UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ (UTFPR)
PROJETO REA

JOE MICHAEL HIDEYUKI FURUYA TAKAHASSI

RECURSOS DE INTERACAO EDUCACIONAL DIGITAL APLICADOS A FISICA, QUIMICA, A MATEMATICA E AS ENGENHARIAS USANDO O SCILAB E O PYTHON

PROGRAMA DE APOIO AO DESENVOLVIMENTO DE RECURSOS EDUCACIONAIS ABERTOS NA GRADUAÇÃO DA UTFPR (ÁREAS TRANSVERSAIS)

CURITIBA

2021

JOE MICHAEL HIDEYUKI FURUYA TAKAHASSI

RECURSOS DE INTERACAO EDUCACIONAL DIGITAL APLICADOS A FISICA, QUIMICA, A MATEMATICA E AS ENGENHARIAS USANDO O SCILAB E O PYTHON

Apostila elaborada para o projeto de Recursos Educacionais Abertos descritos no edital 26/2021 e ofertada pela UTFPR, apresentado aos avaliadores da banca como requisito para a conclusão do projeto.

Orientador: Prof. Dr. Antônio Carlos Amaro de Faria Júnior

CURITIBA

2021

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

Esse material foi elaborado para o projeto ”Recursos de Interação Educacional Digital Aplicados à Física, Química, à Matemática e às Engenharias usando o Scilab e o Python”, ofertado pela UTFPR e pertencente ao Edital 26/2021. Para cada exemplo, foi exibido os respectivos códigos fontes assim como foi escrito uma explicação dos principais conceitos relacionados àquele tema.

1.2 OBJETIVOS

O objetivo do projeto é auxiliar o estudante no aprendizado de conceitos importantes tanto no campo da Física, Química, Matemática e Engenharias, através da elaboração de exemplos práticos no ambiente Jupyter Lab. Para isso, foi consultado referências bibliográficas para a elaboração da fundamentação teórica assim como foi utilizado algumas bibliotecas específicas do Python pertinentes para cada exemplo em questão.

2 EXEMPLOS

Nesta seção serão apresentados os exemplos práticos escolhidos para o projeto e feitos com o auxílio do Jupyter Lab, ambiente de desenvolvimento do Python.

2.1 REGRESSÃO LINEAR

2.1.1 O QUE É REGRESSÃO LINEAR?

Uma regressão linear nada mais é do que uma equação matemática utilizada para se estimar o valor de uma variável y, dados os valores de algumas outras variáveis x. Os modelos de regressão **simples** envolvem somente duas variáveis: uma independente x e uma dependente y. Ela é chamada **linear** poque a relação entre os parâmetros se dá por uma **função linear** do tipo:

$$F(x) = y = m \cdot x + c$$

Onde x é a variável independente, y é a variável dependente, m é o **coeficiente angular/declive** e c é o **coeficiente linear/intercepto**. O parâmetro c é o valor de $F(x)$ quando $x = 0$. O parâmetro m é a variação em $F(x)$ quando variamos x em 1 unidade.

O que pretendemos aqui é, a partir de um conjunto de dados (x, y), obter um modelo linear de função $F(x) = y$ que relacione de modo mais exato possível a relação entre as variáveis x e y. Para que possamos entender a regressão linear em sua integridade, duas concepções são bastante importantes: a de **Função de Custo** e **Gradiente Descendente**.

2.1.2 O QUE É FUNÇÃO DE CUSTO?

No contexto de otimização algorítmica, a função que queremos minimizar ou maximizar é denominada **função objetivo**. Quando queremos minimizá-la, ela é chamada **função de custo**. A função de custo computa a diferença entre o valor obtido em nosso modelo (y_{modelo}) e o valor real (y_{real}). **Entropia cruzada** e **Erro Quadrático Médio** são os tipos de funções de custo mais comuns. A fórmula para o Erro Quadrático Médio para n amostras de dados é:

$$E = \frac{1}{n}\sum_{i=0}^{n}(y_{i,real} - y_{i,modelo})^2 = \frac{1}{n}\sum_{i=0}^{n}(y_{i,real} - (m \cdot x_i + c))^2$$

Em sua essência, o conceito de função de custo é bastante simples: é um método para avaliar o quão bem o seu algoritmo modela o conjunto de dados em estudo. Se as predições estiverem totalmente erradas, a função de custo terá um valor maior. Se as predições estiverem corretas, a função de custo terá um valor menor. Na medida que mudados o nosso algoritmo, a função de custo irá nos dizer se estamos indo para o caminho correto.

2.1.3 GRADIENTE DESCENDENTE

Gradiente Descendente é um algoritmo de optimização usado para obter o mínimo de uma função diferenciável. Aqui, ela será utilizada para realizar o ajuste dos parâmetros m e c de forma iterativa com o objetivo de encontrar os valores para esses parâmetros que minimizem a função de custo o máximo possível. O Método do Gradiente se inicia atribuindo valores aleatórios para os parâmetros m e c, valores estes que irão melhorar gradualmente a cada iteração, dando um pequeno passo de cada vez até que a função de custo convirja para um mínimo. O tamanho dos passos é definido por um parâmetro denominado **taxa de aprendizado**.

Na aplicação do Método do Gradiente, utilizamos as derivadas parciais da função de custo em relação a m e em relação a c. O objetivo do método é, então, achar o mínimo global da função de custo a partir dessas derivadas parciais:

$$D_m = \frac{2}{n}\sum_{i=0}^{n}(y_{i,real} - (m \cdot x_i + c)) \cdot (-x_i) = \frac{-2}{n}\sum_{i=0}^{n}(y_{i,real} - y_{i,modelo})$$

$$D_c = \frac{-2}{n}\sum_{i=0}^{n}(y_{i,real} - y_{i,modelo})$$

2.1.4 REGRESSÃO LINEAR POR GRADIENTE DESCENDENTE NO PYTHON

Vamos modelar uma função $F(x)$ para um conjuntos de dados armazenados em um arquivo csv, utilizando o Erro Quadrático Médio e o Gradiente Descendente.

Primeiramente, vamos plotar o gráfico de dispersão das amostras para se ter uma ideia inicial da distribuição dos dados. Usaremos o conjunto de dados armazenados neste link: `https://github.com/Joe-Taka/REA/blob/main/C%C3%B3digos/Exemplo%201%20-%20Regress%C3%A3o%20Linear/data.csv`

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# Definindo um tamanho para a figura
plt.figure(figsize=(7,6))
# Pegando os valores do arquivo 'csv'
data = pd.read_csv('data.csv')
# Pegando a 1° coluna
X = data.iloc[:, 0]
# Pegando a 2° coluna
Y = data.iloc[:, 1]
# Gráfico de dispersão para as amostras
plt.scatter(X, Y)
plt.show()
```

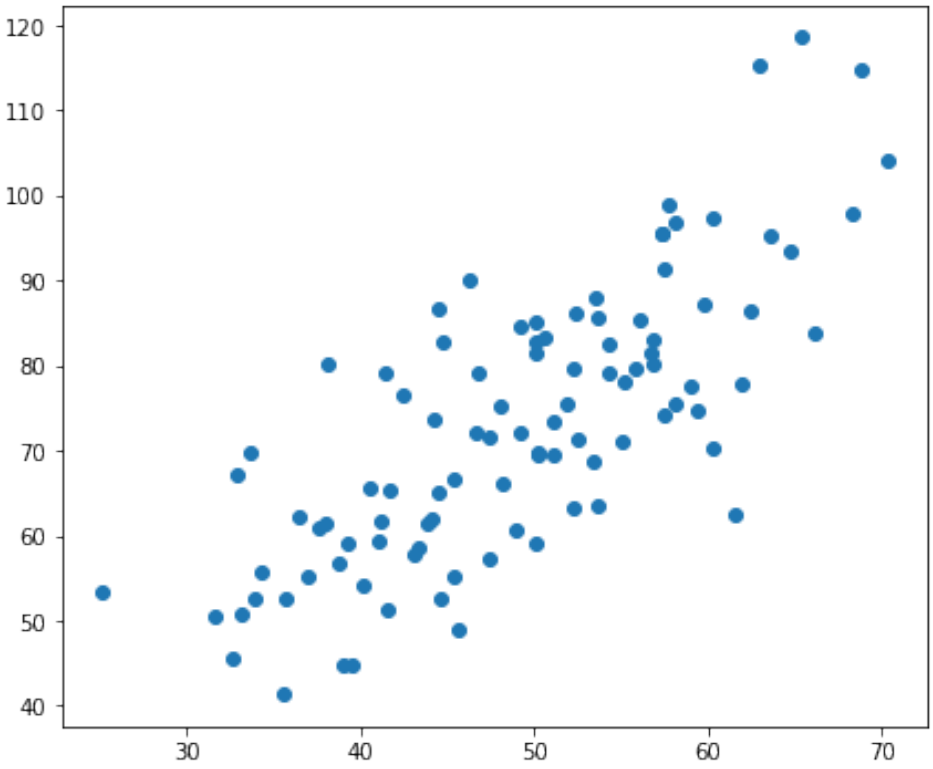

Figura 1: Gráfico de dispersão

Aplicando o **Gradiente Descendente**: começamos atribuindo valores nulos para os parâmetros m e c e iteramos as derivadas parciais da função de custo mil vezes, até encontrar os valores ótimos para os parâmetros:

```
# Suposições iniciais para os parâmetros
m = 0
c = 0
plt.figure(figsize=(7,6))
# Taxa de aprendizado
L = 0.0001
# Número de iterações para o Gradiente Descendente
epochs = 1000
n = float(len(X)) # Número de elementos de X
# Aplicando a iteração do Método Gradiente
for i in range(epochs):
        Y_pred = m*X + c  # Valor atual suposto para Y
        D_m = (-2/n) * sum(X * (Y - Y_pred))
        D_c = (-2/n) * sum(Y - Y_pred)
        m = m - L * D_m  # Atualizando m
        c = c - L * D_c  # Atualizando c
print (f"Valor para m: {m:.2f}; Valor para c: {c:.2f}")
# Valor para m: 1.48; Valor para c: 0.10
Y_pred = m*X + c
plt.scatter(X, Y)
plt.plot([min(X), max(X)], [min(Y_pred),
max(Y_pred)], color='red')
plt.show()
```

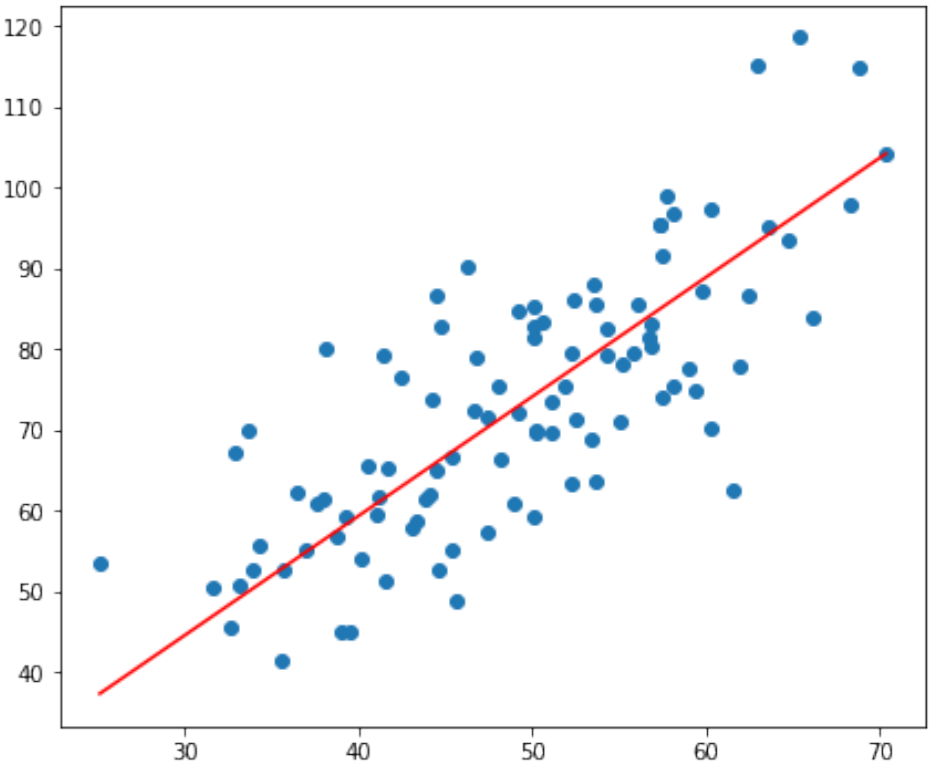

Figura 2: Regressão Linear

Para o conjunto de dados fornecidos em nosso exemplo, a função do nosso modelo fica, então:

$$F(x) = 1,48 \cdot x + 0,10$$

2.1.5 APLICAÇÕES DA REGRESSÃO LINEAR

- Aprendizado de máquina, como uma ferramenta de predição.
- Em modelo de negócios, pode ser utilizado para gerar insights à respeito do comportamento do consumidor, de modo a entender os fatores que possam influenciar na lucratividade do negócio. Por exemplo, se as vendas de uma empresa aumentaram constantemente todos os meses nos últimos anos, ao se realizar uma análise linear dos dados de vendas com as vendas mensais, a empresa é capaz de prever as vendas nos meses futuros.

- Química analítica, principalmente na calibração de dados para gerar as chamadas **curvas de calibração**.
- Nas ciências dos materiais, como uma forma de prever as propriedades de certos materiais.
- Em finanças, como uma ferramenta para traçar as curvas de tendências de ativos.

2.2 TEOREMA DE BAYES

2.2.1 O QUE É O TEOREMA DE BAYES?

O teorema de Bayes fornece um modo de calcular a probabilidade de uma hipótese baseado na probabilidade de outras hipóteses **a priori**.

2.2.2 FÓRMULA DO TEOREMA DE BAYES

$$P(A|B) = \frac{P(A) \cdot P(B|A)}{P(B)}$$

- $P(A|B)$: probabilidade do evento 'A' ocorrer dado que 'B' ocorreu
- $P(A)$: probabilidade de 'A' ocorrer
- $P(B|A)$: probabilidade do evento 'B' ocorrer dado que 'A' ocorreu
- $P(B)$: probabilidade de 'B' ocorrer

Portanto, o Teorema de Bayes afirma que a probabilidade posterior do evento A (ou seja, a probabilidade do evento A dado que o evento B ocorreu) é igual à probabilidade contrária P(B | A) multiplicada pela probabilidade de A e dividido pela probabilidade de B.

2.2.3 TEOREMA DE BAYES UTILIZANDO SCIKIT-LEARN NO PYTHON

Vamos utilizar um conjunto de dados que incluem 4 observações sobre um determinado momento do dia (clima - claro, chuvoso ou geada; feriado ou dia útil; período -

manhã, horário de almoço ou noite; houve ou não congestionamento) e, a partir desses dados base, tentaremos prever a probabilidade de acontecer um congestionamento utilizando valores fornecidos apenas das 3 condições iniciais. Para isso, usaremos a biblioteca **Scikit-Learn** do Python.

```
def getTempo():
return ['Claro', 'Claro', 'Claro', 'Claro', 'Claro',
'Claro', 'Chuvoso', 'Chuvoso', 'Chuvoso', 'Chuvoso',
'Chuvoso', 'Chuvoso', 'Geada', 'Geada', 'Geada', 'Geada',
'Geada', 'Geada']

def getDiaSem():
return ['Útil', 'Útil', 'Útil',
'Feriado', 'Feriado', 'Feriado',
'Útil', 'Útil', 'Útil',
'Feriado', 'Feriado', 'Feriado',
'Útil', 'Útil', 'Útil',
'Feriado', 'Feriado', 'Feriado']

def getHorario():
return ['Manhã', 'Almoço', 'Noite',
'Manhã', 'Almoço', 'Noite',
'Manhã', 'Almoço', 'Noite',
'Manhã', 'Almoço', 'Noite',
'Manhã', 'Almoço', 'Noite',
'Manhã', 'Almoço', 'Noite']

def getCongest():
return ['Sim', 'Não', 'Sim',
'Não', 'Não', 'Não',
'Sim', 'Sim', 'Sim',
'Não', 'Não', 'Não',
'Sim', 'Sim', 'Sim',
'Sim', 'Não', 'Sim']

Tempo = getTempo()
diaDaSem = getDiaSem()
Horario = getHorario()
Congest = getCongest()

pd.DataFrame(zip(Tempo,diaDaSem,Horario,Congest),
columns=['Tempo','Dia da semana',
'Horário','Congestionamento'])
```

	Tempo	Dia da semana	Horário	Congestionamento
0	Claro	Útil	Manhã	Sim
1	Claro	Útil	Almoço	Não
2	Claro	Útil	Noite	Sim
3	Claro	Feriado	Manhã	Não
4	Claro	Feriado	Almoço	Não
5	Claro	Feriado	Noite	Não
6	Chuvoso	Útil	Manhã	Sim
7	Chuvoso	Útil	Almoço	Sim
8	Chuvoso	Útil	Noite	Sim
9	Chuvoso	Feriado	Manhã	Não
10	Chuvoso	Feriado	Almoço	Não
11	Chuvoso	Feriado	Noite	Não
12	Geada	Útil	Manhã	Sim
13	Geada	Útil	Almoço	Sim
14	Geada	Útil	Noite	Sim
15	Geada	Feriado	Manhã	Sim
16	Geada	Feriado	Almoço	Não
17	Geada	Feriado	Noite	Sim

Tabela 1: Condições do dia

2.2.3.1 PRÉ-PROCESSAMENTO DOS DADOS

Vamos utilizar o comando 'sklearn.preprocessing.LabelEncoder' para **normalizar** os dados, ou seja, alterar os valores das colunas numéricas no conjunto de dados para uma escala comum, sem distorcer os intervalos de valores ou provocar perda de informações. No exemplo analisado, os valores do tipo texto serão transformados em valores numéricos. Ainda, usamos o método **fit_transform()** para evitar possíveis erros que poderiam ser provocados por valores nulos, inexistentes ou indefinidos no nosso conjunto de dados. Por fim, organizamos os dados relevantes (**features**) para o nosso exemplo em uma única lista.

```
labelEncoder = preprocessing.LabelEncoder()
# Normalizando e aplicando o 'fit_transform'
normTempo = labelEncoder.fit_transform(Tempo)
normDiaDaSem = labelEncoder.fit_transform(diaDaSem)
normHorario = labelEncoder.fit_transform(Horario)
normCongest = labelEncoder.fit_transform(Congest)
print("-----------------normalização-----------------")
print(normTempo)
print(normDiaDaSem)
print(normHorario)
print(normCongest)
# Organizando os dados relevantes (features)
features = []
for i in range(len(normTempo)):
        features.append([normTempo[i], normDiaDaSem[i],
        normHorario[i]])
print("-----------------features-----------------")
print(features)
# Aplicando o Método de Bayes
modelo = GaussianNB()
# Treinando o modelo
modelo.fit(features, normCongest)
```

2.2.3.2 FAZENDO AS PREDIÇÕES

```
# ["Geada", "Útil", "Manhã"]
print(modelo.predict([[2, 1, 1]]))
# resultado = [1]
# ["Claro", "Feriado", "Almoço"]
print (modelo.predict([[1, 0, 0]]))
# resultado = [0]
```

Portanto, para as condiçoes [”Geada”, ”Útil”, ”Manhã”] por exemplo, há uma grande probabilidade de ocorrer congestionamento. Para as condiçoes [”Claro”, ”Feriado”, ”Almoço”], a probabilidade de ocorrer um congestionamento é baixa.

2.2.4 APLICAÇÕES DO TEOREMA DE BAYES

- Aprendizado de máquinas, principalmente nas classificações de dados.
- Pode ser utilizado na química para avaliar a densidade de probabilidade da composição química de um sistema em termos das densidades de probabilidade das abundâncias das diferentes espécies químicas.
- Pode ser utilizado para prever a qualidade de água com base na concentração de toxinas e outros poluentes que possam ultrapassamr os limites numéricos do padrão de qualidade da água.
- Em engenharia estrutural, auxiliando o engenheiro a determinar se a estrutura é segura ou não, com base nos valores de reabilidade.

2.3 TRANSFORMADA DE FOURIER

2.3.1 O QUE É A TRANSFORMADA DE FOURIER?

A transformada de Fourier é uma transformação matemática que ”quebra”uma forma de onda (função ou sinal) em uma soma de funções periódicas. A análise de Fourier converte um sinal do seu domínio original para uma representação no domínio da frequência.

Jean-Baptiste Joseph Fourier (Auxerre, 21 de março de 1768 — Paris, 16 de maio de 1830) foi o matemático e físico francês que descobriu que todo sinal pode ser descrito como uma superposição de senóides complexas.

2.3.2 SINAIS

Os sinais traduzem a evolução de uma grandeza ao longo do tempo ou espaço e podem ser classificados segundo os critérios:

- Continuidade: sinais contínuos ou discretos
- Periodicidade: sinais periódicos ou aperiódicos

2.3.2.1 SINAIS CONTÍNUOS

Um sinal diz-se contínuo se o seu domínio for IR ou um intervalo contínuo de IR. Assim:

$$x : IR \rightarrow IR$$

$$x : [a,b] \rightarrow IR$$

- Sinais contínuos são sinais que não possuem espaços distinguíveis entre os seus valores. - Advém de grandezas que podem ser medidas.

2.3.2.2 SINAIS DISCRETOS

- Sinais discretos são sinais que possuem espaços entre os seus valores.
- Advém de grandezas que são contadas.
- Só os sinais discretos podem ser armazenados e processados em computadores digitais.
- Pode-se converter um sinal contínuo num sinal discreto através da coleção de amostras do sinal contínuo.

2.3.2.3 SINAIS PERIÓDICOS

Um sinal discreto x(n) é periódico com período $N \in IN$ se:

$$x(n+N) = x(n), \forall\ n\ \in Z$$

2.3.3 TIPOS DE ABORDAGENS

De acordo com o tipo de sinal que estamos lidando, existem 4 diferentes tipos de abordagem:

- Sinais contínuos e aperiódicos: Transformada de Fourier
- Sinais contínuos e periódicos: Série de Fourier
- Sinais discretos e aperiódicos: Transformada de Fourier de Tempo Discreto (TFTD)
- Sinais discretos e periódicos: Transformada Discreta de Fourier (TDF)

2.3.4 FÓRMULA PARA TRANSFORMADA DISCRETA DE FOURIER (DFT)

Para um sequência finita de valores complexos ou reais x_n obtidos pela amostragem de N valores de um sinal contínuo nos instantes $f(0), f(1), f(2), ...f(k), ...f(N-1)$, a Fórmula para a **Transformada Discreta de Fourier** é:

$$X_k = \sum_{n=0}^{N-1} x_n \cdot e^{-j2\pi \frac{kn}{N}}$$

Sendo x_n a amplitude (real ou complexo) de um sinal periódico em um instante de tempo t. Tais valores são obtidos por amostragem de uma função periódica contínua. O resultado da fórmula é uma série de números complexos no domínio da frequência.

O TDF se tornou bastante utilizado em computação numérica devido ao algoritmo chamado **Transformada Rápida de Fourier**, que é um método bastante eficiente para se calcular a Transformada Discreta de Fourier (DFT) e a sua inversa.

Vamos calcular a Transformada Discreta de Fourier pelo algoritmo rápido usando a função **fft** da biblioteca Scipy do Python.

2.3.5 TRANSFORMADA DISCRETA DE FOURIER (DFT) NO PYTHON

Supondo 2 ondas senoidais de frequências 2Hz e 3Hz. Se somarmos ambas as ondas obteríamos uma onda de forma um pouco mais complicada. Vamos gerar essas 2 ondas e

somá-las utilizando o código abaixo:

```
import numpy as np
from matplotlib import pyplot as plt
SAMPLE_RATE = 50 # Hertz
tempo = 10  # Segundos
tempoTicks = [x for x in range(tempo+1)]
f, (ax1, ax2, ax3) = plt.subplots(3, 1,
sharex=False, figsize=(15,8))
# ajustando a distância vertical entre os plots
f.subplots_adjust(hspace=0.3)
def ondaSeno(freq, sample_rate, tempo, amp = 1):
        # gera 'sample_rate * tempo' amostras
        x = np.linspace(0, tempo, sample_rate * tempo,
        endpoint=False)
        # multiplica cada amostra pela frequência
        frequencias = x * freq
        # np.sin(ângulo em radianos)
        y = amp*np.sin((2 * np.pi) * frequencias)
        return x, y
ax1.set_title("Função seno (2Hz)")
ax2.set_title("Função seno (3Hz)")
ax3.set_title("Soma das funções senoidais 2Hz e 3Hz")
# Gerando uma onda senoidal de 2Hz
x, y = ondaSeno(2, SAMPLE_RATE, tempo, amp = 3)
# Gerando uma onda senoidal de 3Hz
x1, y1 = ondaSeno(3, SAMPLE_RATE, tempo)
ax1.grid(True, axis = 'x', alpha = 0.7 ,
color = 'black', which='both')
ax2.grid(True, axis = 'x', alpha = 0.7 ,
color = 'black', which='both')
ax3.grid(True, axis = 'x', alpha = 0.7 ,
color = 'black', which='both')
```

Continuação...

```
plt.xticks([0,1,2,3,4,5,6,7,8,9,10])
ax1.set_xticks(tempoTicks)
ax2.set_xticks(tempoTicks)
ax3.set_xticks(tempoTicks)
ax1.axhline(y=0, color='black')
ax2.axhline(y=0, color='black')
ax3.axhline(y=0, color='black')
ysoma = y + y1
ax1.plot(x, y, 'tab:red')
ax2.plot(x1, y1, 'tab:blue')
ax3.plot(x1, ysoma, 'tab:orange')
plt.show()
```

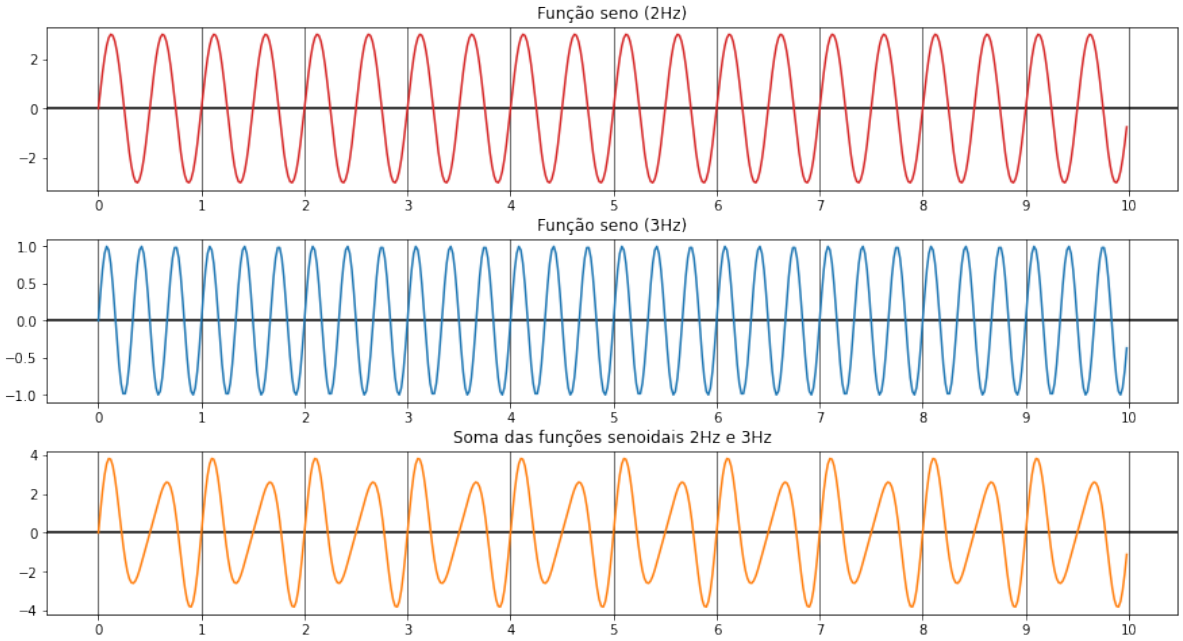

Figura 3: Soma de senoides

A função **fft** aplicado no sinal resultante retornará uma série de números complexos. Utilizaremos os valores absolutos desses números complexos na plotagem do gráfico. Ainda,

esse plot exibirá picos de frequência que nos indicará as frequências dos componentes originais que deu origem ao sinal resultante.

No exemplo abaixo, os picos de frequências estão em $2Hz$ e $3Hz$.

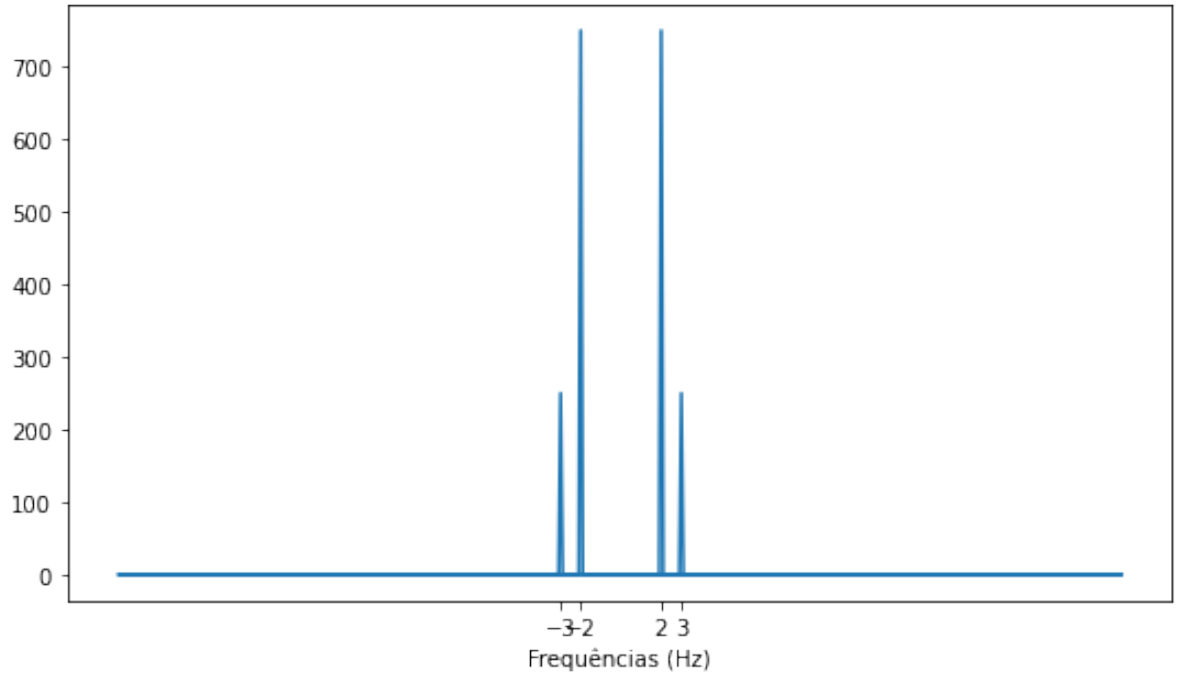

Figura 4: Picos de frequência

2.3.5.1 EXEMPLO: FILTRANDO RUÍDOS

Vamos considerar um exemplo hipotético de um sinal dotado de ruídos e que originalmente veio da soma de dois sinais de frequências $50Hz$ e $120Hz$.

O sinal ”limpo”sem ruídos é representado em cor vermelha, ao passo que o sinal com ruídos está na cor azul ciano.

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

f, ax = plt.subplots(figsize=(12,5))

dt = 0.001

# vetor tempo de 1000 elementos, indo de 0 a 1
t = np.arange(0,1,dt)

# criando e somando senóides de frequência 50 e 120
f = np.sin((2*np.pi)*50*t) + np.sin((2*np.pi)*120*t)

# sinal resultante sem ruído
f_clean = f

# adicionando ruídos à soma
f = f + 2.5*np.random.randn(len(t))

# 1000
# print(len(f))

ax.plot(t,f, color='c', label = 'Sinal com ruídos')
ax.plot(t,f_clean, color='red', label = 'Sinal sem ruídos')
# setando um limite para o eixo x
plt.xlim(t[0],t[-1])
plt.legend()
plt.show()
```

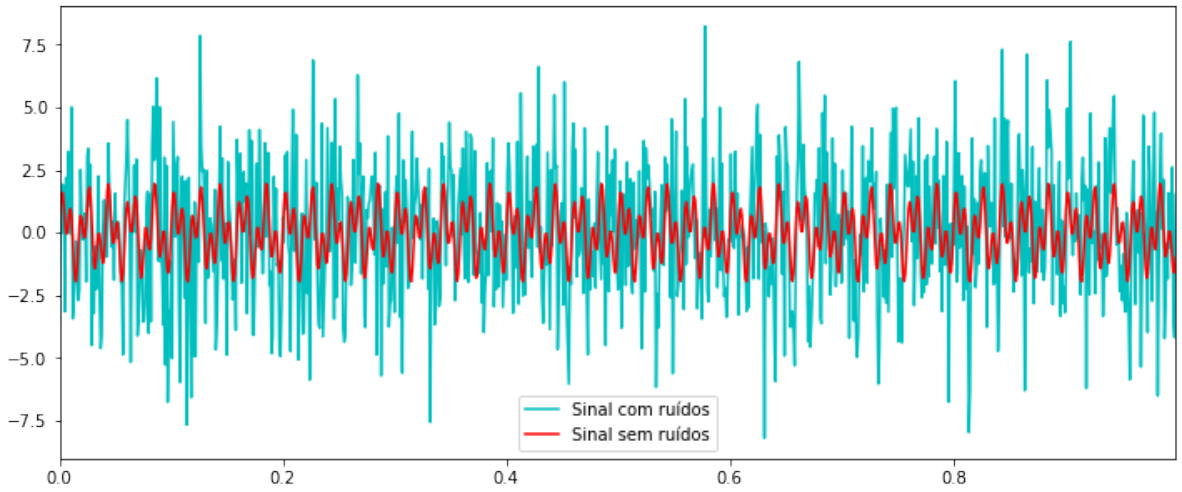

Figura 5: Sinal com e sem ruído

Vamos aplicar a **Transformada Rápida de Fourier** no sinal com ruído, plotar o resultado e trabalhar os dados de modo a eliminar a metade espelhada. Uma vez eliminada, obteremos um plot que nos exibirá as frequências correspondentes aos ruídos (frequências menores) e as frequências dos sinais componentes que deu origem ao sinal que queremos (picos de frequências). Aplicaremos, então, um **filtro** de modo a selecionar apenas as frequências maiores que $100Hz$ e eliminar os ruídos. Essas frequências filtradas serão submetidas, então, à **Transformada Discreta Inversa de Fourier** com o uso da função **ifft** para obtermos, por fim, o sinal limpo sem ruídos.

```
# 1000
n = len(t)

# Transformada discreta do sinal com ruído
# O resultado é um array de números complexos
fhat = np.fft.fft(f,n)

# Multiplicando cada número complexo pelo seu conjugado
# PSD é um vetor de números reais
PSD = fhat * np.conj(fhat) / n

PSD = PSD.real

# print(PSD[:10])
# 1000
# print(len(PSD))

freq = (1/(dt*n)) * np.arange(n)
# [0. 1. 2. 3. 4. 5. 6. 7. 8. 9.]
# print(freq[:10])
# 1000
# print(len(freq))

L  = np.arange(1, np.floor(n/2), dtype = 'int')
# 500
# print(np.floor(n/2))
# [490 491 492 493 494 495 496 497 498 499]
# print(L[-10:])
# 499
# print(len(L))
```

Continuação...

```
fig, axs = plt.subplots(4,1, figsize=(15,10))

# ajustando a distância vertical entre os plots
fig.subplots_adjust(hspace=0.6)

# Sinal com ruído e sem ruído
plt.sca(axs[0])
plt.plot(t,f, color = 'c', label = 'Sinal obtido')
plt.plot(t,f_clean, color='red',  label = 'Sinal desejado')
plt.xlim(t[0], t[-1])
plt.legend()

plt.sca(axs[1])
axs[1].set_xticks([0.05,0.12])
axs[1].set_xlabel("Frequências (Hz)")
plt.plot(t,np.abs(fhat), color = 'blue',
label = 'Transformada discreta espelhada')
# plt.xlim(freq[L[0]], freq[L[-1]])
plt.legend()

axs[2].set_xticks([50,120])
axs[2].set_yticks([100])
axs[2].grid(True, axis = 'y', alpha = 1 ,
color = 'black', which='both')
axs[2].set_xlabel("Frequências (Hz)")
```

Continuação...

```
plt.sca(axs[2])
plt.plot(freq[L],PSD[L], color = 'blue',
label = 'Transformada discreta')
plt.xlim(freq[L[0]], freq[L[-1]])
plt.legend()

axs[2].fill_between(freq[L], 100, 400,
where=np.abs(PSD[L]) > 0j, facecolor='green', alpha=0.5)

# indices: vetor de valores booleanos
# vamos filtrar os ruídos e eliminar
# aqueles com valores menores que 100
indices = PSD > 100
PSDclean = PSD + indices
# print(PSD[:51])
# print(PSDclean[:51])
fhat = indices * fhat
# Transformada inversa dos valores filtrados
ffilt = np.fft.ifft(fhat)
axs[3].grid(True, axis = 'x', alpha = 1 ,
color = 'black', which='both')
plt.sca(axs[3])
plt.plot(t, ffilt, color = 'red',
label = 'Sinal filtrado')
plt.xlim(t[0], t[-1])
axs[3].set_xlabel("Tempo (s)")
plt.legend()
plt.show()
```

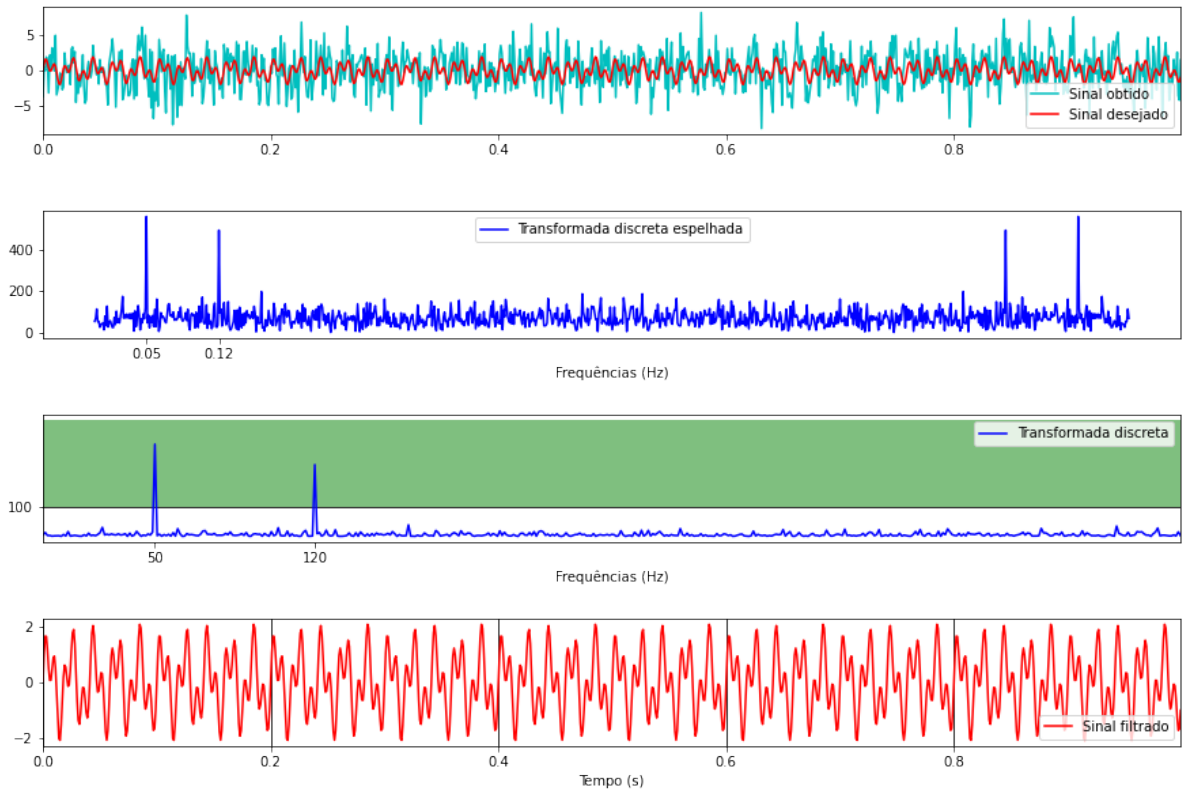

Figura 6: Sinal filtrado

Podemos ver que o sinal filtrado é exatamente igual ao sinal original sem os ruídos:

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

f, (ax, ax2) = plt.subplots(2,1,figsize=(12,5))
dt = 0.0001
t = np.arange(0,0.1,dt)
f = np.sin((2*np.pi)*50*t) + np.sin((2*np.pi)*120*t)
n = len(t)
fhat = np.fft.fft(f,n)
PSD = fhat * np.conj(fhat) / n
PSD = PSD.real
freq = (1/(dt*n)) * np.arange(n)
indices = PSD > 100
PSDclean = PSD + indices
fhat = indices * fhat
ffilt = np.fft.ifft(fhat)
L  = np.arange(1, np.floor(n/2), dtype = 'int')
ax.plot(t,f, color='red', label = 'Sinal original')
ax.legend()
ax2.plot(t, ffilt, color = 'red',
label = 'Sinal obtido a partir do sinal com ruído')
ax2.legend()
plt.show()
```

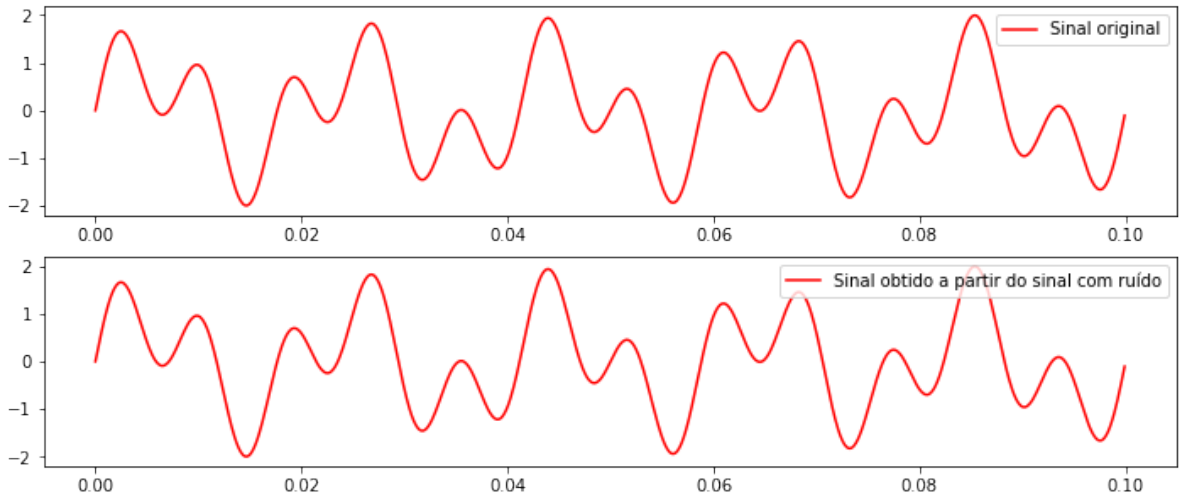

Figura 7: Comparação com o sinal original

2.3.6 APLICAÇÕES DA TRANSFORMADA DE FOURIER

- Processamento digital de sinais para a resolução de equações diferenciais parciais.
- Em fones de ouvido com cancelamento de ruído. Ela converte o sinal original em componentes espectrais individuais e elimina os componentes indesejados.
- Em processamento de imagens, sendo utilizado para decompor uma imagem em seus componentes seno e cosseno.
- Na medicina, na técnica de Imagem por Ressonância Magnética.
- Em óptica, no estudo da difração da luz quando ela passa por fendas estreitas.
- Na geologia e exploração de petróleo, na análise de dados sísmicos.

2.4 DISTRIBUIÇÃO GAUSSIANA

2.4.1 O QUE É A DISTRIBUIÇÃO GAUSSIANA?

A **distribuição Gaussiana** ou **distribuição normal** é uma das distribuições mais importantes em razão da sua enorme presença nos mais variados campos do conhecimento. Se trata de uma curva de distribuição simétrica em torno do seu ponto médio. Ela possui a peculiaridade de ter a média, mediana e moda dos dados com o mesmo valor.

2.4.2 VARIÁVEL ALEATÓRIA]

Variável aleatória é um valor retirado de uma distribuição estatística e que não possui um valor fixo. O seu valor depende de fatores aleatórios. Ex: o resultado do lançamento de um dado pode dar qualquer número entre 1 e 6.

Uma variável aleatória que pode assumir apenas um número finito ou uma sequência infinita enumerável de valores é considerada **discreta**. Aquele que pode assumir qualquer valor no intervalo dos números reais é considerado **contínuo**.

Uma **variável aleatória contínua** é uma variável aleatória que pode tomar qualquer valor numérico em um determinado intervalo ou coleção de intervalos (geralmente do conjunto dos números reais). Ex: uma variável aleatória que mede o tempo que leva para algo ser feito é contínua, pois ela pode assumir infinitos valores.

Como existe um número infinito de valores em qualquer intervalo, não faz sentido falar sobre a probabilidade da variável aleatória assumir um valor específico. Em vez disso, é considerada a probabilidade de uma variável aleatória contínua estar dentro de um determinado intervalo. É a chamada **Função Densidade de Probabilidade**.

2.4.3 FUNÇÃO DENSIDADE DE PROBABILIDADE

É a função $f(x)$ de uma variável aleatória contínua cuja integral em um intervalo dá a probabilidade de que o valor da variável esteja dentro desse mesmo intervalo. Ou seja, **densidade de probabilidade** não é probabilidade. Somente quando a função for integrada entre dois limites é que ela produzirá uma probabilidade, sendo equivalente à área sob a curva da função densidade de probabilidade entre esses dois limites:

$$\int_a^b f(x)dx = P(a \leq x \leq b)$$

As funções densidade de probabilidade devem sempre obedecer à esses 2 requisitos:

- $f(x)$ deve ser não negativo para cada valor da variável aleatória: $f(x) > 0$
- A área total sob a curva de probabilidade vale sempre 1: $\int_{-\infty}^{\infty} f(x)dx = 1$

A forma da função de densidade de probabilidade em todo o seu domínio é chamada de **distribuição de probabilidade**. A distribuição de probabilidade mais importante é a chamada **distribuição normal**, também chamado de **curva em forma de sino** devido à sua forma característica.

2.4.4 DISTRIBUIÇÃO NORMAL OU GASSIANA

A distribuição normal é a distribuição de probabilidade mais importante no estudo da estatística devido à sua presença em muitos fenômenos naturais, por exemplo: a altura das pessoas, pressão arterial, erro de medição e pontuações de QI seguem a distribuição normal.

Como já foi dito, a probabilidade de uma observação assumir um valor entre dois pontos quaisquer é igual à área sob a curva da densidade de probabilidade compreendida entre esses dois pontos. No caso de uma curva de distribuição normal teórica, a regra é:

- $68,26\%$ da população ou amostra está dentro de uma região que varia de $\mu \pm \sigma$
- $95,44\%$ da população ou amostra está dentro de uma região que varia de $\mu \pm 2\sigma$
- $99,72\%$ da população ou amostra está dentro de uma região que varia de $\mu \pm 3\sigma$
- $99,99\%$ da população ou amostra está dentro de uma região que varia de $\mu \pm 4\sigma$

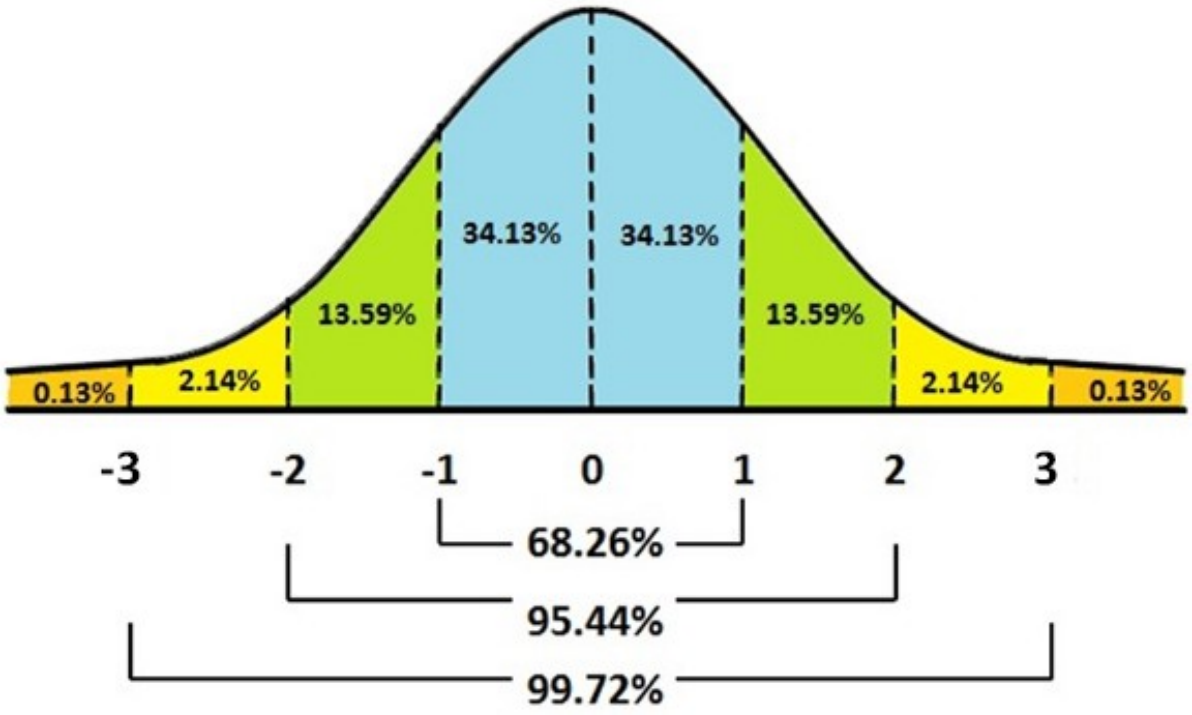

Figura 8: Distribuição normal

2.4.5 DENSIDADE PARA A DISTRIBUIÇÃO NORMAL

Uma variável aleatória contínua X tem distribuição normal se sua função densidade de probabilidade for dada por:

$$p(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

Onde σ é o **desvio padrão** da distribuição e μ é a **média**.

2.4.6 DESVIO PADRÃO

Comumente representado pelo símbolo σ, é uma medida de dispersão em torno da média populacional ou amostral de uma variável aleatória. Quanto maior o seu valor maior a ampla de valores na qual os dados estão espalhados. Um baixo desvio padrão indica que os pontos dos dados tendem a estar próximos da média.

2.4.7 FÓRMULA PARA O DESVIO PADRÃO

O desvio padrão populacional ou amostral é a raiz quadrada da variância populacional ou amostral correspondente.

- Quando o conjunto de dados é a população:

$$\sigma = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2}$$

- Quando o conjunto de dados é uma amostra:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2}$$

2.4.8 DISTRIBUIÇÃO NORMAL NO PYTHON

Vamos traçar a curva de uma **distribuição normal padronizada**, que é uma curva normal de média $\mu = 0$ e desvio padrão $\sigma = 1$.

```
from scipy.stats import norm
import matplotlib.pyplot as plt
import numpy as np

fig, ax = plt.subplots(figsize = (8,5))

# definindo o domínio
dom = np.linspace(-2,2,1000)

plt.plot(dom, norm.pdf(dom, loc = 0 , scale = 1))
plt.title("Distribuição Normal Padronizada")
plt.xlabel("Valor")
plt.ylabel("Densidade")
ax.grid(True)
plt.show()
```

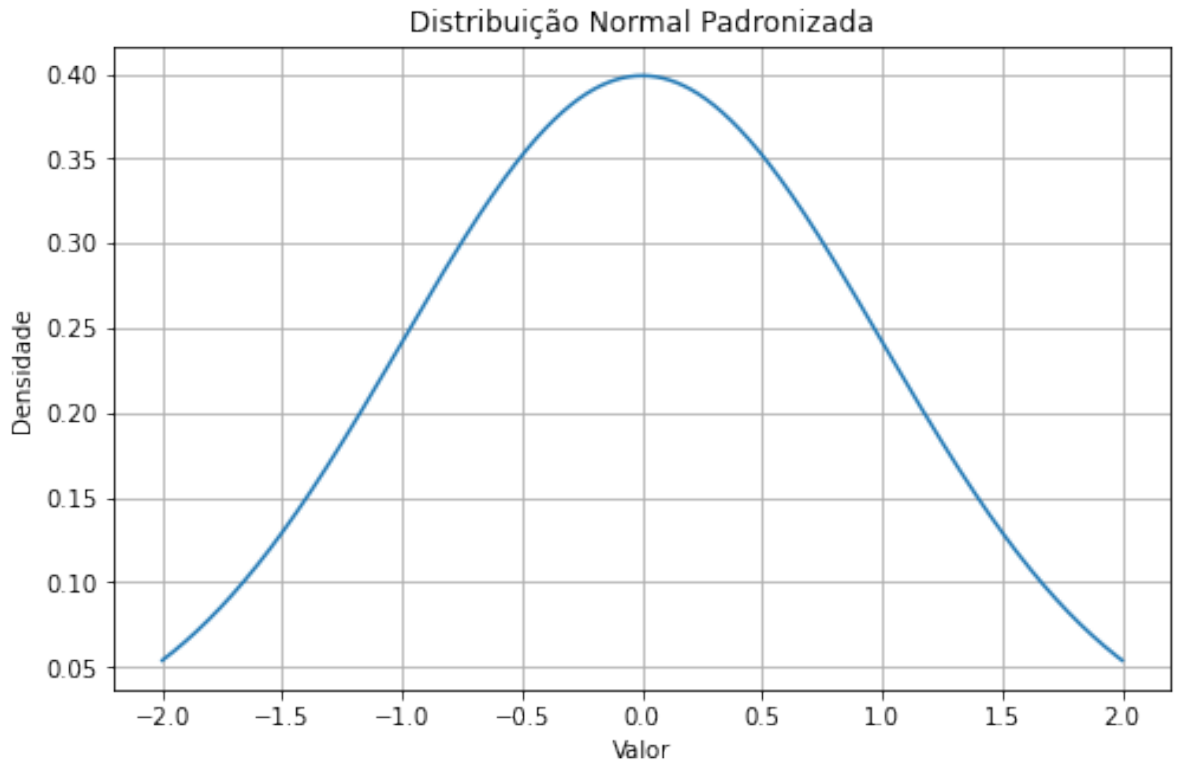

Figura 9: Curva normal

Podemos alterar a forma da curva em sino alterando a média e o seu desvio padrão. Alterar a média deslocará a curva em direção a esse valor, isso significa que podemos alterar a posição da curva alterando o valor médio sem afetar a forma da curva.

```
fig, ax = plt.subplots(figsize = (8,5))

x = np.linspace(-10,15,100)

medias = [0.0, 2.0, 5.0, 10.0]

for medias in medias:
ax.plot(x, norm.pdf(x,loc=medias), label=f"Médias: {medias}")

ax.set_xlabel('x')
ax.set_ylabel('pdf(x)')
ax.set_title('Distribuição Normal')
ax.legend(loc='best', frameon=True)
ax.set_ylim(0,0.45)
ax.grid(True)
```

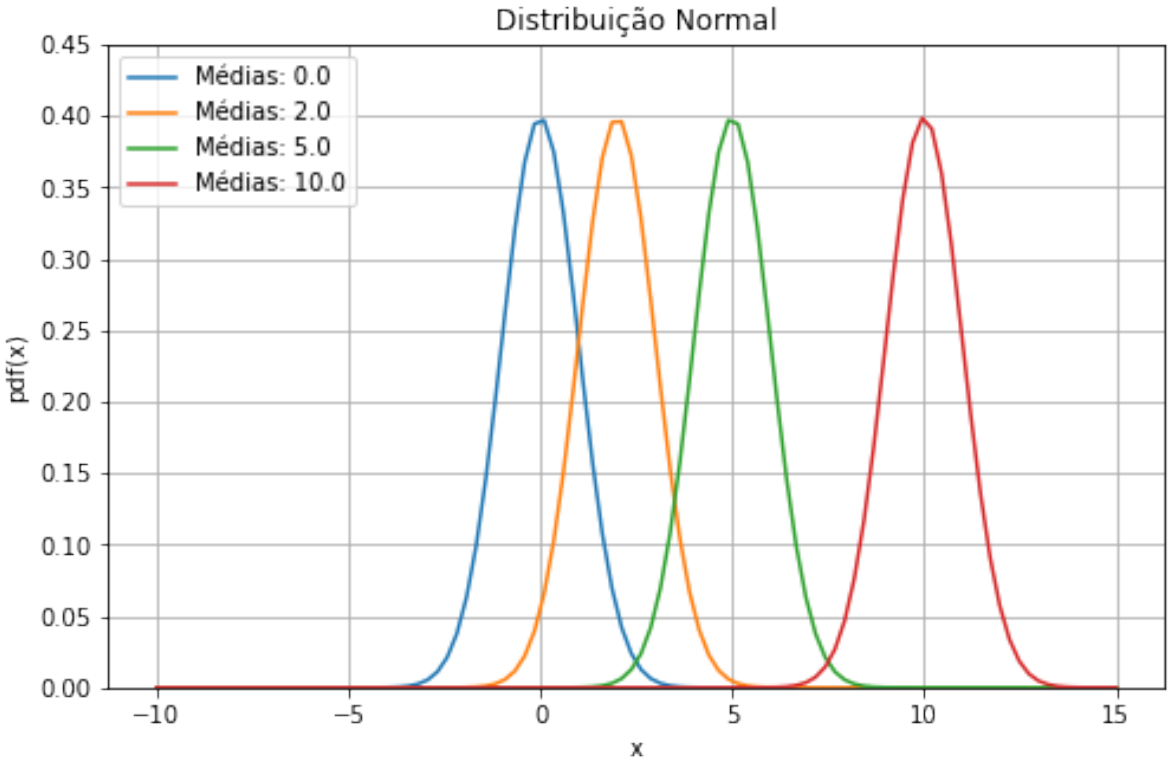

Figura 10: Alterando a média

A forma da curva pode ser controlada pelo valor do desvio padrão. Um desvio padrão menor resultará em uma curva estreitamente limitada, enquanto um valor alto resultará em uma curva mais espalhada.

```
fig, ax = plt.subplots(figsize = (8,5))

x = np.linspace(-10,10,100)

stdvs = [1.0, 2.0, 3.0, 4.0]

for s in stdvs:
ax.plot(x, norm.pdf(x, scale=s), label=f'desvio padrão = {s}')

ax.set_xlabel('x')
ax.set_ylabel('pdf(x)')
ax.set_title('Distribuição Normal')
ax.legend(loc='best', frameon=True)
ax.set_ylim(0,0.45)
ax.grid(True)
```

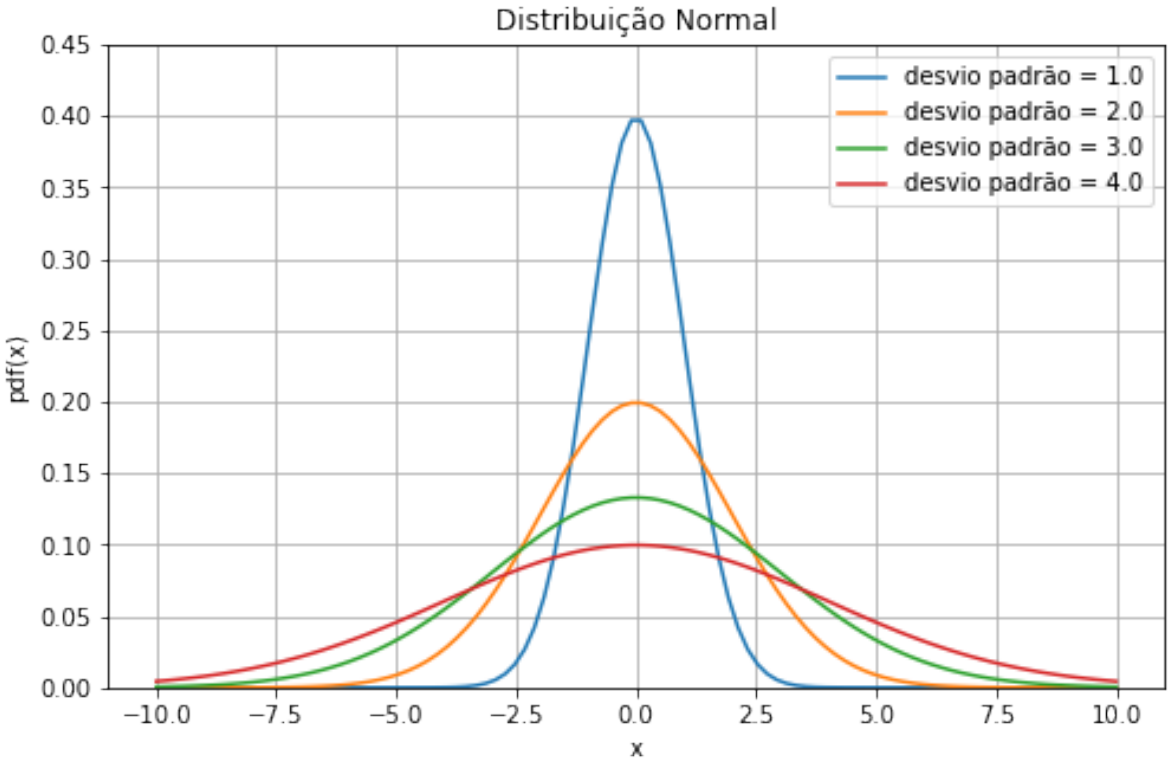

Figura 11: Alterando o desvio padrão

2.4.9 FUNÇÃO DISTRIBUIÇÃO ACUMULADA

É a função $F(x)$ que indica a probabilidade de um determinado valor de uma variável aleatória X ser menor ou igual à x. Em termos matemáticos:

$$F(x) = P(X \leq x) = \int_{-\infty}^{x} f(x_i)dx$$

2.4.9.1 CALCULANDO A PROBABILIDADE DE OCORRÊNCIA DE DADOS ESPECÍFICOS

Vamos utilizar o conceito de função distribuição acumulada (CDF) e calcular a probabilidade de um valor estar abaixo de -1 ao escolhermos um valor aleatório da distribuição. Essa probabilidade será a área da região apresentada abaixo e terá um valor de aproximadamente $15,87\%$.

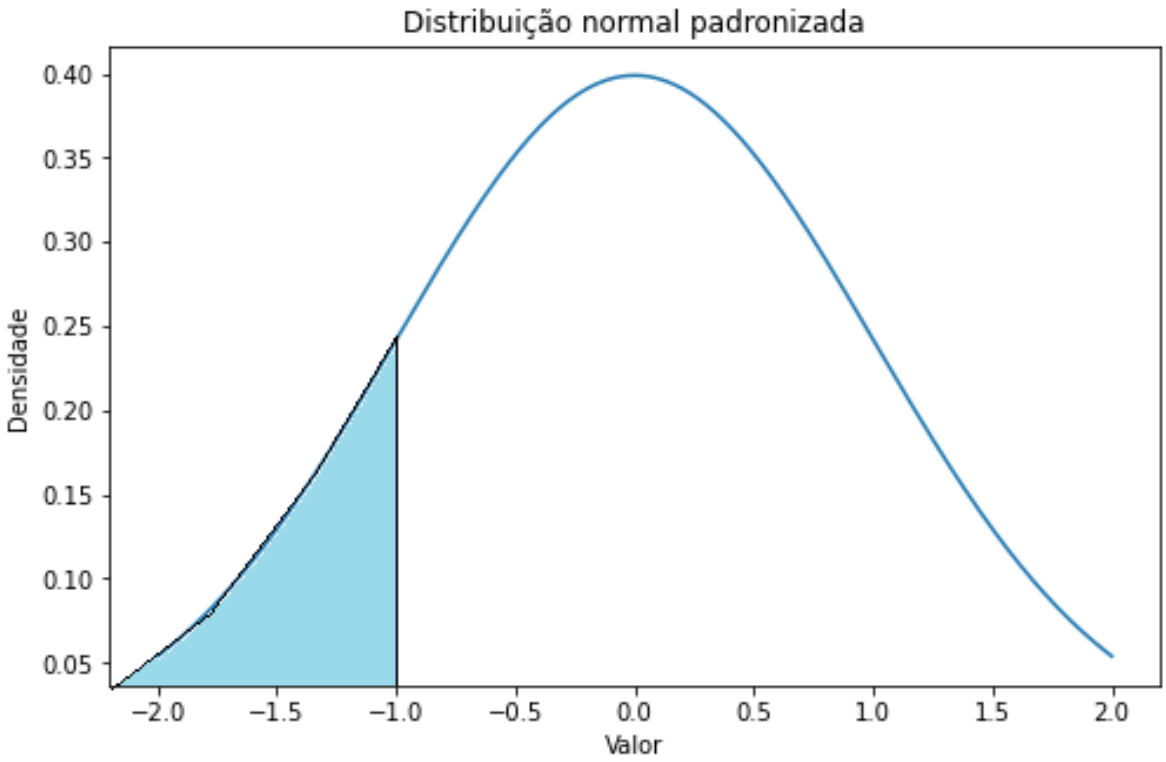

Figura 12: Abaixo de -1

```
prob = norm(loc = 0 , scale = 1).cdf(-1)
print(f"Probabilidade: {prob*100:.2f}%")
# Probabilidade: 15.87%
```

Para calcular a probabilidade do valor estar entre uma região específica (entre 2 e 1 por exemplo):

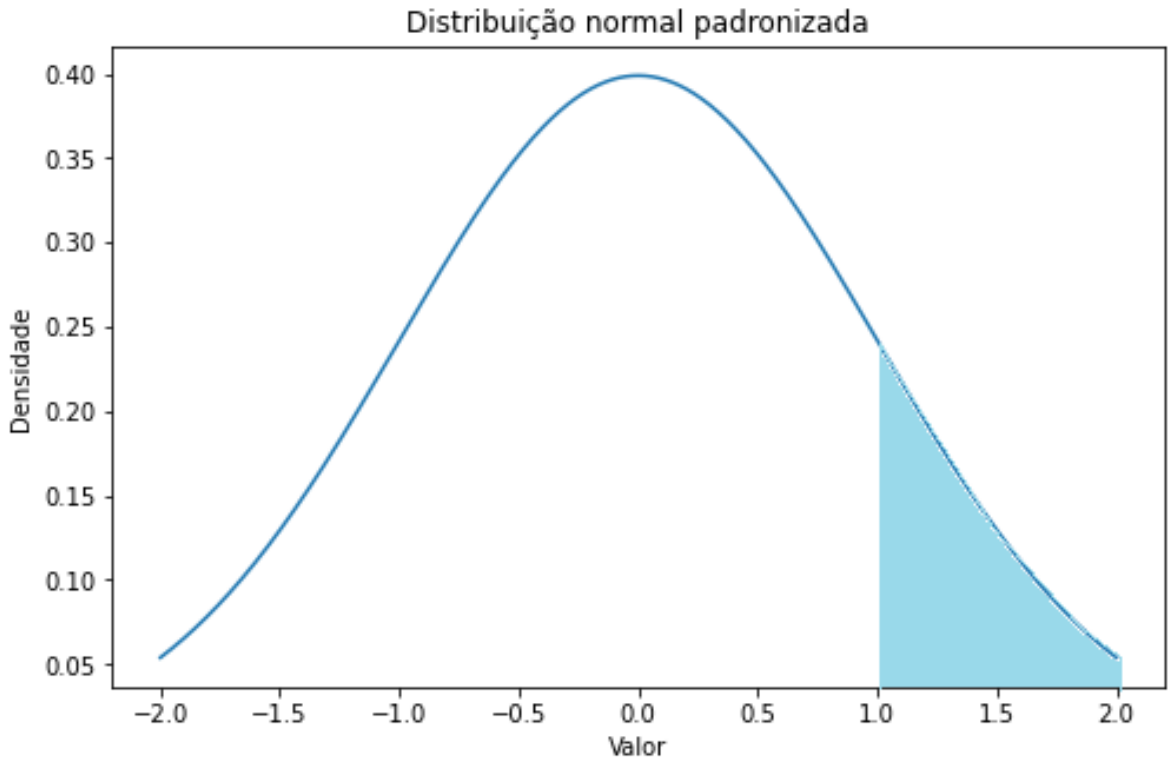

Figura 13: Entre 1 e 2

```
cdf_limite_superior = norm(loc = 0 , scale = 1).cdf(2)
cdf_limite_inferior = norm(loc = 0 , scale = 1).cdf(1)

prob = cdf_limite_superior - cdf_limite_inferior
print(f"Probabilidade: {prob*100:.2f}%")
# Probabilidade: 13.59%
```

Para calcular a probabilidade do valor ser maior que x (0.5 por exemplo):

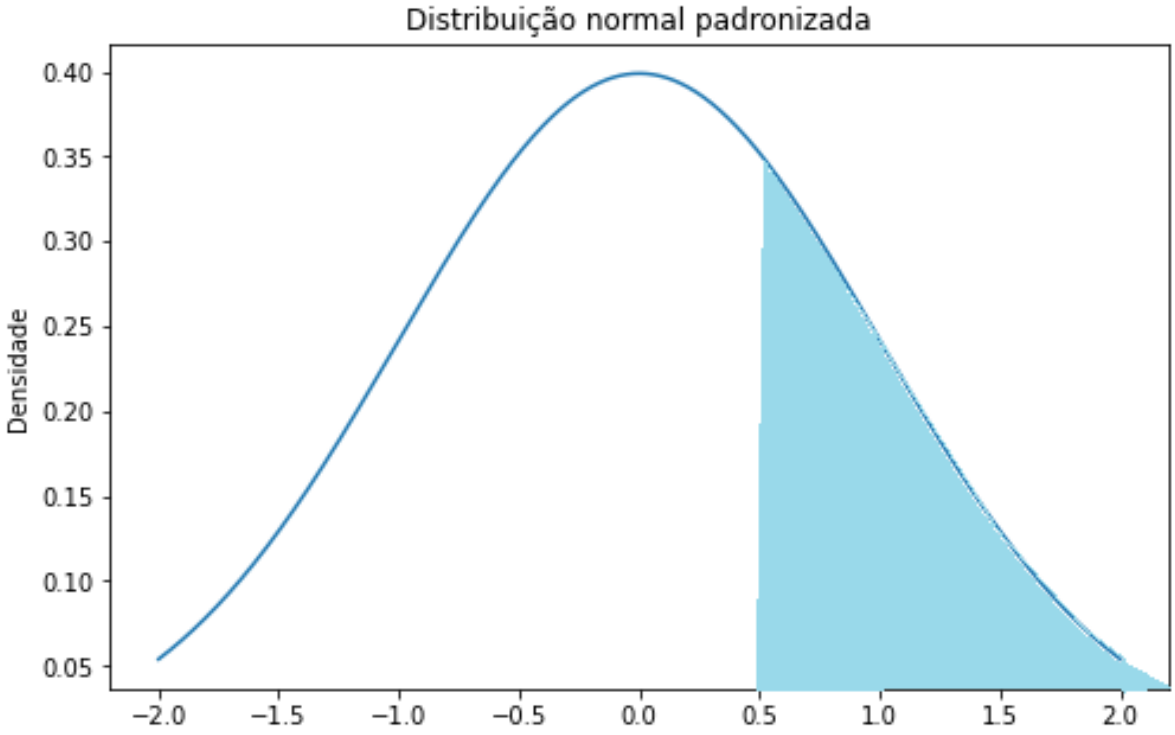

Figura 14: Maior que 0.5

```
prob = 1 - norm(loc = 0 , scale = 1).cdf(0.5)
print(f"Probabilidade: {prob*100:.2f}%")
# Probabilidade: 30.85%
```

2.4.10 APLICAÇÕES DA DISTRIBUIÇÃO NORMAL

- O matemático francês Abraham de Moivre, em seu Doctrine of Chances (1718), primeiro observou que as probabilidades associadas a variáveis aleatórias geradas discretamente (como as obtidas jogando uma moeda ou jogando um dado) podem ser aproximadas pela área sob o gráfico de uma função exponencial. Este resultado foi estendido e generalizado pelo cientista francês Pierre-Simon Laplace.
- Em 1860, Maxwell supões que a velocidade das colisões das partículas obedecem a distribuição normal.
- Pode ser utilizado na análise da resistência dos materiais, onde os dados coletados podem ser generalizados para uma distribuição normal de forma a facilitar as simulações

computacionais e torná-los mais práticos.

- Em projetos de engenharia no campo da ergonomia.

2.5 INTEGRAL DE RIEMANN

2.5.1 CONCEITO DE INTEGRAL

Em cálculo, o conceito de integral se refere à soma infinitesimal de regiões para se obter o valor total de uma região contínua. Ela pode ser considerada uma área ou a generalização de uma área e, juntamente com a derivada, constitui os conceitos fundamentais do Cálculo.

O exemplo mais simples de integral é a **Integral de Riemann**. Considerando uma função contínua $f(x)$ num intervalo $[a,b]$ tal que $f(x) \geq 0$ para todo $x \, \varepsilon \, [a,b]$, a sua curva plotada no sistema cartesiano fica:

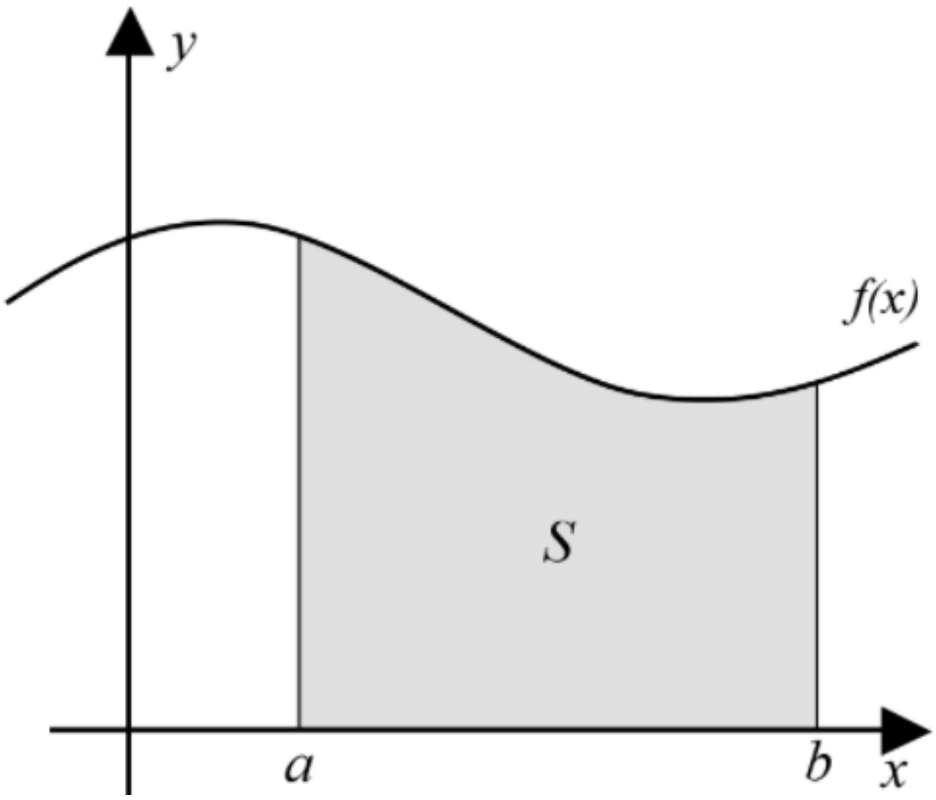

Figura 15: Área sob a curva

2.5.2 INTEGRAL DE RIEMANN

O valor total da área sob a curva da função $f(x)$ pode ser obtido dividindo essa área em diversos retângulos menores com extremidades nos pontos $[x_0, x_1, x_2, x_3, ..., x_n]$. A fórmula para a Integral de Rieman basicamente diz que a área da região compreendida entre o eixo horizontal e o gráfico da função $f(x)$, para x percorrendo o intervalo $[a, b]$, é igual ao limite da soma das áreas dos *n* retângulos, quando o número desses retângulos tende a infinito.

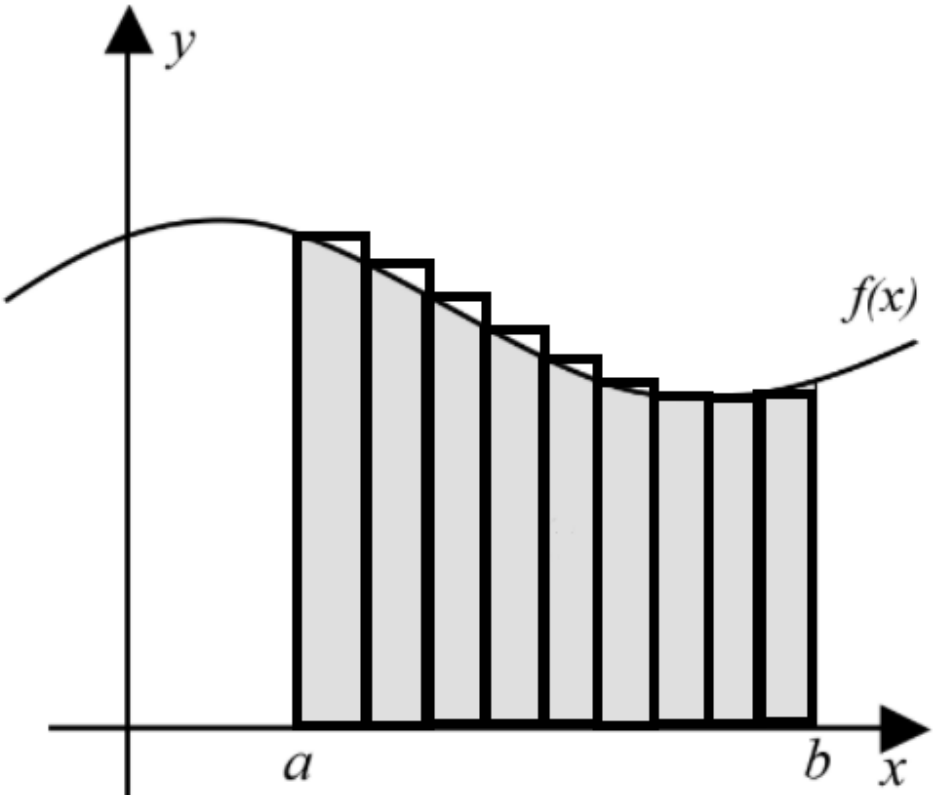

Figura 16: Divisão da área

2.5.2.1 FÓRMULA

A Integral de Riemann da função $f(x)$ em relação a x de a para b é:

$$\int_a^b f(x)\,dx = \lim_{n\to\infty} \sum_{i=1}^{n} \frac{b-a}{n} f(x_{i-1})$$

A integral corresponde à ”área com sinal”, isto é, a área acima do eixo x é positiva e

a área abaixo do eixo x é negativa.

Quando ela possui 2 limites, a integral é do tipo **definida**. Integral sem limite é denominada **indefinida**.

2.5.3 FUNÇÃO PRIMITIVA/INTEGRAL INDEFINIDA/ANTIDERIVADA

A **função primitiva/integral indefinida/antiderivada** de uma função f é uma função diferenciável F cuja derivada é igual à função original f. Isso pode ser representado simbolicamente por $F' = f$.

O processo de calcular funções primitivas (ou integrais indefinidas) de funções é oposto ao da diferenciação de funções, cujo objetivo é obter a derivada. Funções primitivas geralmente são representados por letras maiúsculas e estão relacionadas às integrais definidas pelo **Teorema Fundamental do Cálculo**.

Exemplo:

A função $F(x) = \frac{x^3}{3}$ é a função primitiva de $f(x) = x^2$ uma vez que a derivada de $F(x) = \frac{x^3}{3}$ é x^2. Uma vez que a derivada de uma constante é zero, x^2 possuirá infinitas funções primitivas da forma $F(x) = \frac{x^3}{3} + C$ onde C é denominada constante de integração.

2.5.4 TEOREMA FUNDAMENTAL DO CÁLCULO

O **Teorema Fundamental do Cálculo** diz que se $f(x)$ é uma função contínua em $[a,b]$, a sua integral definida nesse intervalo é a diferença entre as suas funções primitivas $F(x)$ nas extremidades desse intervalo:

$$\int_b^a f(x)\,dx = F(b) - F(a) = F(x)|_a^b$$

2.5.5 INTEGRAL DE RIEMANN UTILIZANDO SCIPY DO PYTHON

Vamos utilizar a função **quad** da biblioteca Scipy para realizar a integração da função abaixo:

$$\int_0^4 x^2 dx$$

Para isso, vamos utilizar a função **lambda** do Python, que permite a criação de funções com número qualquer de argumentos:

```
x2 = lambda x: x**2
print(x2)
print(type(x2))
```

Após isso, usamos a a função **integrate** da biblioteca **scipy** para realizar a integração.

O valor retornado pela função é uma tupla, onde o primeiro elemento é o valor estimado da integral e o segundo elemento é o limite superior de erro.

```
from scipy import integrate
integrate.quad(x2, 0, 4)
```

$$\int_0^4 x^2 dx \cong 21.333$$

2.5.6 EXEMPLO POR APROXIMAÇÃO SUPERIOR

Vamos resolver a mesma integral de Riemann por aproximação superior. Para isso, vamos dividir a área S em diversos retângulos de base $[x_{i-1}, x_i]$ e altura $f(x_i)$

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

fig, ax = plt.subplots(figsize =(8,6))

a = 0
b = 4
n = 5
x = np.linspace(a,b,n+1)

f = lambda x: x**2
vetor = []

for i in x:
        vetor.append(f(i))
        plt.vlines(i,0,f(i))

y = x**2
plt.plot(x,y)
plt.bar(x, vetor, align='edge', width=-(b-a)/n,
color='pink')
plt.show()
```

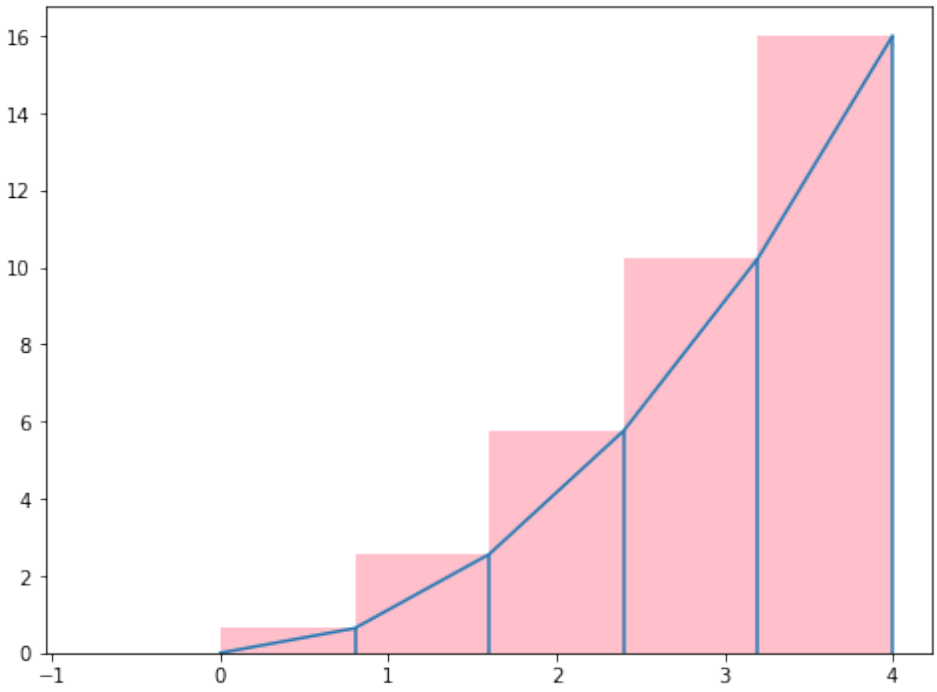

Figura 17: Aproximação superior

A área da região S será tanto mais próxima do valor real 21.333 quanto mais retângulos utilizarmos para dividir a área:

```python
import numpy as np

def area(a,b,n):
y = np.linspace(a,b,n+1)
w = (b - a)/n
f = lambda x: x**2
S = 0

for i in y[1:]:
        S = S + f(i)*w

        print(f"Área: {S:.3f} (para {n} retângulos)")

area(0,4,5)
# Área: 28.160 (para 5 retângulos)
area(0,4,100)
# Área: 21.654 (para 100 retângulos)
area(0,4,500)
# Área: 21.397 (para 500 retângulos)
area(0,4,1000)
# Área: 21.365 (para 1000 retângulos)
area(0,4,5000)
# Área: 21.340 (para 5000 retângulos)
area(0,4,10000)
# Área: 21.337 (para 10000 retângulos)
```

2.5.7 EXEMPLO POR APROXIMAÇÃO INFERIOR

Vamos resolver a mesma integral de Riemann por aproximação inferior. Para isso, vamos dividir a área S em diversos retângulos de base $[x_{i-1}, x_i]$ e altura $f(x_{i-1})$

```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize =(8,6))

a = 0
b = 4
n = 5
x = np.linspace(a,b,n+1)

f = lambda x: x**2
vetor = []
for i in x:
        vetor.append(f(i))
        plt.vlines(i,0,f(i))

y = x**2
plt.plot(x,y)
plt.bar(x[:-1], vetor[:-1], align='edge',
width=(b-a)/n, color='pink')
plt.show()
```

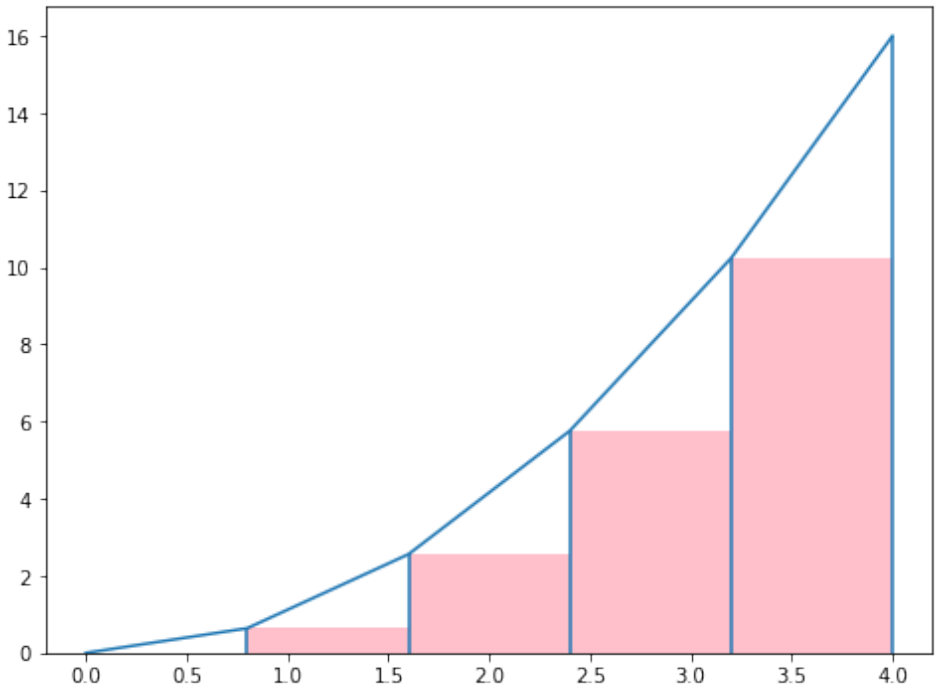

Figura 18: Aproximação inferior

A área da região S será tanto mais próxima do valor real 21.333 quanto mais retângulos utilizarmos para dividir a área:

```python
import numpy as np

def area(a,b,n):
y = np.linspace(a,b,n+1)
w = (b - a)/n
f = lambda x: x**2
S = 0

for i in y[:-1]:
        S = S + f(i)*w

        print(f"Área: {S:.3f} (para {n} retângulos)")

area(0,4,5)
# Área: 15.360 (para 5 retângulos)
area(0,4,100)
# Área: 21.014 (para 100 retângulos)
area(0,4,500)
# Área: 21.269 (para 500 retângulos)
area(0,4,1000)
# Área: 21.301 (para 1000 retângulos)
area(0,4,5000)
# Área: 21.327 (para 5000 retângulos)
area(0,4,10000)
# Área: 21.330 (para 10000 retângulos)
```

2.5.8 APLICAÇÕES DA INTEGRAL DE RIEMANN

- Pode ser utilizado na Matemática para: determinar a área de polígonos regulares ou irregulares; calcular a Transformada de Fourier; determinar o comprimento de uma curva; determinar o volume de um sólido.
- Na física, é utilizado para: determinar a massa de um objeto caso a sua densidade seja

conhecida; calcular o trabalho realizado partindo da força; calcular a velocidade e o instante dado a aceleração e as condições iniciais; calcular as equações de Maxwell.

- Na engenharia é utilizado para determinar a força cortante e momento fletor; centroide de uma área; momento de inércia de uma área;
- Na estatística é utilizado na função densidade de probabilidade.
- Na química, pode ser utilizado para determinar a força a partir da pressão provocada por um conjunto de moléculas no interior de um recipiente.

2.6 OSCILADOR HARMÔNICO AMORTECIDO

2.6.1 CONCEITOS

O oscilador harmônico amortecido descreve o movimento mecânico de um oscilador (ex: pêndulo de massa-mola) sob a influência de uma força restauradora e atrito. O movimento do pêndulo depende basicamente de 3 forças:

- Força de inércia $F = m\ddot{x}$
- Força restauradora $F_f = -kx$
- Força de amortecimento $F_r = -\mu\dot{x}$

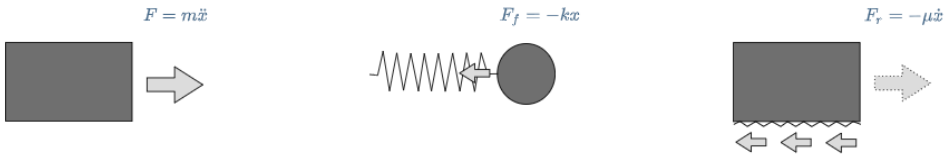

Figura 19: Tipos de forças atuantes

2.6.2 FORÇA DE INÉRCIA

Advém da 2° Lei de Newton que diz que a força resultante que atua sobre um corpo é igual ao produto de sua massa pela aceleração.

2.6.3 FORÇA RESTAURADORA

É a força que puxa a massa do pêndulo de volta para sua posição de repouso. Esta força é diretamente proporcional ao deslocamento em relação à posição de equilíbrio.

- x: é o deslocamento da posição de equilíbrio, em metros (m).
- k: é a constante de proporcionalidade, dada por $\frac{mg}{L}$ e que depende do material.

2.6.4 FORÇA DE AMORTECIMENTO

É a força de resistência ao movimento relativo entre superfícies sólidas, camadas de fluidos e outros materiais que deslizam um contra o outro. Essa força de atrito é proporcional à velocidade do pêndulo.

- $\dot{x}$: velocidade de deslocamento de uma massa em relação a um ponto fixo.
- μ: é o coeficiente de atrito que depende do material e da forma da matéria

2.6.5 EQUAÇÃO DO MOVIMENTO

A força resultante será a soma da força restauradora e de amortecimento.

$$m\ddot{x} = -\mu\dot{x} - kx \rightarrow \ddot{x} + \frac{\mu}{m}\dot{x} + \frac{k}{m}x = 0$$

Esta equação é uma equação diferencial linear, homogênea, de segunda ordem e com coeficientes constantes.

- Vamos definir $\omega_0 = \sqrt{\frac{k}{m}}$ como a frequência natural de vibração não amortecida do sistema.
- Vamos definir a sigla ζ como o **coeficiente de amortecimento** ou simplesmente **amortecimento**, sendo a sua fórmula:

$$\zeta = \frac{\mu}{2\sqrt{km}}$$

- Vamos considerar ainda $p = \frac{dx}{dt}$.

A equação do movimento, fica, então:

$$\frac{dp}{dt} = -2\zeta w_0 p - w_0^2 x$$

O termo $-2\zeta w_0$ é responsável pelo amortecimento. Caso o coeficiente de amortecimento $-2\zeta w_0$ seja nulo, o sistema passa a oscilar indefinidamente, realizando um **Movimento Harmônico Simples (MHS)**.

2.6.6 AMORTECIMENTO

- O amortecimento é uma influência dentro ou sobre um sistema oscilatório que tem o efeito de reduzir, restringir ou prevenir suas oscilações.
- O **coeficiente de amortecimento** é uma medida adimensional que descreve como as oscilações em um sistema decaem após uma perturbação. Muitos sistemas exibem comportamento oscilatório quando são perturbados de sua posição de equilíbrio estático.
- Uma massa suspensa por uma mola pode, se puxada e liberada, pular para cima e para baixo. O sistema tende a retornar à sua posição de equilíbrio a cada salto, mas a ultrapassa. Às vezes, as perdas (por exemplo, atrito) amortecem o sistema e podem fazer com que as oscilações diminuam gradualmente em amplitude para zero ou se atenuem. O **coeficiente de amortecimento** é uma medida que descreve a rapidez com que as oscilações diminuem de um salto para o outro.

2.6.6.1 FÓRMULA DO AMORTECIMENTO

O coeficiente de amortecimento é geralmente representado pela sigla ζ e sua fórmula é:

$$\zeta = \frac{\mu}{2\sqrt{km}}$$

2.6.6.2 TIPOS DE AMORTECIMENTO

O coeficiente de amortecimento (ζ) fornece indicações de como será a resposta transitória do sistema:

- Se $\zeta > 1$: sistema superamortecido (overdamped). O sistema retorna (decai exponencialmente) para o estado estável sem oscilar.
- Se $\zeta = 1$: sistema criticamente amortecido (critically damped). O sistema retorna para o estado estável rapidamente e sem oscilar.
- Se $0 < \zeta < 1$: sistema subamortecido (underdamped). O sistema oscila com uma freqüência levemente diferente que o do caso não amortecido com a amplitude gradualmente decrescendo a zero.
- Se $\zeta = 0$: sistema não amortecido. O sistema oscila sem perda de amplitude.

2.6.7 OSCILADOR HARMÔNICO AMORTECIDO NO PYTHON

O gráfico da posição vs tempo de um oscilador harmônico foi obtido com o uso da biblioteca **odeint** do Python, que possibilita o usuário a encontrar as soluções de um sistema de equações diferenciais ordinárias.

```
from scipy . integrate import odeint
import numpy as np
from matplotlib import pyplot as plt

def dy(y, t, zeta, w0):
x, p = y[0], y[1]
dx = p
dp = -2 * zeta * w0 * p - w0**2 * x
return [dx, dp]

# estado inicial
y0 = [1.0, 0.0]

# intervalo de tempo
t = np.linspace(0, 10, 1000)
w0 = 2*np.pi*1.0 # frequência natural
# print(w0)

# resolvendo a EDO para 4 diferentes
# valores de amortecimento
# não amortecido
y1 = odeint(dy, y0, t, args=(0.0, w0))
# sub-amortecido
y2 = odeint(dy, y0, t, args=(0.2, w0))
# criticamente amortecido
y3 = odeint(dy, y0, t, args=(1.0, w0))
# super-amortecido
y4 = odeint(dy, y0, t, args=(5.0, w0))

fig, ax = plt.subplots(figsize = (10,8))
ax.plot(t, y1[:,0], 'k', label="não amortecido",
linewidth=0.5)
ax.plot(t, y2[:,0], 'r', label="sub-amortecido")
ax.plot(t, y3[:,0], 'b', label=r"criticamente amortecido")
ax.plot(t, y4[:,0], 'g', label="super-amortecido")
ax.legend()
plt.show()
```

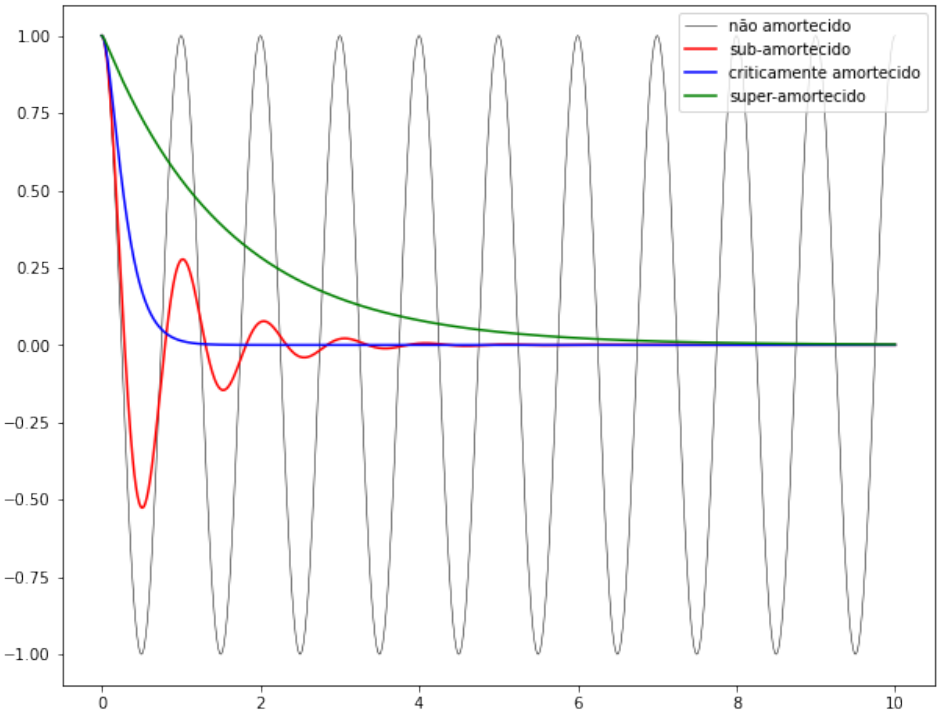

Figura 20: Gráfico posição x tempo

2.6.8 APLICAÇÕES DA OSCILAÇÃO HARMÔNICA AMORTECIDA

- Em sistemas físicos, o amortecimento é produzido por processos que dissipam a energia armazenada na oscilação. Os exemplos incluem arrasto viscoso em sistemas mecânicos, resistência em osciladores eletrônicos e absorção e dispersão de luz em osciladores ópticos.
- Na engenharia elétrica, as repostas naturais de circuitos RLC podem ser super-amortecido, criticamente amortecido, sub-amortecido ou sem amortecimento.
- Os amortecedores de um carro amortecem criticamente a suspensão do veículo e, portanto, resistem à vibrações que poderiam dificultar o controle ou causar danos.
- Amortecedores de porta evitam que a porta oscile quando ela é fechada graças à esse princípio.

3 CONCLUSÃO

3.1 CONCLUSÕES

A partir do desenvolvimento de exemplos que abrangem as mais diversas áreas de ciência utilizando a linguagem Python, percebeu-se o quão prático é o uso dessa ferramenta no ensino e apresentação visual dos conceitos fundamentais. A disponibilidade de uma extensa gama de bibliotecas que possibilita a sua aplicação em diversos ramos do conhecimento mostrou que o uso dessa linguagem de programação é bastante conveniente na elaboração de materiais educacionais.

3.2 TRABALHOS FUTUROS

Espera-se para os trabalhos futuros um aumento gradativo na complexidade dos exemplos escolhidos, utilizando os conceitos básicos e fundamentais deste trabalho.