

# Leitura dos dados, ajustando separadores e formatos
df = pd.read_csv(io.StringIO(dados), header=None, sep=',', decimal=',')
df.columns = ['X', 'Y']  # Nomear as colunas para facilitar o uso

# Separar as colunas X e Y
x = df['X'].values
y = df['Y'].values
```

Dentro da função do pandas, vista com mais detalhes na aula referente à biblioteca, a função io.StringIO é utilizada para transformar os dados CSV em uma string. Embora já abordado, também vale lembrar que se o decimal estiver indicado com pontos ao invés de virgulas na tabela, basta alterar a variável deicmal da função.

Abaixo, são renomeadas as colunas como X e Y para melhor visualização. Estas são separadas (x e y) e finalmente transformadas em arrays do NumPy (.values), para facilitar o uso matemático.

No caso de mais colunas, como em gráficos com incertezas:

```
import io                                              #exp1- massa mola
import pandas as pd

dados='''
"0,0577","0,65","0,002","0,0001"
"0,0401","0,546","0,003","0,0001"
...
"0,1119","0,886","0,001","0,0001"
'''

# Leitura dos dados, ajustando separadores e formatos
df = pd.read_csv(io.StringIO(dados), header=None, decimal=',')
df.columns = ['X', 'Y','iY','iX']

# Separar as colunas X e Y
x = df['X'].values
y = df['Y'].values
iy = df['iY'].values
ix = df['iX'].values
```

4.2 Sequência de dados

No caso de uma histrograma, os os dados podem estar em uma ou multiplas linhas e/ou colunas (independe) de modo a serem traduzidos pelo código abaixo. Segue o exemplo com uma linha única:

```
import io
import pandas as pd
pd.read_csv(io.StringIO('''
"2,5","2,63","2,66","2,59","2,56","2,59","2,53",...,"2,53","2,62","2,56","2,37"
'''), header=None)
```

Código completo:

```
import matplotlib.pyplot as plt                          #exp1- queda livre (peletron)
import numpy as np
import io

# Dados para o histograma do google sheets (tudo em uma unica linha)
dados = '''"2,5","2,63","2,66","2,59","2,56",...,"2,69","2,56","2,53","2,62","2,56","2,37"
'''

# Usar pandas para ler os dados
df = pd.read_csv(io.StringIO(dados), header=None, decimal=',')
# Converter o DataFrame para uma lista unidimensional
dados_lista = df.values.flatten().tolist()

print(dados_lista)    #apenas para exemplo
```

Nesse caso, os dados serão trabalhados em formato de lista. A leitura é feita similarmente à versão anterior, excluindo apenas a necessidade do 'sep' pela falta de colunas.

Abaixo são transformados em lista por *.tolist()*, e a função *.flatten()* garante que independente da disposição dos dados na planilha todos serão lidos como uma sequência única, o que é necessário para a plotagem de um histograma.

4.3 Extra: exclusão de linhas

No caso de haver a nessecidade de exclusão de linhas da tabela por algum critério específico, a operação pode ser realizada com apenas uma linha de código. Seguem abaixo os 2 exemplos mais comuns. No caso de haverem linhas vazias na tabela, a leitura dos dados CSV devolverá "NaN", que significa ausente. Nesse caso a exclusão pode ser realizada pela segunte função do pandas, já discutida.

```
df.dropna(inplace=True)
```

No caso de algo ainda mais específico, pode ser realizada uma exclusão manual do dataframe. Segue o exemplo para uma linha de zeros.

```
df = df[(df['X'] != 0) | (df['Y'] != 0)]
```

Assim, você garante que todos os dados contabilizados serão não nulos. O código mostrado impõe a condição aos dois eixos, mas linhas inteiras podem ser excluídas com condições impostas a apenas um dos eixos. No caso de eliminar todas as linhas de eixo X nulo:

```
df = df[df['X'] != 0]
```

5 Gráficos

5.1 Plotagem dos dados

Para a plotagem de gráficos é importada a biblioteca matplotlib. Usualmente, os gráficos requeridos nas análises são gráficos de dispersão que consideram incertezas, o que objetiva o ajuste de dados a fim da obtenção de parâmetros diversos, a depender do experimento.

```
#exp3- curvas resistor/pilha
import matplotlib as plt
#plotagem
plt.errorbar(x, y, xerr=ix, yerr=iy, fmt='o',
     ecolor='red', capsize=2, label="Dados
    com incertezas")

#personalizando
plt.xlabel('corrente(A)')  #legendas
plt.ylabel('Tensao(V)')
plt.title('Pilha')    #titulo do grafico
plt.grid(True, linestyle='--', alpha=0.7)
plt.legend()   #mostrar legendas dos dados
    plotados
plt.show()   #mostrar grafico
```

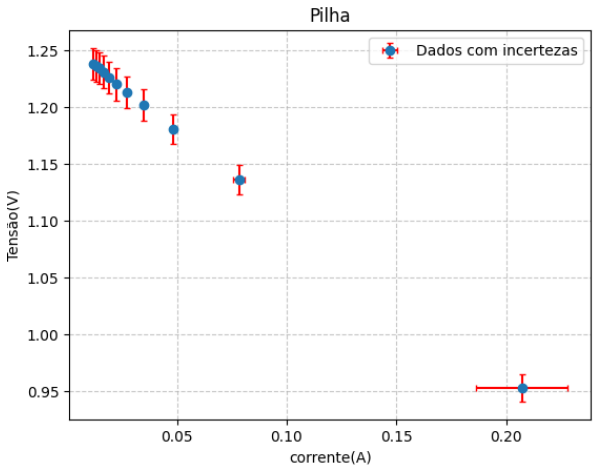


Figure 1: Gráfico de dispersão com incertezas

Acima está o código padrão utilizado nas análises exemplo para gráficos desse tipo. A função para a plotagem no caso em que se considera incertezas é plt.errorbar() , e a personalização pode ser complementada, alterando limites do gráfico, eixos e tamanhos de fonte, por exemplo (pode ser encontrado na seção do matplotlib em "bibliotecas base").

No caso de não serem consideradas incertezas a plotagem se dá por:

```
plt.scatter(x, y, color='blue', label='Dados originais')          #exp1- fractais
```

No caso de histogramas:

```
plt.hist(dados_lista, bins=10, edgecolor='black')        #exp1- queda livre(peletron)
plt.xlabel('Valores medidos (X)')
plt.ylabel('Numero de medidas (Frequencia)')
plt.title('Histograma das medidas')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

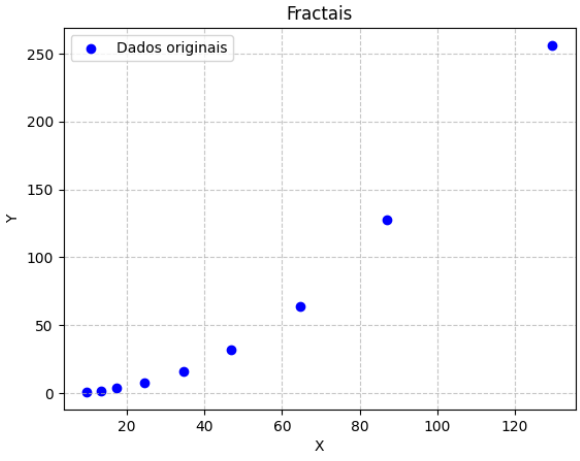


Figure 2: Gráfico de dispersão

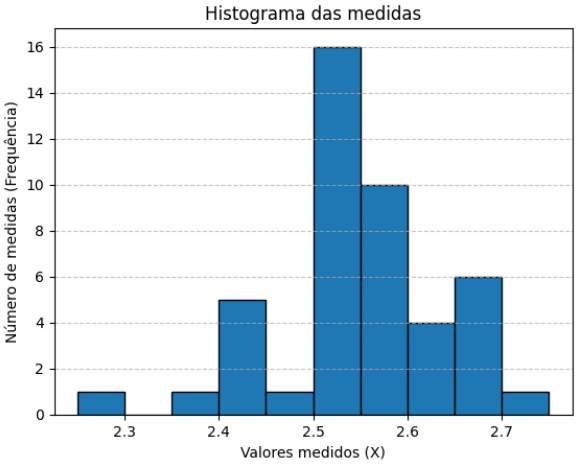


Figure 3: Histograma

5.2 Ajustes

Nos dois primeiros casos podem ser aplicados ajustes de modelos variados. Segue como exemplo do código utilizado uma aplicação de ajuste linear. Inicialmente, se define a função:

```
def funcao(x, a , b):
    return a*x+b
```

Para realizar o ajuste importamos da biblioteca scipy.optimise a função *curve_fit*(). Ela conta com a função previamente definida, os arrays referentes a cada eixo, a incerteza associada a y (sigma) e finalmente afirma que os valores de sigma são absolutos e não normalizados. O curve fit define duas variáveis. Primeiramente, os valores ajustados dos parametros, posteriormente reatribuidos às suas referentes incógnitas (nesse caso a e b). Já a covariância define uma matriz que informa como as incertezas no dados afetam os parâmetros ajustados, ou seja, contém as incertezas dos parametros.

```
from scipy.optimize import curve_fit

parametros, covariancia = curve_fit(funcao, x, y,sigma=iy, absolute_sigma=True)
a, b = parametros  # Coeficiente ajustado
```

Observação: no caso de não haver incertezas apenas exclua as variaveis sigma e $absolute_sigma$

Posteriormente deve ser plotada uma curva contínua (plt.plot), em que a função y é aplicada em um array criado com auxílio da função np.linspace (NumPy). Vale lembrar que quanto maior a quantidade de pontos mais suave é a curva gerada.

As incertezas dos parâmetros correspondem às raizes dos elementos da diagonal principal da matriz de covariância.

```
# Plotar os dados e a curva ajustada
x_ajuste = np.linspace(min(x), max(x), 1000)
y_ajuste = funcao(x_ajuste, a,b)
plt.plot(x_ajuste, y_ajuste, color='red',
    label=f'Ajuste: $y= {a:.2f}x + {b:.2f}$')

# Exibir os coeficientes
print(f"Coeficiente a: {a:.4f} +/- {np.sqrt(
    covariancia[0, 0]):.3f}")
print(f"Coeficiente b: {b:.4f} +/- {np.sqrt(
    covariancia[1, 1]):.3f}")
```

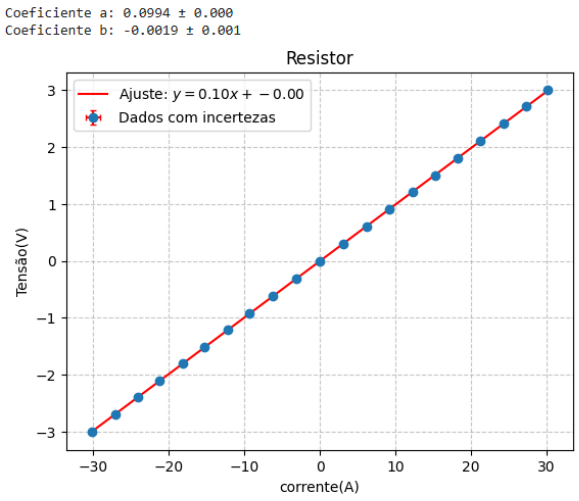
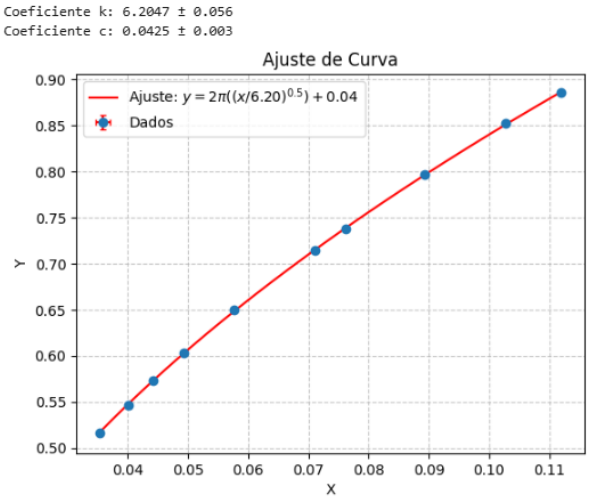


Figure 4: Ajuste linear

5.2.1 Aplicando para outros modelos

O mesmo código pode ser aplicado para diversos modelos. Além de definir a função correta para ajustar os dados, alterações devem ser feitas nos pontos onde são apresentados os parâmetros (no caso acima a e b). Nos dois pontos indicados devem ser substituidos os parâmetros da função linear apresentada pelos do modelo requerido, assim como no exemplo que segue.



Massa Mola:

```python
def funcao(x, k , c):
    return 2*np.pi*((x/k)**0.5) + c

parametros, covariancia = curve_fit(funcao, x
    , y, sigma=iy, absolute_sigma=True)
k, c = parametros

x_ajuste = np.linspace(min(x), max(x), 500)
y_ajuste = funcao(x_ajuste, k,c)

plt.plot(x_ajuste, y_ajuste, color='red',
    label=f'Ajuste: $y = 2\pi((x/{k:.2f})
    ^{{0.5}}) + {c:.2f}$')
```

Figure 5: Ajuste função do período de um sistema massa mola

6 Resíduos e χ^2

6.1 Gráfico de Resíduos

Uma importante ferramenta para vizualizar melhor a dispersão de pontos em relação ao ajuste realizado é o gráfico de resíduos. Basicamente, ele aponta a distância entre cada ponto e a curva ajustada no eixo y, o que permite uma análise qualitativa do modelo utilizado naquele caso. Os dois pontos a serem observados é se o intervalo das incertezas cobre a variação dos pontos em torno da curva (como um zoom do gráfico original) e se nesse novo gráfico a dispersão dos pontos aparenta seguir algum padrão. No segundo caso, a dispersão não aleatória geralmente indica que o modelo utilizado está equivocado.

Observação: este recurso não vale no caso de histogramas.

Para ilustrar, ajustamos os mesmos dados com um ajuste linear e um exponencial.

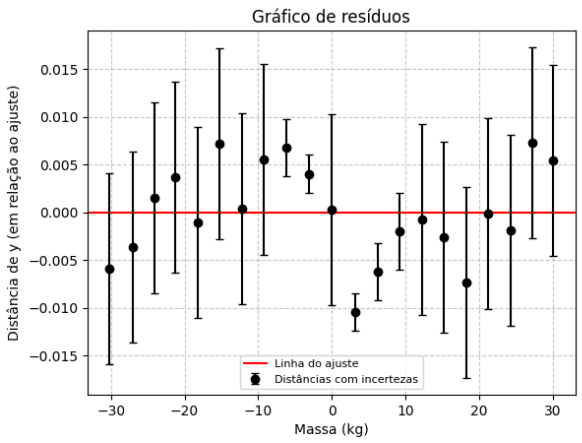


Figure 6: função: $ax + b$

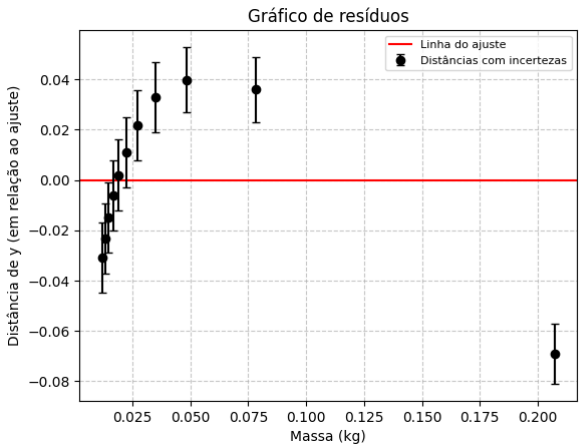


Figure 7: função: ax^b

É visível que os pontos estão melhor distribuidos ao redor da linha de ajuste quando se trata do ajuste linear. O segundo gráfico representa um padrão, os pontos formam uma curva que indica que a função exponencial não é a mais adequada neste caso.

A parte da dispersão é plotada como visto anteriormente, a salvo de que a função é aplicada diretamente nos valores de x, não à interesse em criar-se uma cuva contínua. Os valores reais de y são diminuidos dos encontrados e a variação é o que entra como coordenada do eixo y. No caso, o ajuste é representado como uma linha constante no eixo x. Segue o código para uma uma curva linear (Figure 5).

```
# ajuste                                                    #exp3- curvas resistor/pilha
parametros, _ = curve_fit(funcao, x, y)
a, b = parametros

#plotagem dos pontos
y_ajuste = funcao(x, a,b)
distancias = y - y_ajuste
plt.errorbar(x, distancias, yerr=iy, fmt='o', color='black', ecolor='black', capsize=3,
    label="Dados obtidos")
```

Observação: na primeira linha o underline apenas indica que o parâmetro covariancia (visto anteriomente) nesse caso não apresenta relevância e não será utilizado.

Para plotar a reta no eixo x que representa o ajuste é utilizada a função *plt.axhline* assim como no exemplo abaixo.

```
plt.axhline(0, color='red', linestyle='-', linewidth=1.5, label="Linha do ajuste")
plt.xlabel('Massa (kg)')
plt.ylabel('Distancia dos dados')
plt.title("Grafico de residuos")
plt.legend(fontsize=8)
plt.grid(True, linestyle='--', alpha=0.7)
plt.show()
```

Está incluído também a personalização básica deste gráfico, que seria a mesma figura para as duas plotagens. Em caso deste gráfico estar no mesmo código do gráfico orignal de dados deve ser adicionada a função plt.figure() no início do código.

6.1.1 Alteração do modelo de ajuste

Nesses casos, como não é necessário reescrever a função que já estará sendo utilizada para a plotagem do gráfico de dados, apenas é necessário se atentar aos pontos onde são apresentados os parâmetros. Os pontos de alteração são similares aos apresentados na aula anterior. Segue um exemplo para o sistema massa mola, também visto no ajuste.

```
parametros, _ = curve_fit(funcao, x, y)
k, c = parametros

y_ajuste = funcao(x, k, c)
distancias = y - y_ajuste
plt.errorbar(x, distancias, yerr=iy, fmt='o', color='black', ecolor='black', capsize=3,
    label= "Dados com incertezas")
```

6.2 χ^2 (qui-quadrado)

6.2.1 Cálculo do χ^2

O qui-quadrado é outro recurso utilizado para analizar a qualidade do ajuste. É uma comparação entre valores obtidos e valores esperados que segue a seguinte fórmula:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Em código, seguindo as definições feitas anteriormente:

```
chi2_valor = np.sum(((distancias) / iy) ** 2)
```

Basicamente, quanto mais próximo do número de graus de liberdade (N) for o valor obtido, mais coerente o ajuste realizado. Se o valor para χ^2 for muito maior que N o ajuste não é adequado, caso contrário, o modelo pode estar superajustado ou as incertezas superestimadas.

6.2.2 Graus de liberdade e valor crítico

Os graus de liberdade são a diferença entre o número de dados obtidos e o número de parâmetros da função de ajuste.

```
n_dados = len(y)  # Numero de pontos
n_parametros = 2  # Numero de parametros (a e b)
graus_de_liberdade = n_dados - n_parametros
```

É possível calcular um valor crítico, limite, para o qui-quadrado. Para isso utiliza-se a funçaõ chi2.ppf() da biblioteca scipy.stats . Ela tem como parâmetros N e o nível de significância alfa, geralmente 0.05. Para deixar claro, alfa não apresenta probabilidade de o ajuste estar correto ou não, os 5% representam o risco de afirmar que o ajuste faz sentido quando ele pode ser só sorte.

```
from scipy.stats import chi2
alfa = 0.05
valor_critico = chi2.ppf(1 - alfa, graus_de_liberdade)

#valores obtidos
print(f"Valor do \chi^2: {chi2_valor:.2f}")
print(f"Graus de liberdade: {graus_de_liberdade}")
print(f"Valor critico (alfa={alfa}): {valor_critico:.2f}")

#compara qui-quadrado e valor critico
if chi2_valor > valor_critico:
    print("os dados diferem significativamente do modelo.")
else:
    print("os dados nao diferem significativamente do modelo.")
```

7 Gráfico modelo ATUS

7.1 Plotagem: formatação de imagem

A partir dos princípios discutidos é possível plotar um gráfico único para dados e resíduos, que compartilha o eixo x. A imagem obtida assemelha-se à gerada no ATUS, software utilizado normalmente pelos alunos. Os princípios de plotagem seguem os mesmos, apenas dando maior atenção às dimensões da figura por meio de axes. A função *figsize* determina o tamanho e o formato da figura, neste caso, quadrada, enquanto *fig.add_axes* determina as dimensões de cada gráfico em altura, largura e posicionamento. Para As funções referentes ao plot basta substituir *plt* por *ax* e o índice do gráfico desejado. Segue abaixo um exemplo dessa adaptação para o experimento da Lei de Own:

```
# Criando a figura quadrada
fig = plt.figure(figsize=(6, 6))

# Primeiro subplot
ax1 = fig.add_axes([0, 0.25, 1, 0.75])  # [
    esquerda, baixo, largura, altura]
ax1.errorbar(x, y, xerr=ix, yerr=iy, fmt='o',
    color='black', ecolor='red', capsize=2,
    label="Dados com incertezas")
ax1.plot(x_ajuste, y_ajuste, color='red',
    label=f'Ajuste: $y= {a:.2f}x + {b:.2f}$')
ax1.legend()
ax1.set_title('Dados - Lei de Ohm')

# Segundo subplot
ax2 = fig.add_axes([0, 0, 1, 0.25])
ax2.axhline(0, color='red', linestyle='-',
    linewidth=1.5)
ax2.errorbar(x, distancias, yerr=iy, fmt='o',
     color='black', ecolor='red', capsize=3)
ax2.legend()
ax2.set_title('Residuos')
```

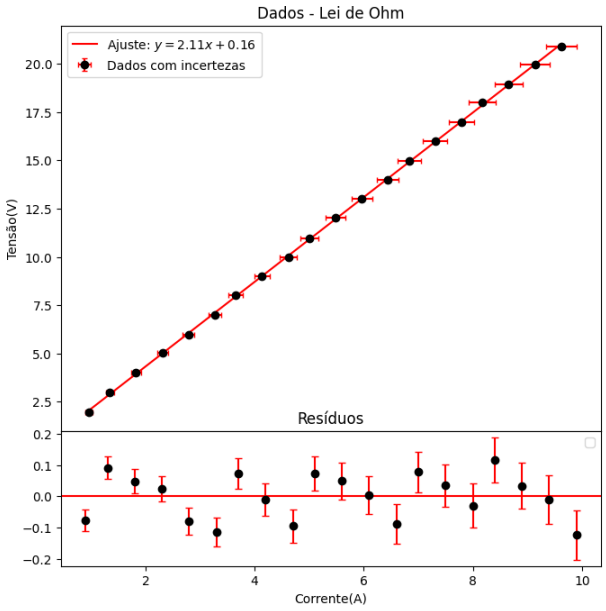


Figure 8: função: $ax + b$

7.2 Plataforma de análise

Como objetivo principal, esse programa é disponibilizado em um Notebook do Google Colab e permite que os alunos baixem uma cópia no próprio computador para que usem durante as disciplinas. O notebook permite, de forma prática, que o aluno analise os dados coletados e produza gráficos variados, ajustando parâmetros com valor e incerteza associados; além disso, retorna os valores de Chi² e NGL. Assim, esse programa pode ser usado ao invés de softwares, como o Analysis Tool for Undergraduate Students (ATUS) ou o WebRoot, já que possui, a princípio, as mesmas funções.

O arquivo colab contém os códigos completos para gráficos modelo ATUS e histogramas, pode acessa-lo **CLICANDO AQUI**.

8 Experimentos analizados

Para produção e aplicação teste do código de referência foram analisados alguns experimentos básicos que fazem partes de diferentes matérias experimentais do curso. Segue abaixo a lista dos mais relevantes, separados por disciplinas e seguidos dos links de suas reespectivas análises (todos podem ser consultados como arquivos do google colab).

8.1 Experimental 1- Dinâmica e cinemática

- Fractais **link**
- Queda livre - Peletron **link**
- Sistema massa mola **link**

8.2 Experimental 2- Movimento harmônico e ondas

- Lei de Ohm **link**
- Circuito RC **link**

8.3 Experimental 3- Eletrostática e circuitos

- Curvas características: Resistor e pilha **link**

9 Conclusão

Novamente, o Python é uma linguagem de programação que tem ampla aplicação dentro e fora STEM, de forma que é fundamental da formação de um físico, principalmente se pretende atuar na parte aplicada/experimental da academia ou no mercado de trabalho atrelado a tecnologia. Durante o curso, é obrigatório o treinamento dos alunos para utilizar essa linguagem através de matérias direcionadas à computação e, ademais, é estimulado o uso da mesma em outras disciplinas da própria física, a fim de aperfeiçoar a aplicação. O segundo caso se assemelha ao que ocorre em muitos laboratórios do instituto, um grande número de dados é analizado por meio do python, de modo que os alunos também precisam desse conhecimento para a realização de vários projetos de iniciação científica. É possível concluir que o contato com essa linguagem logo nas materias iniciais do curso é benéfico para os alunos.

É de costume utilizar a plataforma ATUS para análise de dados no laboratório, visto que os experimentos usualmente se limitam a captação de dados para dois eixos e plotagem direta. Há de ser dito que atende bem às necessidades dos alunos, porém o python permite maior personalização do seu gráfico, além de manipulação direta e simplificada dos dados que torna desnecessário o manejo de tabelas, como é geralmente feito.

Pensando na parte técnica, pode-se concluir que grande benefício é a possibilidade de simplificar a análise estatística quando necessária, principalmente em casos de looping, além de centralizar a mesma e a plotagem de gráficos. Ademais, a produção de histogramas que permite uma análise mais completa que o sheets. Porém, o maior ganho é a longo prazo, é o aprendizado que será muito requerido futuramente. Portanto, se houver investimento por parte dos alunos em treinar e se acostumar com a linguagem, reproduzindo um sistema para adquirir parâmetros que já conhecem, se familiarizando assim com funções básicas e as bibliotecas mais utilizadas, o processo pode trazer grandes facilidades para suas empreitadas futuras, sejam disciplinas ou projetos externos.